On Neuromorphic Computing: A Case Study on Radio Resource Allocation with LAVA Software Framework

Ajay Kumar Lakshmipura Vijaykumar

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

March 2023

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Ajay Kumar Lakshmipura Vijaykumar

Entitled: On Neuromorphic Computing: A Case Study on Radio Resource Allocation with LAVA Software Framework

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair
Dr. Anjali Agarwal

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Anjali Agarwal

_____ Supervisor
Dr. Roch Glitho

Approved by _____
Dr. Yousef Shayan, Chair
Department of Electrical and Computer Engineering

_____ 2023

_____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

ABSTRACT

On Neuromorphic Computing:
A Case Study on Radio Resource Allocation with LAVA Software Framework

Ajay Kumar Lakshmipura Vijaykumar

Neuromorphic computing is a neuro-inspired computing gaining traction because of its ability to perform complex calculations faster, with greater energy efficiency, and on a smaller footprint compared to traditional Von Neumann architectures. Due to how fundamentally different their architecture is from the Von Neumann architecture, there are currently significant uncertainties over how we program and use neuromorphic chips. When neuromorphic chip implementation is linked with the implementations employing emerging device technologies, additional challenges related to programming devices are introduced.

To address these issues, neuromorphic frameworks with the abstractions and tools to develop applications might prove beneficial. Lava framework is an open-source neuromorphic framework designed by Intel Labs to build applications that fully exploit the principles of neural computation and map them to neuromorphic hardware. The Lava framework includes high-level libraries for deep learning, dynamic neural fields, and constrained optimization for productive algorithm development. It also consists of tools to map those algorithms to different types of hardware architectures. Lava is the only existing neuromorphic framework that comes with specifically designed optimization solvers. Therefore, we have selected the Lava framework for our resource allocation problem implementation.

This thesis aims to conduct a case study on the use of Lava neuromorphic framework for radio resource allocation. First, we design and formulate the problem as an integer linear program (ILP). Then, it is reduced to a quadratic unconstrained binary optimization (QUBO), a format that can be solved using Lava. To evaluate the performance of the proposed Lava-based approach, we compare our results to the ones obtained with classical CPU-based solvers. Solving problems on Lava requires several hyperparameters as inputs. Currently, tuning the hyperparameters is a tedious task. Determining the optimal set of hyperparameters is even more difficult for large problem instances. We believe that improving the Solver-Tuner utility for hyperparameter tuning can help solve large problems.

Acknowledgements

First, I sincerely thank my supervisor, Dr. Roch Glitho, for whose guidance and constant support I would have needed to reach this far. I wholeheartedly thank him for accepting me as his student and helping me stay on track and progress well by providing me with his expertise. His patience, knowledge, and motivation helped me enhance my abilities and overcome challenging tasks that seemed impossible initially.

I also express my sincere gratitude to Dr. Juergen Rilling and Dr. Anjali Agarwal for serving as my thesis committee members. I would also like to thank Dr. Anjali Agarwal for serving as the chair of my thesis defense.

I would also like to thank my colleagues at Concordia University for their guidance, help, and encouragement. I especially want to thank Amina Hentati for helping me and guiding me at every stage of the thesis. I would also like to thank Daniel Gehberger for providing valuable advice whenever I faced issues. I thank my friends Dhanush and Pradnya Thakar for motivating me and boosting my confidence.

Finally, I am forever indebted to my parents, Vijay Kumar L S and Radhika S, and my sisters Brunda and Vennela, without whose constant motivation, encouragement, and support I would not have reached this far in life and accomplished so much. I can never thank them enough for supporting me throughout my journey and always believing in me incessantly.

Table of Contents

List of Figures

List of Tables

Acronyms and abbreviations

3GPP 3rd Generation Partnership Project

ALU Arithmetic-Logic Unit

AMC Adaptive Modulation and Coding

API Application Programming Interface

APs Access Points

ARM Advanced RISC Machine

ASIC Application-Specific Integrated Circuit

BS Base Station

CDMA Code-Division Multiple Access

CPU Central Processing Unit

CSI Channel State Information

CSP Constraint Satisfaction Problem

DSA D-Wave's Simulated Annealing

EE Energy Efficiency

eMBB enhanced Mobile Broadband

FDM Frequency Division Multiplexing

FFT Fast Fourier Transform

FPAA Field programmable analog arrays

FPGA Field Programmable Gate Array

FPNA Field Programmable Neural Array

OFDM Orthogonal Frequency Division Multiplexing

OFDMA Orthogonal Frequency-Division Multiple Access

ILP Integer Linear Programming

LIF Leaky Integrate and Fire

LQS Lava QUBO solver

MAC Media Access Control

MAI Multiple Access Interference

MILP Mixed-Integer Linear Programming

MIMO Multiple-Input Multiple-Output

MIQP Mixed-Integer Quadratic Programming

NEF Neural Engineering Framework

NGN New Generation Radio Mobile Networks

NR New Radio

NSCS Neurosynaptic Simulator for Corelet System

PAPR Peak-to-Average Power Ratio

PDF Proactive Downlink Transmission Framework

PON Passive Optical Network

QAM Quadrature Amplitude Modulation

QUBO Quadratic Unconstrained Binary Optimization

QP Quadratic Programming

QSA Qubovert's Simulated Annealing

UAV Unmanned Aerial Vehicle

UE User Equipment

URLLC Ultra-Reliable Low Latency Communications

QOS Quality-Of-Service

RBs Radio Blocks

RISC Reduced Instruction Set Computer

RNN Recurrent Neural Networks

RRUs Radio Resource Units

ROS Robot Operating System

SNN Spiking Neural Network

STDP Spike-Timing-Dependent Plasticity

TDM Time Division Multiplexing

TDMA Time Division Multiple Access

TS Tabu Search

TTI Transmission Time Interval

VLSI Very Large-Scale Integration

WLANs Wireless Local Area Networks

WMANs Wireless Metropolitan Area Networks

WRANs Wireless Regional Area Networks

Chapter 1

1. Introduction

This chapter starts with the definitions of the key concepts which are related to this thesis. These definitions are then followed by a description of the motivation and problem statement, and the contributions of this thesis. The last section of this chapter consists of a summary of how this thesis is organized.

1.1 Definition

The terms defined in this section are Neuromorphic computing, Lava, OFDMA (orthogonal frequency-division multiple access), Optimization- ILP (integer linear programming) and QUBO (quadratic unconstrained binary optimization). These terms are the ones that are most important to this thesis.

1.1.1 Neuromorphic Computing

Neuromorphic computing has come to refer to a *"variety of brain-inspired computers, devices, and models that contrast the pervasive Von Neumann computer architecture"* [1]. Neuromorphic computing is a neuro-inspired computing paradigm. Its driving force is its ability to achieve orders of magnitude gains in energy efficiency and performance compared to conventional architectures such as the Von Neumann model [2]. Carver Mead first used the term neuromorphic computing in 1990 [1]. Mead at the time referred to "neuromorphic" systems as very large-scale integration (VLSI) with analog components that resembled biological neural systems [1]. In more recent times, the phrase has expanded to include implementations based

on biologically inspired or artificial neural networks in or using non-Von Neumann architectures. These neuromorphic architectures stand out for their high degree of connectivity and parallelism, low power requirements, and integration of memory and processing [1].

1.1.2 Lava Framework

"Lava is a Software framework to develop applications for neuromorphic hardware architectures" [3]. Lava is an open-source library that has been developed by Intel Labs [3]. The Lava software framework satisfies the demand for a standard software framework in the field of neuromorphic research. Lava, which is an open, modular, and extensible framework, will enable academics and application developers to build on one another's advancements and come up with a shared set of tools, processes, and libraries. To encourage wider adoption of neuromorphic technologies, the Lava software framework combines micro and macro-scale architectural components in a coherent, approachable, and effective manner [3].

1.1.3 OFDMA:

Orthogonal Frequency Division Multiplexing (OFDM) is a *"multiplexing technique that subdivides the available bandwidth into multiple orthogonal frequency sub-carriers"* [4]. OFDM has been widely used for wideband communication standards because of its efficiency and robustness to multipath propagations. OFDM combines the benefits of different modulation techniques namely Quadrature Amplitude Modulation (QAM), Time Division Multiplexing (TDM) and Frequency Division Multiplexing (FDM) to produce a high data-rate communication system. Orthogonal frequency division multiple access (OFDMA) is an extension of OFDM digital modulation technology into a multiuser environment. OFDMA is a *"multiple access/ multiplexing*

scheme that provides multiplexing operation of user data streams onto the downlink sub-channels and uplink multiple access by means of uplink sub-channels" [4].

1.1.4 Optimization:

The goal of optimization is to select the optimal solution out of all feasible solutions. According to a more comprehensive perspective, an optimization problem involves either maximization or minimization of a real function by systematically selecting input values from within an allowed set and determining the value of the function. A significant field of applied mathematics is the extension of optimization theory and methods to new formulations. In a broader sense, optimization entails determining the "best available" values of an objective function given a specified domain (or input), considering a wide range of different types of objective functions and different types of domains [5].

1.1.4.1 Integer Linear Programming

An integer programming problem is a mathematical optimization program in which some or all the variables are restricted to integers. Integer programming is frequently used in conjunction with integer linear programming (ILP), in which the objective function and all other constraints (aside from the integer constraints) are linear [6].

An integer linear program in the canonical form is expressed as

Maximize/ minimize: $c^T \boldsymbol{x}$

Subject to $\mathrm{A}\boldsymbol{x} \leq b,$

$\boldsymbol{x} \geq 0,$

and $\boldsymbol{x} \in \mathbb{Z}^{\boldsymbol{n}}$

1.1.4.2 Quadratic Unconstrained Binary Optimization

In recent years, an unprecedented range of significant Combinatorial Optimization problems has been embraced by a mathematical formulation known as QUBO, an acronym for a Quadratic Unconstrained Binary Optimization problem. The QUBO model has become a focus of research in neuromorphic computing and has developed as a foundation for the quantum computing field known as quantum annealing and Fujitsu's digital annealing [7].

QUBO model is expressed by the optimization problem:

minimize/maximize $\boldsymbol{x^T}\mathrm{Q}\boldsymbol{x}$

where x is a vector of binary decision variables and Q is a square matrix of constants.

1.2 Motivation and Problem Statement

In this section, we present the motivation and problem statement of this thesis.

1.2.1 Motivation

Some problems, such as resource allocation problems (e.g., scheduling problems) that occur in wireless communication systems, are highly complex due to the dynamic and uncertain nature of the wireless environment and the size of the considered systems. Solving these problems on traditional computer architectures is becoming increasingly challenging. This is particularly demanding when it comes to satisfying the real-time constraints of these systems such as the transmission time interval (TTI) of 1 millisecond or less. Neuromorphic computing offers a promising alternative approach to address these complex problems. It is inspired by both the structure and the function of the human brain. Neuromorphic architectures have the potential to

achieve orders of magnitude gains in energy efficiency and performance compared to traditional architectures. This makes them well-suited for solving computationally complex problems.

There is a growing interest in using neuromorphic computation frameworks for wireless communication problems, and research in this area is ongoing. However, previous works did not investigate the potential benefits and limitations of using neuromorphic software frameworks to solve radio resource allocation problems. This thesis aims to fill this gap by investigating the use of the Lava neuromorphic framework for radio resource allocation and evaluating its performance in comparison to traditional approaches. This thesis aims to contribute to the understanding of the potential of neuromorphic computing software to solve optimization problems. In addition, this thesis provides insights into the use of neuromorphic computing framework to solve practical optimization problems related to wireless communication systems by highlighting the advantages of using the Lava framework to solve such problems, as well as the challenges that still remain to be solved.

1.2.2 Problem Statement

The main objective of this thesis is to investigate the potential of using the Lava neuromorphic framework for radio resource allocation problems in OFDMA systems. Specifically, the thesis focuses on the design of a radio resource allocation problem and its formulation as an optimization problem. The formulated problem is then solved using Lava's QUBO solver. To evaluate the performance of the Lava-based solution, the obtained performance results are compared to the ones obtained using traditional solvers.

The research question that this thesis aims to answer is: Can the Lava neuromorphic framework offer a more practical and efficient solution with gains in performance for radio resource allocation in wireless communication systems?

To answer this query, the thesis performs a case study on radio resource allocation in OFDMA systems using the Lava neuromorphic framework and assess its performance against classical solvers. The results of the case study will provide insights into the potential of the Lava neuromorphic framework for radio resource allocation in OFDMA systems and will guide future research in this field.

1.3 Thesis Contributions

The contributions made by this thesis are as follows:

- Review of state of art works on Neuromorphic software frameworks and radio resource allocation in the OFDMA system.
- Design and formulation of radio resource allocation problem as an ILP optimization problem in OFDMA system.
- Implementation and execution of the formulated ILP problem on MATLAB.
- Conversion of the resource allocation problem from ILP to QUBO Formulation.
- QUBO problem implementation and execution on both Classical QUBO solvers and Lava QUBO optimization solver.
- Performance evaluation on MATLAB, classical solvers, and Lava QUBO solver.

1.4 Thesis Organization

The remainder of this paper is organized as follows:

Chapter 2 provides a brief background on Neuromorphic Computing and its Software Platforms.

Chapter 3 presents the related works on Neuromorphic Computing Software and OFDMA Resource Allocation.

Chapter 4 presents the System model, Problem Formulation and Evaluation.

Chapter 5 Concludes the paper and presents Future Research Avenues.

Chapter 2

2. Neuromorphic Computing and its Software Platforms

The background concepts pertinent to the thesis's research domain are presented in this chapter. First, we present an overview of neuromorphic computing. This is followed by a detailed description of the Lava Software framework because it is the framework which is used in the thesis. Then, we present other neuromorphic software platforms and summarize the differences between them and Lava framework. Finally, we summarize the chapter.

2.1 Neuromorphic Computing

In this section, we present an overview of neuromorphic computing. First, we present a general definition and history of neuromorphic computing. This is followed by a presentation of the architecture and the different types of neuromorphic systems.

2.1.1 General Definition and History

Neuromorphic computing is a neuro-inspired computing paradigm. It refers to a "variety of brain-inspired computers, devices, and models that contrast with the ubiquitous Von Neumann computer architecture" [1]. The driving force behind neuromorphic computing is its ability to achieve orders of magnitude gains in energy efficiency and performance over conventional architectures such as the Von Neumann model [2]. Neuromorphic computing aims to design chips that are closely inspired by the structure and functioning of biological neural circuits [2]. These chips will thus be able to process new information, adapt, behave, and learn in real-time while consuming less energy [2]. Recently, neuromorphic computing has emerged as an alternative

architecture to Von Neumann computers. The term "neuromorphic computing" was first coined by Carver Mead in 1990 [1]. At the time, Mead referred to "neuromorphic" systems as very large-scale integration (VLSI) systems with analog components that resembled biological neural systems [1]. The term was later expanded to include implementations based on artificial or biologically inspired neural networks in or using non-Von Neumann architectures. These neuromorphic architectures are distinguished by their high degree of connectivity and parallelism, low power requirements, and integration of memory and processing. Although fascinating, neuromorphic designs have attracted more attention due to the impending end of Moore's Law, increasing power requirements due to Dennard scaling, and the Von Neumann bottleneck, or low bandwidth between the central processing unit (CPU) and memory [1]. Compared to conventional Von Neumann designs, neuromorphic computers can perform complex computations faster, more energy efficient, and with a smaller physical footprint.

2.1.2 Architecture:

In this sub-section, we first present the architecture of traditional computers, i.e., Von Neumann's architecture. Secondly, we will briefly discuss both ideal and practical neuromorphic computing architectures. Finally, we summarize the comparison between these architectures.

2.1.2.1 Von Neumann Architecture:

Modern general-purpose computers are based on the Von Neumann architecture in which computation and storage are physically separated. It is shown in Figure 1. CPU, memory, and input/output (I/O) devices make up the majority of Von Neumann architectures. Bus systems are used to connect various devices. Data and programs are successively fetched from memory by the CPU. Such an architecture experiences the well-known Von Neumann bottleneck, in which the

speed of data transfer between the CPU and memory limits the system's performance. The massive concurrency and parallelism found in biological neural systems cannot be provided by Von Neumann's architecture. Even though Von Neumann's architecture offers considerable flexibility, it still makes it unsuitable for neuromorphic computing [8].

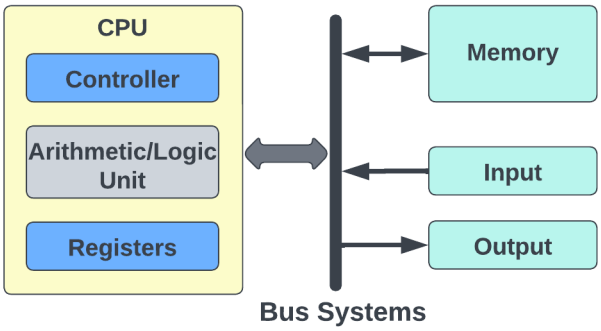

Figure 1 Von Neumann Architecture [8]

2.1.2.2 Ideal Neuromorphic Computing Architecture

An ideal neuromorphic architecture maintains one processing unit for each neuron, similar to the biological system. The ideal architecture is not scalable for large neural networks and causes large circuit overhead [8]. Each neuron and synapse in a biological system has its unique state, which may be characterized by a collection of parameters and variables in software/algorithm models. These parameters or variables are concurrently updated locally and are not shared among different neurons or synapses. The communication between the neurons is also a parallel process through a massive number of synapses. According to the above observations, an ideal design should provide 1) Local and dedicated data storage, 2) Massive concurrency, and 3) High connection density. Based on the aforementioned specifications, Figure 2 depicts the ideal architecture of digital neuromorphic hardware, where each processing unit and its local memory serve to represent a

single neuron, and the local arithmetic/logic unit (ALU) is in charge of updating neuron status. Distributed memory facilitates parallel computation, while close-to-memory computing lowers the latency of data retrieval. The number of synaptic operations per second, a crucial indicator of the performance of neuromorphic technology, is maximized by this ideal architecture [8].

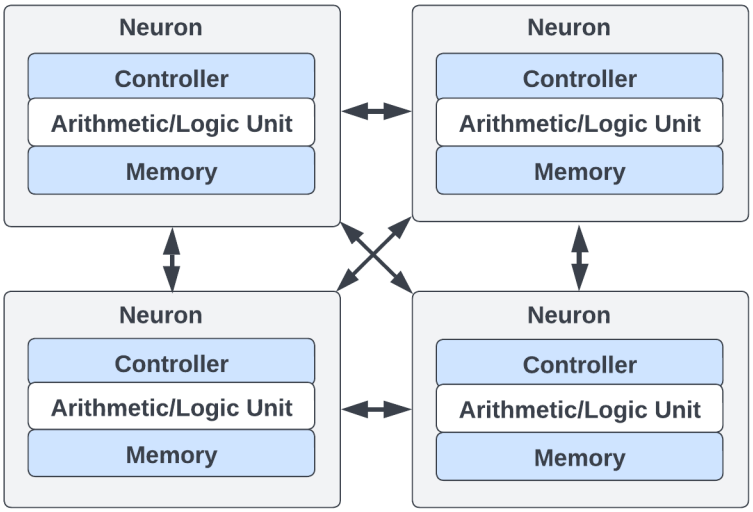

Figure 2 Ideal Neuromorphic Architecture [8]

2.1.2.3 Practical Multi-Core Neuromorphic Computing Architecture

In practical neuromorphic architecture, multiple neurons are grouped into a single unit called a core, unlike the ideal neuromorphic architecture [8]. When the neural network's scale grows, the ideal neuromorphic architecture, which retains one processing unit for each neuron, is not scalable. For large-scale SNNs, the significant circuit overhead caused by an ALU assigned to each neuron

and hardwiring the neurons to one another is highly impractical. As demonstrated in Figure 3, a practical solution is to group several neurons into a core [8].

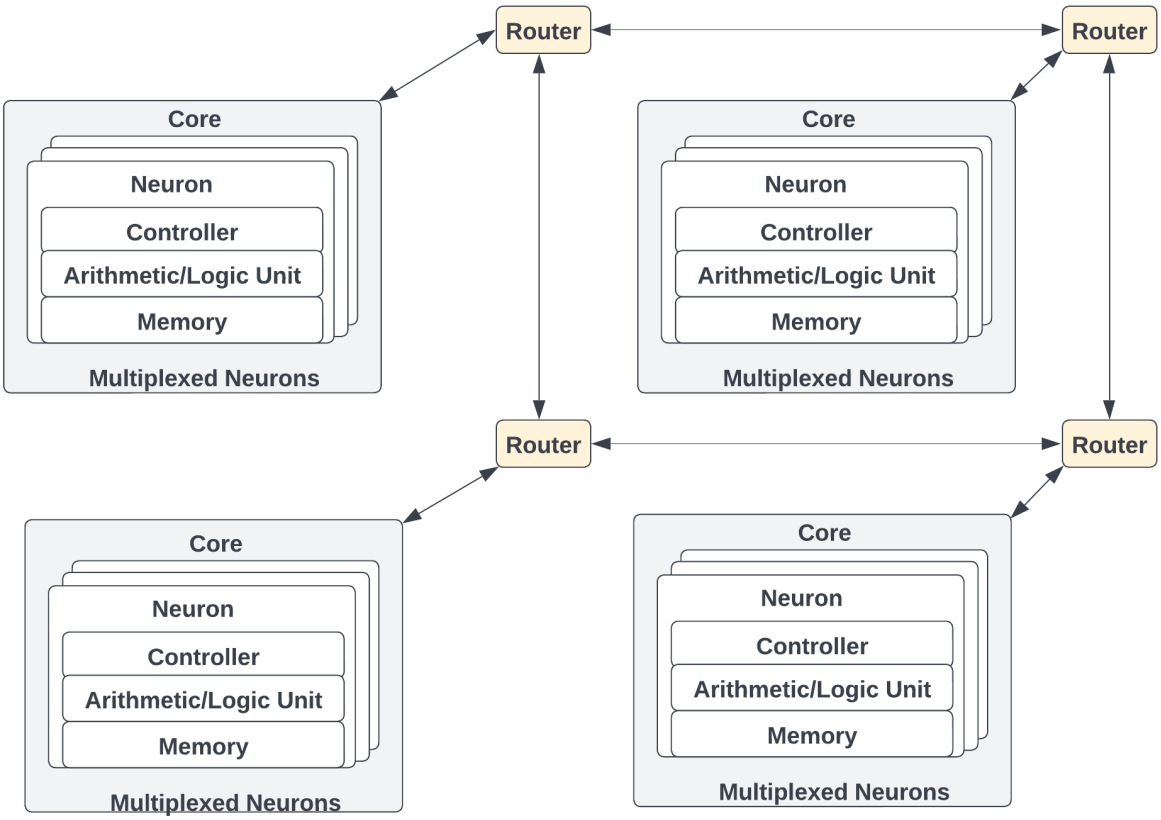

Figure 3 Practical Neuromorphic Architecture [8]

Although these neurons share the same data channel to update neuron and synapse status, they each have their own local data. This results in a considerable reduction of the effective circuit area per neuron as compared to the ideal neuromorphic architecture. The cores also allow the neurons to share common parameters for more efficient memory use. However, the same ALU updates the status of the neurons and synapses associated with a core using time multiplexing. Since only one neuron may use the ALU at a time, this lowers parallelism. Furthermore, this adds spike delays (or delays in neuron update) that may cause errors in neuronal encoding.

2.1.2.4 Comparison Between Different Architectures

In this sub-section, we consider three important factors such as scalability, cost, and performance to compare between above discussed architectures. The definitions of these three factors are as follows. *Scalability* is defined as the capacity of a system to support growth and manage increasing workloads over time. The financial resources needed to implement, run, and maintain a system are referred to as its cost. Performance is the efficiency and responsiveness of a system, and it may be assessed using metrics like response time, throughput, and resource use.

Table 1 displays comparison between the aforementioned three architectures [8]. In the case of Von Neumann architecture, scalability is high, and cost is low. However, the system's performance is very low because of Von Neumann bottleneck. With ideal neuromorphic architecture, the system can achieve very high performance. However, the manufacturing cost is high, and scalability is low for the ideal neuromorphic architecture. In the case of practical neuromorphic architecture, we can achieve high scalability with moderate manufacturing cost. In addition, it outperforms the Von Neumann architecture. On the other hand, its performance is lower than ideal neuromorphic architecture.

Table 1. Comparison Between Different Architectures [8]

Comparison Between Different Architectures			
Architecture	**Scalability**	**Cost**	**Performance**
Von Neumann	High	Low	Very low
Ideal	Low	High	Very High
Practical	High	Moderate	High

2.1.3 Types of Neuromorphic Systems:

In this sub-section, we present the types of neuromorphic systems.

Neuromorphic systems can be classified into three categories based on their implementation choices, (1) digital, (2) analog, or (3) mixed-signal platform [8].

2.1.3.1 Digital Neuromorphic Systems

The implementations of digital neuromorphic systems can be further broken down into CPU-based, application-specific integrated circuit (ASIC)-based, and field-programmable gate array (FPGA)-based ones. SpiNNaker is an illustration of CPU-based implementation. SpiNNaker is a massively parallel, entirely digital Advanced RISC Machine (ARM)-based system. It has a unique interconnect communication system optimized for spike-based network communication, consisting of thousands of ARM cores. The processor unit itself is not customized for neuromorphic functions and is general-purpose. IBM's TrueNorth and Intel's Loihi are well-known examples of fully custom ASIC implementation of neuromorphic systems. The use of FPGA is frequently employed in implementations as a stop-gap measure before a specific chip implementation [8].

2.1.3.2 Analog Neuromorphic Systems

We divide analog systems into programmable and custom chip implementations, just to how digital systems are broken down. Field programmable analog arrays (FPAAs) are similar to FPGAs for digital systems. FPAAs have been used for analog neuromorphic implementations for many of the same reasons as FPGAs have. In addition, there have been specialized FPAAs created for neuromorphic systems, such as the NeuroFPAA and the Field Programmable Neural Array

(FPNA). These circuits are not broader FPAAs for general analog circuit design but feature programmable components for neurons, synapses, and other components [8].

2.1.3.3 Mixed Analog/Digital Neuromorphic Systems

The best way to get around some inherent restrictions of analog implementation is typically through mixed analog and digital implementation. In many analog neuromorphic systems, synaptic weights are typically stored in digital memory for reliability and longer duration. Digital communication is used either within the chip or between neuromorphic chips in some analog neuromorphic systems. Typically, these communications occur in the form of digital spikes. It is also common to use digital components for programming or learning mechanisms. The mixed analog and digital family of projects includes Neurogrid and BrainScaleS [8].

2.2. Lava software framework

In this section, we present an overview of the Lava software framework. First, we present a general definition and the key attributes. This is followed by a presentation of detailed information on the Lava software framework.

2.2.1 General Definition and Key Attributes

Lava is an open-source software framework used to build applications for neuromorphic hardware. It provides developers with the tools and concepts needed to build distributed, massively parallel applications. These can be used on a variety of system architectures, ranging from ordinary processors to event-driven neuromorphic chips. Lava provides libraries for deep learning and constrained optimization, which facilitates the creation of algorithms. In addition, it includes tools to adapt these algorithms to different types of hardware [3].

The key attributes of the Lava Software framework are listed in the following:

1. Asynchronous parallelism - Lava software framework supports asynchronous event-based processing for heterogeneous, distributed, neuromorphic, and massively parallel hardware platforms. It can be used for edge and cloud-based systems [3].

2. Refinement - The Lava framework facilitates an iterative approach to software development. This is accomplished by enabling the transformation of abstract processes into architecture-specific implementations. As a result, applications can be developed using high-level behavioural models, which can be progressively decomposed into lower-level models optimized for different platforms [3].

3. Cross-platform - Lava supports cross-platform execution of computational processes on neuromorphic Intel Loihi architectures, as well as conventional CPU/GPU architectures. It is flexible in the dual sense that the same computational processes can be executed on different platforms, and different processes can be executed and interacted across architectures through message passing [3].

4. Extensible – Since Lava is open-source, it can be extended to accommodate an expanding range of use cases over time. It can also connect with other frameworks like TensorFlow, ROS, and Brian [3].

5. Trainable - Lava comes with powerful training algorithms that can be used to train models offline as well as online (i.e., in real-time) in the future [3].

6. Accessible - Lava offers a Python API, making it easy to build and execute models on distributed parallel systems [3].

2.2.2 Detailed information on LAVA framework

The LAVA software stack is shown in Figure 4.

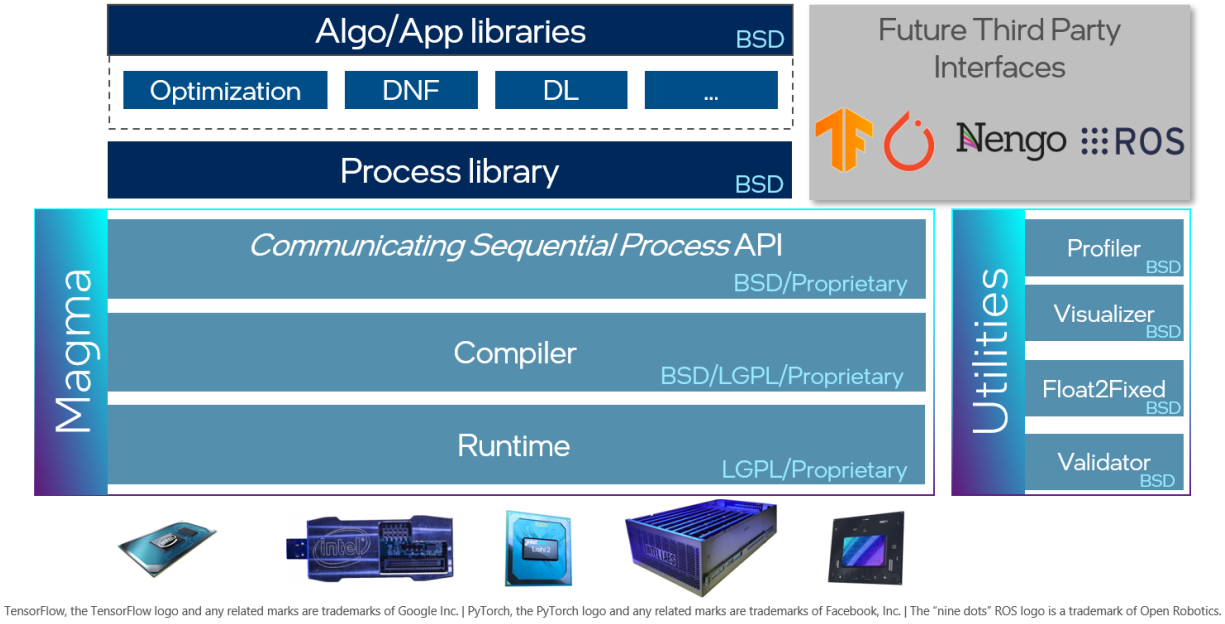

Figure 4 LAVA Software stack [3]

The fundamental components of the Lava software stack consist of the Communicating Sequential Process API, the compiler, and the runtime, as depicted in Figure 4. When combined, these components form the Magma layer of Lava. Lava offers the ability to compile and execute processes on various backends, including neuromorphic hardware, using the Magma layer's low-level interface. The Magma layer serves as the basis for the development of new processes and process models. The Lava process library offers generic, low-level, and reusable processes that can be used in creating higher-level algorithm and application libraries [3].

Lava offers several libraries for deep learning, optimization, and dynamic neural fields, with plans to release future libraries supporting symbolic vector architectures and evolutionary

optimization. In addition, Lava provides several utilities for application profiling, automatic float-to-fixed-point model conversion, and network visualization [3].

All of Lava's features and libraries are available through Python, and the software is open for extension to third-party frameworks such as TensorFlow, ROS, or Nengo. Lava also welcomes open-source contributions to its future libraries and utilities. The low-level interface also supports mapping neural networks onto neuromorphic hardware and asynchronous message passing. The software is available for free use under BSD-3 licensing at GitHub [3]. Lava has lower-level components that are responsible for mapping algorithms to various hardware backends. These components are licensed under the LGPL-2.1 license, which aims to prevent commercial proprietary forks. However, some specific components that support architectures, such as Intel Loihi, may not be publicly available. Intel shares them only as extensions with eligible users. Loihi is a neuromorphic chip designed by Intel Labs that uses an asynchronous spiking neural network (SNN) to implement adaptive self-modifying event-driven fine-grained parallel computations [1]. Loihi was succeeded by Loihi 2 in late 2021.

2.2.2.1 LAVA Constrained Optimization Library

The Lava framework offers a set of optimization solvers that leverages Intel Loihi neuromorphic hardware for constrained optimization [3]. Constrained optimization is a common problem in various fields where the goal is to minimize or maximize an objective function while subject to a set of constraints. Neuromorphic computing is a promising approach to address constrained optimization problems because it aligns well with the dynamics of spiking neural networks. In this approach, individual neurons represent different states of variables, and the neuronal connections can encode constraints between the variables. Recurrent inhibitory synapses connect neurons representing mutually exclusive variable states, while recurrent excitatory

synapses link neurons representing reinforcing states. By implementing this on neuromorphic hardware, a spiking neural network can evaluate conflicts and cost functions involving many variables, leading to a fast convergence towards an optimal state. In addition, the precise timing of individual spikes that occur within the SNNs allows them to readily escape from local minima [3].

At the time of writing this thesis, the Lava repository offered only quadratic programming (QP) and quadratic unconstrained binary optimization (QUBO) to solve optimization problems on both conventional central processing unit (CPU) and Loihi 2 backends. The development team plans to add support for other relevant constrained optimization problems. The order of development will build on the capabilities of existing solvers and includes Constraint Satisfaction Problems (CSP), Integer Linear Programming (ILP), Mixed-Integer Linear Programming (MILP), Mixed-Integer Quadratic Programming (MIQP), and Linear Programming (LP) [3].

The library's general architecture is illustrated in Figure 5.

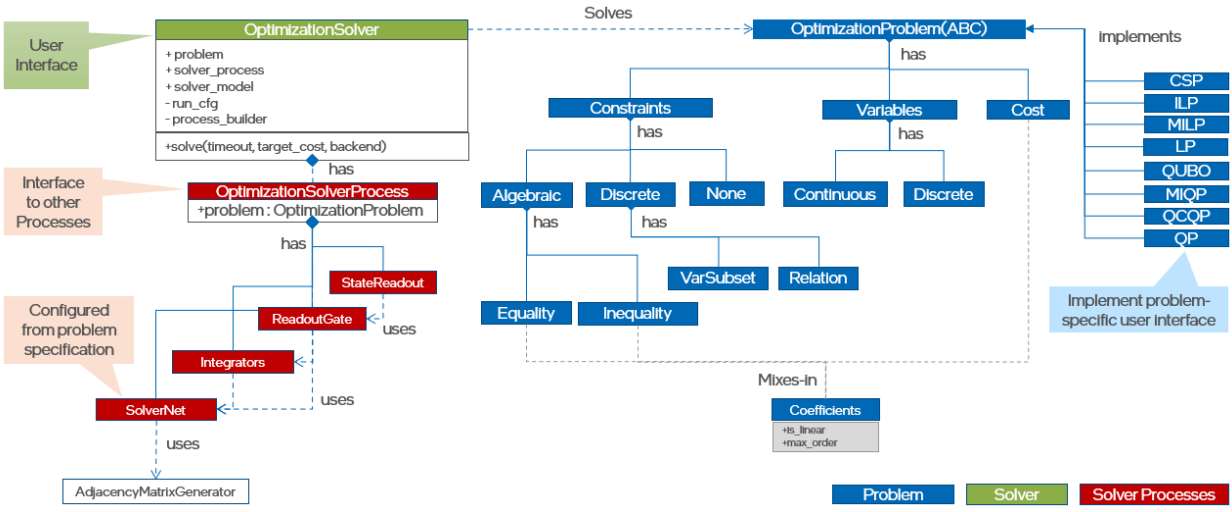

Figure 5 General architecture of LAVA Constrained Optimization Library [3]

The OptimizationProblem instances are created by combining the Constraints, Variables, and Cost classes, which describe the characteristics of each subproblem class, based on the general definition of constraint optimization problems. QP problems are defined by linear equality and inequality constraints with continuous domain variables and a quadratic function. While CSPs are defined by discrete constraints, consisting of variable subsets and a binary relation that describes the mutually permitted values for such discrete variables, and have a constant cost function that aims solely to fulfil the constraints [3].

For each problem class, an API can be built by inheriting from OptimizationSolver and combining specific flavours of Constraints, Variables, and Cost. To instantiate the generic OptimizationSolver class, an instance of an Optimization problem is required. By doing so, the interface of the OptimizationSolver remains constant, while the OptimizationProblem allows for more flexibility in creating new APIs. The OptimizationSolver understands the composite structure of the OptimizationProblem and composes the required solver components and Lava processes behind the scenes [3].

2.3 Other Neuromorphic Software Platforms:

This sub-section presents several neuromorphic software platforms. First, we present software platforms by grouping them into frameworks, simulators, and neuromorphic supporting software. Then, the main differences between these neuromorphic software platforms are summarized.

2.3.1 Neuromorphic Frameworks

Frameworks are general-purpose tools that provide a set of libraries and algorithms for building, training, and deploying neuromorphic systems. These frameworks include tools for

building and training neural networks, as well as for deploying these networks on real-world systems.

2.3.1.1 NxSDK

Intel's Loihi neuromorphic platform comes with its own set of programming abstractions, compilers, and firmware. The set of these software tools is referred to as NxSDK. The NxSDK supports a runtime for the lower layers and interfacing with various third-party frameworks at both the development and runtime levels [9]. The NxSDK offers various levels of abstraction for network specification. At the lowest level, users can manage each neuron's parameters. At the highest level, users can use pre-packaged modules to carry out specific tasks without managing individual neuron parameters. NxSDK acts as a backend for various external frameworks, allowing users to create networks using the programming language of their preference [2]. Furthermore, it provides interfaces for the Robot Operating System (ROS).

ROS (Robot Operating System) is a collection of software libraries and tools for creating robot applications. It is open-source software that aims to promote code reuse in robotics research and development. ROS is a system that consists of multiple processes that work together, allowing separate components to be created and connected at runtime. It offers typical operating system services, including hardware abstraction, low-level device control, standard functionality implementation, communication between processes, and package management. ROS also provides resources for acquiring, constructing, writing, and executing code on multiple computers [10].

2.3.2 Neuromorphic Simulators

Simulators are software tools used to simulate the behaviour of neuromorphic systems. These simulators allow users to model and test the performance of neural networks in a controlled environment.

2.3.2.1 BRIAN

BRIAN is a *"simulator for spiking neural networks in Python"* [11]. It is a user-friendly and incredibly adaptable tool for quickly creating new models, particularly networks of single-compartment neuron models. Users may define models by expressing arbitrary differential equations in regular mathematical notation in addition to using standard types of neuron models. Modeling and data analysis may both be done using scientific modules in Python. Despite the overheads of an interpreted language, effective simulations are possible because of vectorization techniques. BRIAN will be especially helpful when working with non-standard neuron models that are difficult to cover by current software and when looking for an alternative to MATLAB or C for simulations. BRIAN is also ideal for teaching computational neuroscience because of its simple and clear grammar [11].

2.3.2.2 Nengo

"Nengo is a neural simulator based on a theoretical framework called the Neural Engineering Framework (NEF). The NEF is a large-scale modelling approach that can leverage single neuron models to build neural networks with demonstrable cognitive abilities" [12]. Nengo is a python library for creating and modelling extensive neural networks. Nengo can produce complex spiking and non-spiking neural simulations with reasonable defaults in just a few lines of code. Nengo, however, is highly extensible and flexible. We can define our own neuron types and learning rules,

receive input directly from hardware, design and operate deep neural networks, control robots, and even mimic your model on a different neural simulator or neuromorphic hardware [13].

2.3.2.3 NEST

"The neural simulation tool NEST is designed for large networks of simple spiking model neurons " [14]. NEST offers high-level instructions to build spatially structured networks and includes a wide range of neuron and synapse types. Parallel simulation is supported by NEST, which has a Python-based user interface. The GNU Public License governs NEST, which is accessible through www.nest-initiative.org. NEST is best suited for neural networks whose subthreshold dynamics are well captured by a small number of differential equations. NEST simulations run on a set time grid by default. NEST, however, also offers precisely timed spikes, combining the accuracy of event-driven simulators with the effectiveness of grid-based modelling [14]. PyNEST, which combines NEST simulations with Python programming and data analysis, is the most popular user interface for NEST [14].

2.3.3 Neuromorphic Supporting Softwares

In the field of neuromorphic computing, TensorFlow and PyTorch are two popular deep-learning libraries used. They offer a variety of tools and interfaces for constructing and training neural networks and can be utilized for various tasks such as image recognition, natural language processing, and predictive analysis. These libraries simplify the process for developers to utilize neuromorphic hardware, allowing them to create and implement deep learning models on these platforms quickly. Furthermore, PyNN is a python package that provides a common programming interface to multiple neuromorphic simulators.

2.3.3.1 PyTorch

PyTorch is an open-source machine learning framework created by the Meta AI research team. It is designed for constructing and educating neural networks, emphasizing versatility and quickness. PyTorch provides a user-friendly interface for constructing deep learning models and a low-level interface for manipulating tensors and executing tensor operations. This framework is widely utilized in computer vision, natural language processing, and various AI and machine learning fields. PyTorch is recognized for its simplicity, dynamic computation graphs, and capability for training across multiple devices [15].

2.3.3.2 TensorFlow

TensorFlow is an open-source platform created by Google's research team for executing machine learning, deep learning, and other statistical and predictive analytics tasks. It offers both a user-friendly high-level application programming interface (API) for creating and training neural networks and a low-level API for writing efficient custom operations. TensorFlow comes equipped with visualization and debugging tools for the training process, as well as a vast library of pre-trained models and tutorials [16].

The SNN Conversion Toolbox [17] helps convert Artificial Neural Networks from popular frameworks like TensorFlow and PyTorch. This tool takes in the networks and changes them into a format compatible with the Loihi hardware. Finally, the tool maps the converted networks onto Loihi using NxTF [2]. NxTF is an API and compiler for developing deep spiking neural networks on Intel Loihi.

2.3.3.3 PyNN

PyNN is a simulator-independent python package for building neuronal network models [18]. PyNN provides a common programming interface to multiple simulators to reduce or eliminate the problems of simulator diversity while retaining the benefits [18]. PyNN improves neural network modelling efficiency by offering high-level abstraction, encouraging code sharing and reuse, and serving as a platform for simulator-independent analysis, visualization, and data-management tools. PyNN makes it simpler to compare results across different simulators, improving the reliability of modelling research. The website http://neuralensemble.org/PyNN hosts PyNN, which is open-source software.

2.3.4 Differences Between Neuromorphic Software Platforms

Table 2 presents the differences between neuromorphic software platforms. In this table, a '✓' means that the feature is supported by the particular neuromorphic software platform, whereas a '×' means that the feature is not supported by the neuromorphic software platforms.

Asynchronous message passing, where the exchange of information takes place in an asynchronous manner, is supported only by ROS and Lava software frameworks. Hardware acceleration is not supported by BRIAN and ROS platforms. While others listed in the table support hardware acceleration. Direct backpropagation, an algorithm for supervised learning of neural networks, is supported only in TensorFlow, PyTorch, and Lava software platforms. Behavioural abstraction is present only in the Nengo and Lava software platforms. Machine learning can be incorporated into neuromorphic computing through spiking neural networks. Thus, all the software platforms mentioned in the table except PyTorch and TensorFlow are optimized to support SNNs.

Table 2. Difference Between Neuromorphic Software Platforms [19]

	TensorFlow	PyTorch	Nengo	PyNN	NxSDK	BRIAN	ROS	Lava
Asynchronous message passing	☒	☒	☒	☒	☒	☒	☑	☑
HW acceleration	☑	☑	☑	☑	☑	☒	☒	☑
Direct Backprop	☑	☑	☒	☒	☒	☒	☒	☑
Behavioural abstraction	☒	☒	☑	☒	☒	☒	☒	☑
Spiking neuron modelling	☒	☒	☑	☑	☑	☑	☑	☑

2.4 Chapter Summary

In this chapter, we have discussed the background concepts that are crucial to this thesis. First, the general definition and history of neuromorphic computing are presented. It was followed by a brief overview of architecture and types of neuromorphic systems. Finally, we discussed on LAVA software framework and examined different neuromorphic software platforms.

Chapter 3

3. Related Work

In this chapter, we first review the existing works on neuromorphic computing software. Then, we review the relevant literature on resource allocation problems in OFDMA systems. To the best of our knowledge, our work is the only one that studies the use of the Lava neuromorphic framework to solve resource allocation problem in wireless networks. In the end, we summarize the chapter.

3.1. Works on Neuromorphic Computing Software

This section presents the works on neuromorphic computing software. First, we present the works on neuromorphic software platforms for different neuromorphic hardware. This is followed by the presentation of Works on Neuromorphic computing simulators.

However, all these enumerated works did not consider radio resource allocation problems in wireless networks. In addition, these works do not perform evaluations on the Lava framework.

3.1.1 Works on Neuromorphic software platforms for different neuromorphic hardware.

This sub-section presents works on neuromorphic software platforms for various neuromorphic hardware.

Several case studies on neuromorphic hardware platforms and their respective architecture and software ecosystems are presented in [8]. The purpose of this survey was to provide an overview of the current state of neuromorphic computing systems. The survey began by

introducing the neuron and synapse models that were central to neuromorphic computing, and then went on to discuss how these models impacted the design of hardware platforms. To provide a more detailed understanding of these systems, the survey also included case studies of several representative hardware platforms like SpiNNaker, TrueNorth, Loihi, BrainScaleS, and Tianjic. For each platform, the survey described its architecture, as well as the software ecosystem that supported it. By examining these platforms in detail, the survey aimed to provide readers with a deeper understanding of the current state of the art on neuromorphic computing. Although interesting, this survey did not cover the Lava framework for Loihi.

The performance of the TrueNorth neuromorphic chip in solving different types of graph problems has been evaluated in [20]. This work presented the implementation of QUBO problems by scripting in MATLAB, utilizing the corelet programming integrated development environment for IBM's Neurosynaptic system. The implementation of the model was supported by two platforms - the first being the Neurosynaptic Simulator for Corelet System (NSCS) simulation platform, while the other platform was IBM TrueNorth Neurosynaptic hardware. The authors of this study carried out practical applications of the Quadratic Unconstrained Binary Optimization (QUBO) problem as a potential solution for various graph-related problems. They also explored how QUBO can be implemented on IBM TrueNorth Neurosynaptic hardware. The proposed system was implemented using recurrent neural networks (RNNs). Furthermore, this paper discussed various potential problems that can be addressed using the graph problem approach based on QUBO. These findings provide valuable insights into the practical applications of QUBO in solving complex problems in different fields.

3.1.2 Works on Neuromorphic computing simulators.

This sub-section presents works on various neuromorphic computing simulators.

The work in [21] aimed to assess the performance of several publicly available SNN simulators, such as NEST, Brian2, Brian2GeNN, BindsNET, and Nengo, in terms of speed, flexibility, and scalability with respect to non-computational neuroscience workloads. In addition, their performance was evaluated using a common front-end neuromorphic TENNLab framework. The software architecture of the TENNLab framework is described in [22]. The authors also provided a set of recommendations on the suitability of employing these simulators for different tasks and scales of operations, as well as the characteristics for a future generic ideal SNN simulator for different neuromorphic computing workloads. Developing and testing computer models of brain function and analyzing and interpreting large-scale neuroimaging data sets are several examples of Computational neuroscience workloads. Although there were several openly available SNN simulation packages developed by various research groups, these simulators mostly focus on computational neuroscience simulations, and a few targets small-scale machine learning tasks with SNNs. Comparisons of these simulators have also mostly targeted computational neuroscience-style workloads. Therefore, the authors in this work aimed to address the lack of flexible and easy-to-use SNN simulators by evaluating their performance for non-computational neuroscience workloads and providing recommendations for their use.

Reference [23] presented the principles of the neural engineering framework (NEF) and a number of its extensions which provide additional methods and techniques for analyzing and leveraging spiking dynamical computations. In addition, it presented a review of several recent applications that use Nengo to build models for neuromorphic hardware. In this work, the authors introduced the NEF and showcased some illustrations of how it can be used to create new recurrent neural networks (RNNs) and map them onto various neuromorphic hardware. Although this work covered only a few examples of NEF applications, they endeavored to explain its fundamental

concepts both formally and through practical illustrations. By comparing it to other techniques, this work shows that the NEF provides certain efficiency and accuracy benefits, which are essential for low-power neuromorphic applications. When combined with the Nengo software, the NEF offers both theoretical and practical tools for designing a diverse array of spiking networks across a broad range of hardware platforms. Finally, the authors expect that the NEF will be utilized in innovative and unforeseen manners as it continues to evolve.

Reference [1] provided a review of a spiking neural network and a neuromorphic architecture. Moreover, it covered communication frameworks and supporting software that is designed for various neuromorphic systems. Furthermore, the article discussed multiple factors that should be considered when selecting algorithms such as Backpropagation, Evolutionary algorithm, Hebbian learning, and spike-timing-dependent plasticity (STDP) for neuromorphic implementation. This reference paper's research offered a comprehensive analysis of the research and justifications that have supported neuromorphic computing throughout its history. The authors' review started with a 35-year examination of the reasons and stimuli behind neuromorphic computing. This was followed by an examination of the main research domains in Neuromorphic computing. These domains consisted of neuro-inspired models, algorithms, learning techniques, hardware, and devices, supporting systems, and, ultimately, applications. The objectives of this reference paper study are to provide a thorough summary of the research performed in neuromorphic computing from its origins and to inspire further research by highlighting the gaps in the field that necessitate new investigations.

3.2 Works on OFDMA Resource Allocation

In this section, we present background information on OFDMA systems and related works on radio resource allocation in OFDMA systems.

3.2.1 Background Information on OFDMA System

In this sub-section, we present an overview of the OFDMA system. First, the fundamentals of OFDMA are explained. Then, its advantages and disadvantages are detailed. Finally, the main applications of OFDMA are provided.

3.2.1.1 Fundamentals

Orthogonal frequency division multiple access (OFDMA) is an extension of OFDM digital modulation technology into a multiuser environment. The main idea of OFDM is to divide the high-rate data stream into many low-rate streams transmitted simultaneously over their subcarrier. The subcarriers are spaced from one another by an amount that makes them orthogonal to one another. To provide a high data-rate communication system, OFDM combines the advantages of several modulation techniques, including Quadrature Amplitude Modulation (QAM), Time Division Multiplexing (TDM), and Frequency Division Multiplexing (FDM). In addition, OFDM is one of the 5G fundamentals [21]. OFDMA inherits from OFDM the capacity to correct channel distortions in the frequency domain without the requirement for computationally demanding time-domain equalizers. In addition, the system is given significant resource management flexibility by using a dynamic subcarrier assignment approach and the inherent protection against multiple access interference (MAI) provided by subcarrier orthogonality [24]. OFDMA is essential for real-time applications since it enables several Wi-Fi stations to transmit or receive data simultaneously. It also allows allocating resource units for several Wi-Fi stations to transmit without collisions and defines uplink OFDMA-based random access. Thus, OFDMA is a promising solution for low-latency data transmission from numerous stations to an access point [25].

In the following, the fundamentals of OFDMA are described:

Orthogonality: Subcarriers in OFDMA are orthogonal to one another, preventing interference between the subcarriers. This makes it possible for multiple users to simultaneously share the same frequency band without causing interference.

Frequency Division: OFDMA divides the overall frequency band into multiple subcarriers. Each subcarrier can be assigned to a separate user. As a result, the frequency spectrum can be used more effectively, and more users can efficiently share a single channel.

Communication through the OFDMA system is illustrated in the simplified block diagram depicted in Figure 6.

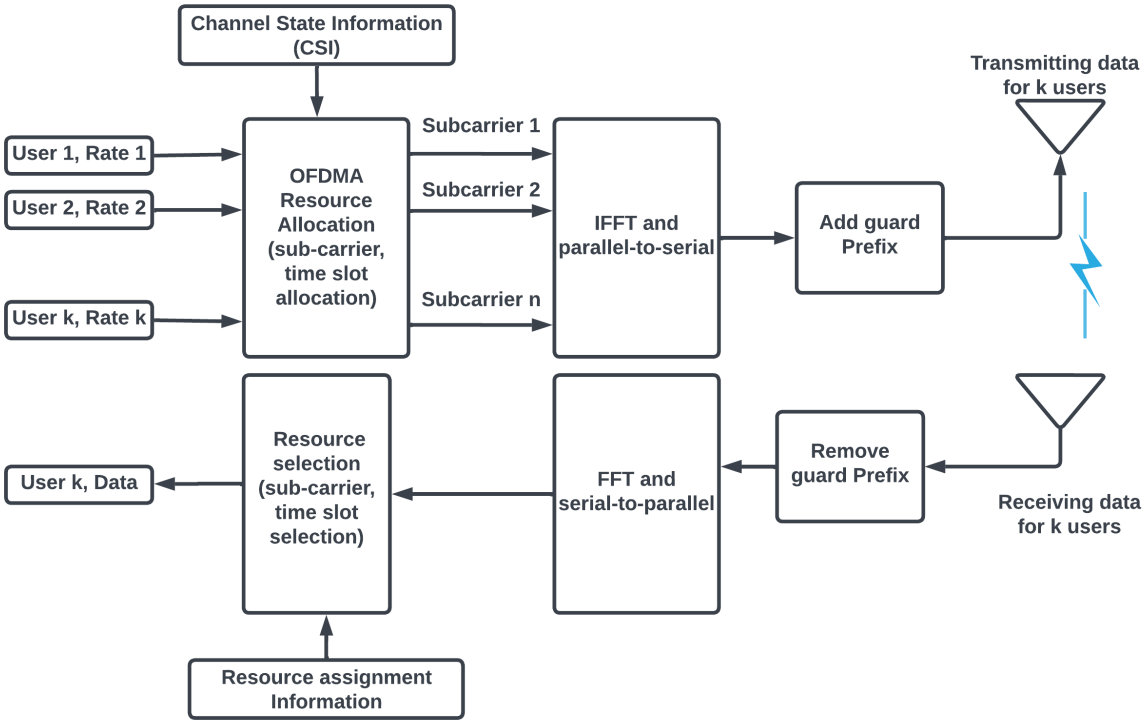

Figure 6 Diagram of OFDMA system [26].

In the OFDMA system, the overall bandwidth is divided into N sub-carriers, and information can be sent over these subcarriers from and to multiple users simultaneously. In Figure 6, k and n

denote the number of users and sub-carriers, respectively. In the downlink OFDMA system, the base station simultaneously transmits data to k users. First, the channel state information (CSI) from all k users is collected, then the resource allocation algorithm is applied based on this information. According to this algorithm, the k users' data is divided, modulated, and fed into the N orthogonal sub-carriers simultaneously. At the receiver side, all data from N subcarriers are demodulated. The allocation information is sent to the corresponding users so that they can extract their data via a designated control channel. The uplink OFDMA is similar to the downlink OFDMA but in the reverse direction, i.e., from multiple users to the base station.

3.2.1.2 Advantages

The key advantages of OFDMA are explained briefly in this sub-section.

1. Scalability

One of the most significant benefits of OFDMA is its scalability. The number of subcarriers used, and the size of each subcarrier can be adjusted to accommodate different bandwidth requirements. Therefore, OFDMA can handle a wide variety of bandwidth requirements by using its flexible subcarrier structure to allocate bandwidth in an efficient and adaptable manner. Fast Fourier Transform (FFT) transforms data from the time to the frequency domain. Scalability is made possible by adopting the FFT size to the channel bandwidth while maintaining a fixed sub-carrier frequency spacing. It is possible to determine the physical (time and frequency) resource unit by adjusting the subcarrier spacing and symbol duration. Scalability has several direct benefits, such as deployment flexibility. OFDMA systems can be implemented in various frequency band intervals with no change to the air interface to adapt to diverse spectrum allocation and usage model requirements. However, other multiple access schemes like Time Division Multiple Access

(TDMA) do not naturally offer such flexibility [4]. TDMA divides the available bandwidth into multiple time slots.

2. Robustness to Multipath

Subchannels in OFDMA systems maintain the orthogonality in multi-path channels. Multi-path interference occurs when the signal is reflected and arrives at the receiver at different times. The cyclic prefix window mitigates this interference by adding a portion of each symbol ending at the beginning of the same symbol. Suppose the length of the cyclic prefix window is long enough to contain all of the multi-path components of the signal. In that case, the signal can be accurately reconstructed at the receiver without interference. Therefore, the system's performance is not limited by the number of multi-path components as long as they are contained within the cyclic prefix window. As a result, OFDMA systems are resistant to multi-path effects. The subchannel orthogonality within the cyclic prefix window further eliminates the requirement for time synchronization [4].

3. Downlink Multiplexing

Since OFDMA is orthogonal in nature, downlink transmission may utilize the maximum available power without the need for power regulation. In addition, OFDMA has one more dimension of flexibility compared to TDMA to allocate power and subchannels to different users in the same time slot [4].

4. Uplink Multiple Access

Orthogonal sub-channels are used for OFDMA uplink access. An OFDMA system can provide higher reverse-link capacity than a traditional Code-Division Multiple Access (CDMA) systems

by removing intra-cell interference. In frequency selective channels, OFDMA uplink can also benefit from frequency selectivity in the same way as downlink does by providing the best sub-channels to the appropriate access users to enhance the entire system's performance further [4].

3.2.1.3 Disadvantages

The disadvantages of OFDMA are:

1. The whole bandwidth of OFDMA is split into several narrowband subcarriers. OFDMA is more susceptible to frequency offset and phase noise since the subcarrier spacing is often relatively small. In order to reduce inter-carrier interference, frequency synchronization is crucial. At high mobile speeds, the Doppler effect makes the issue much more important [4].

2. The main drawback of OFDMA is the high peak-to-average power ratio (PAPR). PAPR is defined as the ratio of peak power to the average power of a signal. Indeed, the high PAPR causes interference and degrades the system's performance while the OFDM signal passes through the amplifier. Several methods may be used to reduce the PAPR. Finally, a successful OFDMA system design should carefully consider all these challenges [4].

3.2.1.4 Applications:

OFDMA is widely used in wireless communication systems for efficient radio resource allocation.

Because of its effectiveness and robustness to multipath propagations, OFDMA has been extensively utilized for wideband communication protocols. Some of the key applications of OFDMA systems include:

1. Cellular Networks: OFDMA is implemented in many technologies and standards, such as

3GPP, LTE, 5G and beyond [3]– [5]. Cellular networks use OFDMA to improve system capacity and increase user data rates. By allowing several users to share a single channel, OFDMA efficiently utilizes the available frequency spectrum.

2. Wireless Metropolitan Area Networks (WMANs): OFDMA has become part of the IEEE 802.16 standards for WMANs [24].

3. Wireless Local Area Networks (WLANs): OFDMA is one of the fundamentals of IEEE 802.11ax [25].

4. Satellite Communication: OFDMA is used in mobile satellite communications [27], a satellite/terrestrial integrated mobile communication system [28] to increase the amount of data transmitted over a satellite communication channel and enable efficient use of the available frequency spectrum.

5. Cognitive Radio Networks: OFDMA is a potential access technique for IEEE 802.22 Wireless Regional Area Networks (WRAN), a cognitive radio technology that makes use of white spaces in the TV frequency spectrum [29].

3.2.2 Related works

In this section, we present the related works of resource allocation in OFDMA systems. First, we present works on RA in OFDMA for 5G mobile networks. This is followed by the presentation of works on RA in OFDMA for beyond 5G mobile networks. Then, we present works on RA in OFDMA for WiMAX systems. Finally, we present other general works on RA in OFDMA.

Although interesting, the presented solutions designed to solve the resource allocation in OFDMA systems were based on standard CPU. While in this work, we focus on a novel

neuromorphic approach. OFDMA systems are used in the implementation of multiple other technologies and standards. Therefore, many works have been interested in OFDMA.

3.2.2.1 Works on RA in OFDMA for 5G mobile networks.

This sub-section presents works on RA in OFDMA for 5G mobile networks.

In [30], the authors investigated resource allocation in OFDMA-based 5G networks with slicing considerations. Specifically, enhanced Mobile Broadband (eMBB) and ultra-reliable low latency communications (URLLC) services were considered. RAN resources were then allocated to these different slices according to their requirements. In addition, the authors presented an adaptive modulation and coding (AMC) resource allocation method that allowed simultaneous transmission of eMBB and URLLC users on the same RAN infrastructure. The resource allocation problem was formulated as a sum-rate maximization problem while satisfying the orthogonality constraint (i.e., service isolation), the URLLC users' latency constraint, and the minimum rate constraint of eMBB users. However, solving the problem mathematically was intractable due to the complexity introduced by the AMC scheme and binary constraints. To overcome this issue, the authors used two approximation functions and a penalized formulation to transform the initial stepwise optimization problem into a continuous linear problem. In addition, a heuristic scheduling algorithm was proposed to address complexity, time consumption, and feasibility issues. The performance of the proposed schemes was evaluated and compared using extensive simulations to illustrate their ability to allocate resources dynamically for different services while satisfying their reliability, latency, and minimum rate requirements.

In [31], a proactive downlink transmission framework (PDF) based on wireless network unit association and frequency division multiplexing (FDM) was proposed to manage both user

association and radio resource allocation. The developed framework decomposed the proactive task data transmission problem into three sub-problems. The system model consists of anchor nodes and several access points (APs). The anchor node forwards the task data to multiple APs and manages radio resource allocation among and under the APs. Three low-complexity optimization algorithms were designed to optimize the size of wireless network units, the FDM allocation among APs, and the radio resource allocation for individual AP transmission. The second-phase frequency division algorithm was geolocation-based to ensure the radio resource allocation fits the forwarding task distribution. In the third phase, a reinforcement learning-based algorithm is proposed to manage the link Radio Resource Units (RRUs) allocation under a non-cooperative mode of a single AP. The framework's performance is evaluated using simulations to demonstrate its ability to allocate resources while dynamically satisfying reliability, latency, and minimum rate requirements.

This work focused on OFDMA adaptation for physical layer transmission in wireless communication. The wireless spectrum was divided into Radio Resource Units (RRUs) that consist of a fixed number of subcarriers and symbols per time slot. These RRUs are then combined to form Radio Blocks (RBs), which are used as the primary scheduling unit for transmitting data packets. The growth of the Intelligent Industrial Internet of Things is attributed to advanced sensing, data analysis, and communication methods, which have also enhanced 5G networks. For the URLLC scenario, a low-complexity algorithm is needed to ensure reliable transmission and prompt radio resource allocation. Therefore, the objective of the designed scheme was to achieve ultra-low latency and stable downlink transmission.

Reference [32] studied the OFDMA-based downlink resource allocation problem in 5G new radio (NR) mobile networks. The main objective of the considered resource allocation

problem aimed to optimize the average throughput of the mobile network. One of the distinctive features of 5G mobile networks was the utilization of antenna arrays, which enabled beams to be formed in a particular direction, allowing for data transmission to multiple users simultaneously during the same transmission time interval (TTI) via the same carrier or resource block (RB). Typically, antenna arrays include up to 64 single antennas. This work modelled the data transmission process from the cell to the user equipment (UE). In addition, the authors formulated a mathematical abstraction of the resource allocation problem and proposed a lightweight algorithm that uses a proportional fair metric. The proposed algorithm utilized calculation procedures that were not resource-intensive, and these procedures are based on the conjugation matrix of the related graph. When this algorithm was compared to the modified round-robin algorithm, it was found that they have similar performance in terms of average throughput. However, the proposed algorithm was more resource-efficient and consumed fewer resources at each step. Therefore, the authors suggested that adjusting the algorithm parameters can further improve its performance.

Reference [33] presented subcarrier allocation in dual-band OFDMA as a potential choice for future passive optical network (PON) systems because OFDMA can provide high spectral efficiency that can meet the requirements of 5G mobile fronthaul and backhaul. The backhaul refers to the connection between a mobile network and a wired network, whereas the fronthaul refers to the network architecture that connects the remote cell sites to the Baseband Unit. In this work, the authors proposed a two-step algorithm for dynamic subcarrier allocation in dual-band OFDMA-PON. The first step involved an exhaustive search in determining the initial subcarrier allocation and the frequency band for each remote terminal. In the second step, the algorithm monitored the traffic of the associated services during a monitoring window and adjusted the

subcarrier allocations if necessary. The simulation results of this work demonstrated that the proposed algorithm achieved high bandwidth utilization while meeting the QoS requirements of fronthaul and backhaul services in 5G mobile networks.

3.2.2.2 Works on RA in OFDMA for beyond 5G mobile networks.

This sub-section presents works on RA in OFDMA for beyond 5G mobile networks.

The authors of [34] presented a reliable OFDMA-based media access control (MAC) protocol for terahertz wireless communication in vehicular 6G networks. Under a fading environment, severe transmission errors were unavoidable when two mobile nodes or stations around a cell boundary were far away or in a bad channel. In such cases, the authors claimed that the proposed protocol could be used to ensure the reliability of the data transmission between the source and destination nodes. Moreover, [35] considered spectrum assignment and access schemes in non-contiguous OFDMA for Beyond 5G and 6G mobile networks.

3.2.2.3 Works on RA in OFDMA for WiMAX systems.

This sub-section presents works on RA in OFDMA for WiMAX systems.

In [36], fair radio resource allocation in downlink OFDMA was investigated. This work focused on fair resource allocation in downlink OFDMA-based WiMAX systems. Specifically, a sub-carrier allocation algorithm was proposed to maximize the sum of logarithmic user data rates. The data rate was arranged among the users in order to schedule the best channel for achieving fairness in resource allocation. Simulation results indicated that the proposed algorithm was 20% more efficient than the existing algorithms.

3.2.2.4 Other general works on RA in OFDMA.

This sub-section presents other general works on RA in OFDMA.

Reference [37] dealt with the radio resource allocation in New Generation Radio Mobile Networks (NGN) utilizing OFDMA. In this work, the main objective of the designed resource allocation algorithm was to optimize the total cell capacity and enhance the satisfaction level of the network users. The simulation results of this work demonstrated that the proposed algorithm outperforms other algorithms while maintaining the same level of complexity.

The work in [38] examined a single-cell multi-user network that utilizes OFDMA and includes an unmanned aerial vehicle (UAV) acting as an amplify-and-forward relay to enhance the quality-of-service (QoS) for user equipment (UE) at the cell edge. This work aimed to increase throughput while maintaining user fairness by jointly optimizing the communication mode, the subchannel allocation, the power allocation, and the UAV trajectory. The problem has been demonstrated to be NP-hard. Therefore, to optimize both the UAV trajectory and the resource allocation efficiently, the problem was decomposed into three subproblems which were (i) mode selection and subchannel allocation, (ii) trajectory optimization, and (iii) power allocation. These subproblems were then solved iteratively. The simulation results of the proposed algorithm demonstrated better performance than the random algorithm and cellular scheme.

Reference [39] studied the resource allocation for optimizing the energy efficiency (EE) in OFDMA-Enabled wireless powered communication network (WPCN). The energy-limited device used harvested energy to transmit independent signals to multiple IoT users via OFDMA. In order to attain the objective of maximizing the EE of the system, a joint optimization of the duration of energy harvesting, subcarrier allocation, and power allocation was carried out. The problem

formulation falls under the challenging category of MINLP, which was arduous to solve directly. Therefore, an iterative algorithm was developed using the Dinkelbach method. The numerical results showed that the proposed algorithm achieved fast convergence and outperformed baseline algorithms. These results also emphasized the significance of the power source transmitting its maximum allowable power to achieve optimal EE performance.

3.3 Chapter Summary

In this chapter, we presented the state of art works on neuromorphic computing software. This was followed by the presentation of background information on OFDMA systems and state of art works related to resource allocation in OFDMA systems. Finally, we concluded that none of the existing works in the state-of-the-art considers radio resource allocation problems in wireless networks and performs evaluations on the Lava framework.

Chapter 4

4. System Model, Formulations and Evaluation

In this chapter, we present the system model, problem formulations, and evaluation. In the end, we summarize the chapter.

In this work, we consider using the Lava framework to solve a resource allocation problem in an OFDMA system. First, the problem is formulated as an ILP to be solved optimally using off-the-shelf functions and solvers such as CPLEX and MATLAB. Then, since the Lava QP solver does not support binary variables, the ILP is converted into a QUBO form to be solved using Lava neuromorphic constraint optimization library. To evaluate the performance of the proposed Lava-based approach, we compare our results to the ones obtained with classical solvers. Finally, we highlight the advantages of using the Lava framework to solve practical QUBO optimization problems, as well as the challenges that still remain to be solved.

4.1 System Model

In this section, we consider a downlink OFDMA system consisting of one base station (BS) that serves multiple users, as shown in Figure 7.

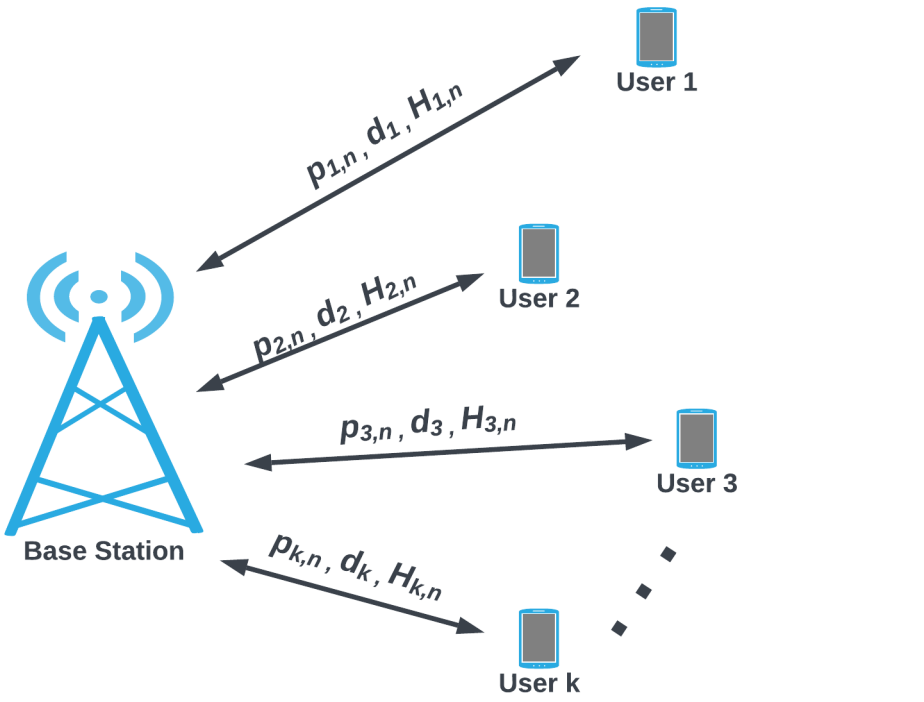

Figure 7 System Model

The number of users is denoted by k, while the number of sub-carriers is denoted by n. For every user, the base station (i) determines over which sub-carrier's data will be transmitted and (ii) decides how much data will be transmitted to that user over the selected sub-carrier. Let $R_{k,n}$ be the maximum achievable data rate if data is sent from the BS to user k on sub-carrier n and C_k be the amount of data that the BS must transmit to user k. The expression of $R_{k,n}$ is given as follows:

$$R_{k,n} = \log_2\left(1 + \frac{p_{k,n}\left|H_{k,n}\right|^2}{d_k^{\beta} N_{k,n} \Gamma}\right) \quad (1)$$

Where d_k is the distance between user k and the BS, $H_{k,n}$ is the channel gain between user k and the BS, $N_{k,n}$ is the noise variance, β is the pathloss exponent, and $P_{k,n}$ is the transmit power of

the BS used to transmit data to user k on sub-carrier n while Γ is a pathloss factor relative to a reference distance and it is set to 1 without loss of generality [40] [41].

The summary of notations used in the system model and for the problem formulation of the OFDMA downlink system model are defined in Table 3.

Table 3. Summary of Notations

Symbol	**Definition**
$\boldsymbol{k}$	Total number of users present in the system.
n	Total number of available sub-carriers in the system.
$\boldsymbol{R_{k,n}}$	The maximum achievable data rate if data is sent from the BS to user k on sub-carrier n.
$\boldsymbol{x_{k,n}}$	It is a binary variable that takes the value 1 if the sub-carrier n is assigned to user k. Otherwise, its value is 0.
$\boldsymbol{r_{k,n}}$	It is a continuous variable that returns the data amount sent to user k through sub-carrier n.
$\boldsymbol{C_k}$	The amount of data that the BS must transmit to user k.
$\boldsymbol{d_k}$	The distance between user k and the BS.
$\boldsymbol{H_{k,n}}$	The channel gain between user k and the BS.
$\boldsymbol{\beta}$	The pathloss exponent.
$\boldsymbol{P_{k,n}}$	The transmit power of the BS used to transmit data to user k on sub-carrier n.

Γ	Pathloss factor relative to a reference distance, and it is set to 1 without loss of generality [40] [41].
$N_{k,n}$	The noise variance.

4.2. Problem Formulation

In this section, we present the radio resource allocation problem formulation for a downlink OFDMA system. The objective of the considered joint sub-carrier selection and rate allocation is to maximize the sum-rate over all subcarriers and for all users under channel fading conditions. First, in section 4.2.1, the problem is formulated as an ILP. Then, in section 4.2.2, we transform it into a QUBO problem to be implemented and solved by the LQS.

4.2.1. ILP Formulation

In this sub-section, the considered radio resource allocation problem is formulated as an ILP where the objective is to maximize the sum-rate in a downlink OFDMA system. To formulate the problem, we need to define the following variables:

- $x_{k,n}$ is a binary variable that takes the value 1 if the sub-carrier n is assigned to user k. Otherwise, $x_{k,n} = 0$.
- $r_{k,n}$ is a continuous variable that returns the data amount sent to user k through sub-carrier n. For binary expansion purposes, as part of the QUBO formulation and following [42], $r_{k,n}$ is assumed to take integer values in the rest of the paper.

The ILP formulation of the problem is depicted in (P1).

$$\text{Maximize} \sum_{k=1}^{K} \sum_{n=1}^{N} r_{k,n} \quad \text{(P1a)}$$

$$\text{Subject to} \sum_{k=1}^{K} x_{k,n} \leq 1, \forall n \quad \text{(P1b)}$$

$$r_{k,n} \leq R_{k,n} \cdot x_{k,n}, \ \forall k, \text{n} \quad \text{(P1c)}$$

$$\sum_{n=1}^{N} r_{k,n} \leq C_k, \forall k \quad \text{(P1d)}$$

$$r_{k,n} \geq 0, x_{k,n} \in \{0,1\}, \forall k, \text{n} \quad \text{(P1e)}$$

In the following, we explain the objective function and the constraints that are considered in our problem.

The objective function in (P1a) aims to maximize the total system throughput. Indeed, the sum-rate maximization is achieved by maximizing the individual user allocation rate in such a way that the total system throughput is increased.

- Constraint (P1b) means that each sub-carrier must be assigned to at most one user. In other words, two or more users cannot be served simultaneously on a single subcarrier. However, it should be noted that a user can be served by one or more sub-carriers.
- Constraint(P1c) means that the data rate assigned to user k on sub-carrier n, denoted $r_{k,n}$, cannot exceed the maximum achievable data rate $R_{k,n}$.
- Constraint (P1d) means that the total data rate transmitted on all sub-carriers should not exceed the data amount the base station should transmit to user k.
- Constraint (P1e) defines the domain of the optimization variables.

4.2.2. QUBO Formulation

In this sub-section, the problem formulated as an ILP in (P1) is converted into a QUBO formulation in order to be mapped in the neuromorphic domain using the Lava framework. The following steps are used to convert the resource allocation problem into an appropriate QUBO form:

1) Perform the binary expansion of all integer variables. That is, all non-binary variables are written as a linear combination of binary variables so that the objective function and the constraints contain only binary variables. Since the LQS does not support real variables, the rate variables $r_{k,n}$ are assumed to take integer values in the rest of this paper.
2) Transform all present constraints which are present in the form of linear inequalities into general form of matrix equation $\mathbf{A}x = \mathbf{b}$, by inserting slack variables in the appropriate places. Where **A** is a matrix containing the coefficients of the binary variables in the column vector **x**, and **b** is a column vector containing the constants of the system of linear equations. Since the binary expansion has already been performed, the system $\mathbf{A}x = \mathbf{b}$ can be converted to a suitable quadratic penalty function defined as $(Ax-b)^2$. After adding slack variables, (P1) is given as follows:

$$\text{Maximize} \sum_{k=1}^{K} \sum_{n=1}^{N} r_{k,n} \tag{P2a}$$

$$\text{subject to} \sum_{k=1}^{K} x_{k,n} + a_n = 1, \forall n \tag{P2b}$$

$$r_{k,n} + b_{k,n} = R_{k,n} \cdot x_{k,n}, \forall k, \text{n} \tag{P2c}$$

$$\sum_{n=1}^{N} r_{k,n} + d_k = C_k, \forall k \quad \text{(P2d)}$$

$$a_n \in \{0,1\}, \forall n, d_k \in N, \forall k, \quad \text{(P2e)}$$

$$b_{k,n} \in N, r_{k,n} \in N, x_{k,n} \in \{0,1\}, \forall k, \text{n} \quad \text{(P2f)}$$

Since $x_{k,n}$ variables are already binary, and their summation over k is at most equal to 1, by definition, $a_n$are binary variables. Therefore, only slack variables $b_{k,n}$, d_k and rate variables $r_{k,n}$ need to be expanded into binary variables. Hence, the total number of positive integer slack variables added to the problem is equal to K + N + K × N. The binary expansion of the aforementioned variables is given as follows:

$$b_{k,n} = \sum_{j=0}^{\lfloor \log(R_{k,n}-1) \rfloor} 2^j s_{knj}, \quad (2)$$

$$d_k = \sum_{j=0}^{\lfloor \log(C_k) \rfloor} 2^j w_{kj}, \quad (3)$$

$$r_{k,n} = \sum_{j=0}^{\lfloor \log(R_{k,n}) \rfloor} 2^j q_{knj}, \quad (4)$$

where $s_{knj} \in \{0,1\}$, $w_{kj} \in \{0,1\}$ and $q_{knj} \in \{0,1\}$

The total number of binary variables resulted from converting the integer variables into binary variables are equal to

$\text{K} \times \text{N} + \text{N} + \sum_{k=1}^{K} (\lfloor log2(C_k) \rfloor + 1) + \sum_{k=1}^{K} \sum_{n=1}^{N} \left(\lfloor log2(R_{k,n} - 1) \rfloor + 1 \right)$.

3) Move the problem constraints into the objective function by converting them to an appropriate quadratic penalty to form a single quadratic expression equivalent to the form $\boldsymbol{x^T Q x}$ as follows.

$$\boldsymbol{x^T C x + P(Ax - b)^T (Ax - b)} \tag{5}$$

$$\boldsymbol{= x^T C x + x^T D x + C} \tag{6}$$

$$\boldsymbol{= x^T Q x + C} \tag{7}$$

where P is the penalty scalar, the matrix **D**, and the additive constant c result directly from the matrix multiplication indicated above. The suitable penalty scalar P is chosen such that the optimal solution to QUBO is the optimal solution to the original constrained problem.

4) Finally, the maximization problem is transformed into a minimization problem since the LQS finds the minimum of objective function. This can be done by negating all the coefficients of the objective function. Dropping the additive constant $\boldsymbol{C}$ in (7), the equivalent unconstrained version of the constrained problem becomes:

$$\text{minimize} - \boldsymbol{x^T Q x} \tag{8}$$

4.3 Evaluation

In this section, the performance of the LQS to solve the joint sub-carrier selection and rate allocation problem defined in section 4.2 is evaluated. Our LQS results are compared to the solutions obtained by solving the ILP and QUBO problems using ILP and Classical QUBO solvers executed on conventional CPUs.

4.3.1 Benchmark Solutions

In this sub-section, we benchmark the solutions. First, we present the classical QUBO solvers in detail. The presentation of the simulation setup and simulation results follows this.

The ILP problem has been solved using *intlinprog* function on MATLAB. The QUBO problem has been solved using four different algorithms:

- Qbsolv - D-Wave's Simulated Annealing (DSA): Qbsolv is a classical CPU-based decomposing solver that divides a large QUBO problem into sub-problems to find the minimum value [43]. The sub-problems are solved using DSA or TS. DSA implements the simulated annealing algorithm, which is based on the technique of cooling metal from a high temperature to improve its structure (i.e., annealing) [44].
- Qbsolv - Tabu Search (TS): TS is based on a tabu search algorithm which is selected as the default by Qbsolv.
- Qubovert's Simulated Annealing (QSA): Qubovert is a single package for formulating, simulating, and solving boolean and spin problems [45].
- Lava QUBO solver (LQS): LQS is a solver for QUBO optimization problems supported by the Lava framework.

4.3.1.1 Classical QUBO Solvers:

In this sub-section, we discuss the classical CPU-based solvers which are used for the execution of QUBO problems to evaluate the performance of the Lava QUBO solver. First, we present the QBSolv- D-Wave's Simulated Annealing and Tabu Search. Then we present the Qubovert's Simulated Annealing.

1. QBSolv:

Qbsolv is a classical CPU-based decomposing solver that divides a large QUBO problem into sub-problems to find the minimum value [43]. The sub-problems are solved using DSA or TS. DSA implements the simulated annealing algorithm, which is based on the technique of cooling metal from a high temperature to improve its structure (i.e., annealing) [44]. Tabu Search (TS) is based on a tabu search algorithm which is selected as the default by Qbsolv.

1.A Tabu Search:

F. Glover introduced the Tabu Search algorithm in 1986, a global optimization algorithm step by step. First, the current solution must be created randomly before being followed by a search for several of its neighbors, and lastly, the best of those neighbors must be selected as a new current solution. In order to avoid repeatedly searching for the local optimal solution and to arrive at the problem's global optimal solution during the Tabu Search, some historical information pertaining to the search's evolution should be kept, and this information will be used to direct the movement from one solution to the next while avoiding cycling. As a result, TSA keeps track of the most recently searched local optimum solutions using a taboo list [46].

Neighborhood taboo list, taboo length, candidate set aspiration criterion, and termination criterion are all components of the TS algorithm process. The taboo list is one of the key factors determining a TS algorithm's quality. The recently visited solutions, or the moves to a solution to escape from local optima and reach the global satisfactory solution, make up the most popular taboo list. As the candidate solution set, we can choose a number of the current solution's neighbors. When all candidate solutions are taboo, or a candidate solution superior to the currently optimal solution is taboo, the aspiration criterion will ensure the search process can release a specific solution and

enable an effective global optimization search. The computation will end according to the termination criterion after evolving through a predetermined number of steps [46]. Figure 8 depicts the TS procedure as follows.

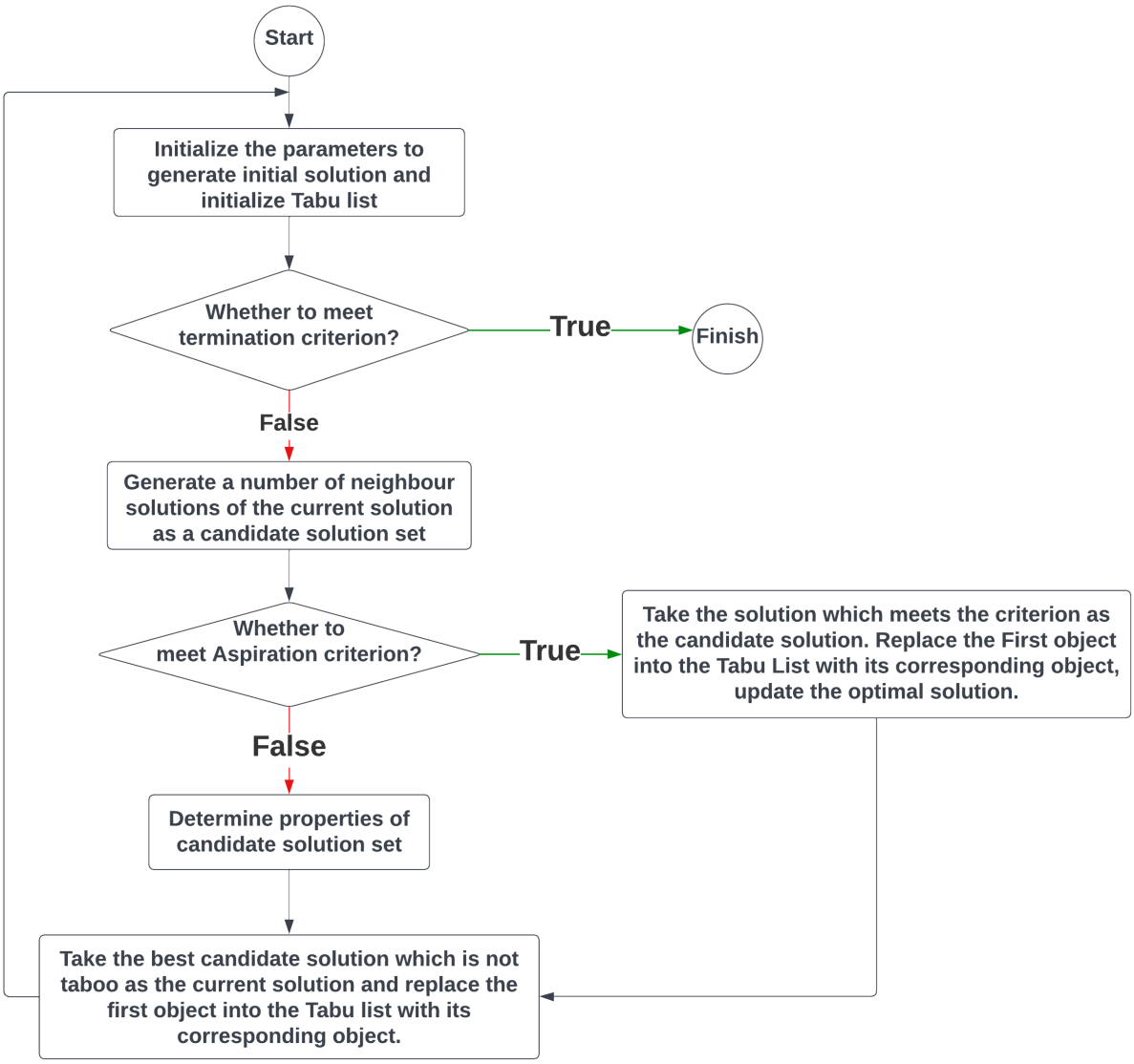

Figure 8 Tabu search framework [46]

1.B Simulated Annealing:

The name Simulated Annealing and its inspiration come from the metallurgical process of annealing. An increase in crystal size and a decrease in crystal flaws are achieved through the controlled heating and cooling processes of annealing. A stable state will be present in the system with large crystals that can regularly form next to one another, have minimal energy, and satisfy our needs if the temperature decreases very slowly. If the temperature decreases quickly, the crystals will not have enough time to form regular and stable structures due to their high energy, and they cannot reach a stable state. SA gradually lowers the temperature while considering the neighboring state at each temperature. If the energy of the neighbor is greater than the present state, SA moves the system to the neighbor state. However, if the energy of the neighbor is lower than the current state, SA applies an acceptance function to determine whether or not to move to the neighbor state. Since the system may currently enter a local minimum state and this movement requires voiding the local minimum state, SA will occasionally employ a bad state based on the acceptance function to reach the final state in the future [47].

Three factors should be determined before applying SA.

(1) A starting point is a location in the search space where searching begins.

(2) Neighbor state generation, the generation of neighbor states.

(3) Annealing schedules with those parameters include instructions on how to lower the temperature.

Figure 9 illustrates the steps of SA algorithm.

2. Qubovert

Qubovert is a single package for formulating, simulating, and solving boolean and spin problems [45]. Qubovert's anneal_qubo function executes a simulated annealing algorithm to

determine the minimum of the QUBO given by Q. The general definition of anneal_qubo function is as shown below:

anneal_qubo(*Q*, *num_anneals*, *anneal_duration*, *initial_state*, *temperature_range*, *schedule*, *in_order*, *seed*)

Initialization(Current_Solution, Temperature)
Calculation of the Current_Cost
LOOP
    New_State
    Calculation of the New_Cost
    IF Δ(Current_cost – New_Cost) $\leq$ 0 THEN
        Current_State = New_State
    ELSE
        IF Exp($\frac{\text{Current\_cost} - \text{New\_Cost}}{\text{Temperature}}$) > Random (0,1)
    THEN
        --Accept
        Current_State = New_State
    ELSE
        --Reject
    Decrease the temperature
EXIT When STOP_CRITERION
END LOOP

Figure 9 Simulated Annealing Algorithm [47]

The parameters of the anneal_qubo function are explained as follows,

- Q – (type- dictionary)- defines a QUBO dictionary that maps the Boolean labels to their values in the objective function. The utility function matrix_to_qubo can be used to convert the Q matrix to a QUBO dictionary.
- num_anneals (type- int>1)– defines the number of times the simulated annealing algorithm is executed. The default value is one.
- anneal_duration (type- int>1)– defines the total number of updates to the simulation during the anneal. This is related to the amount of time spent in the cooling schedule. The anneal_duration will not be considered if an explicit schedule is given. Default value is 1000.
- initial_state (type- dictionary)– The initial state to begin the anneal process. The default value is None. when initial_state is None, a random state will be selected to begin each anneal. Else, initial_state will be the beginning state for all of the anneals.
- temperature_range (type - tuple)- defines the temperature to begin and end the anneal at. Default values is taken as None.
- schedule (type- str or iterable of tuple)– defines the cooling schedule to be used. Examples are linear and geometric. When the schedule is linear, then the cooling schedule will be a linear interpolation between the values in temperature_range. If the schedule is geometric, then the cooling schedule will be a geometric interpolation between the values in temperature_range. The default value is 'geometric'.
- in_order(type - bool) – During the update step, whether the variables should be iterated in order or random is defined by in_order parameter. The simulation is more physically realistic if in_order value is False. Else, when using the simulation for annealing, the value is set to True for better results. The default value is set to True.

- Seed(type-int) – Used to set Python's built-in random module seed. If the value is None, random.seed will not be called. The default value is set to True.

In order to estimate the execution time on Loihi hardware, we relied on [48], in which a series of experiments is presented using a random walk algorithm, with an execution time of around 7.5-million-time steps over 42 seconds. In other words, Loihi spends 5.6μs per time step. Thus, using this time proportion per time step and the number of time steps resulting from solving problem instances listed in Table 4 on LQS with CPU as the backend, the corresponding estimated execution times on Loihi are computed. As a lower bound estimation on the Loihi 2 hardware, we also use 199 ns per time step based on Intel's document [49]. We note that this is only a rough estimation, as the actual runtime heavily depends on the complexity of the actual spiking neural network and its embedding onto the hardware.

4.3.2. Simulation Setup

In this sub-section, we present the simulation setup implementing the QUBO problem in the Lava framework.

Implementing the QUBO problem in the Lava framework involves the following three steps:

1) Define the QUBO workload (i.e., the optimization problem),
2) Provide a set of hyperparameters, and
3) Execute the Lava solver.

In the Lava framework, the QUBO problem is encoded in the form of a QUBO matrix. Once the problem has been encoded, it is then provided to Lava's Optimization Solver, which is a generic solver for constraint optimization problems. As input for each optimization problem, the solver

Table 4. Execution Results

Problem Instance	Problem Size (Binary variables)	Execution Time (in ms)	
		(MATLAB and Classical QUBO Solvers)	Lava QUBO Solver (LQS)
P1 K = 2 N = 2 $R_{k,n}$ = [2 1; 1 2] C_k = [2, 2]	20	• MATLAB: ILP = 187.1 ms • Classical QUBO Solvers: • DSA = 25.43 ms • Tabu = 30.28 ms • QSA= 39.32 ms	• CPU = 288.45 ms (sts-57) • Estimation on Loihi = 0.31920 ms • Estimation on Loihi 2 = 0.01083 ms • Hyperparameters = 24, 80, 4, seed=1
P2 K = 2 N = 2 $R_{k,n}$ = [3 1; 1 4] C_k = [4, 4]	25	• MATLAB: ILP = 205.7 ms • Classical QUBO Solvers: • DSA = 28.84 ms • Tabu = 27.86 ms • QSA= 61.16 ms	• CPU = 3496 ms (sts-4614) • Estimation on Loihi = 25.83840 ms • Estimation on Loihi 2 = 0.87666 ms • Hyperparameters = 34, 74, 4, seed=323
P3 K = 2 N = 3 $R_{k,n}$ = [2 1 2; 1 2 1] C_k = [3, 2]	28	• MATLAB: ILP = 209.1 ms • Classical QUBO Solvers: • DSA = 29.14 ms • Tabu = 25.66 ms • QSA= 134.8 ms	• CPU = 7389.8 ms (sts-8810) • Estimation on Loihi = 49.33600 ms • Estimation on Loihi 2 = 1.67390 ms • Hyperparameters = 42, 46, 4, seed=173
P4 K = 3 N = 3 $R_{k,n}$ = [2 1 1; 1 1 2; 1 2 1] C_k = [2, 3, 2]	39	• MATLAB: ILP = 219.9 ms • Classical QUBO Solvers: • DSA = 37.44 ms • Tabu = 62.5 ms • QSA= 187.15 ms	• CPU = 36009 ms (sts-63098) • Estimation on Loihi = 353.3488 ms • Estimation on Loihi 2 = 11.9886 ms • Hyperparameters = 52, 51, 3, seed=42

requires (i) two stopping conditions (i.e., timeout and target cost), (ii) a backend, and (iii) four hyperparameters. The solver keeps running until one of the stopping conditions is reached. Once the execution is completed, both the solution to the QUBO optimization problem and the number of time steps needed to reach that solution are obtained. Lava's optimization solver hyperparameters are provided in the form of dictionary type and are described as follows:

- The steps to fire hyperparameter give the number of time steps it requires a neuron to fire in the absence of noise.
- The step size hyperparameter denotes the value added in each time step to the state variable.
- The noise amplitude hyperparameter is the multiplicative factor to upscale the stochastic noise in the network.
- The init value is a vector providing an initial value for the variables defining the problem.

The quality of the solution (i.e., optimal or sub-optimal) and the execution time of the LQS highly depend on the choice of the hyperparameters values. Figure 10 describes the impact of the hyperparameter values on the problem instance P3 defined in Table 4. We can see that the number of steps needed to obtain a solution has drastically decreased from 568197 to 26028, when the hyperparameters are set to optimal values. Once this process of tuning the hyperparameters is performed, the solve method of Lava's optimization solver is called to tackle the provided workload. To conduct performance evaluation, we use CPU as the backend. Although CPU is used, Lava's solver mimics the algorithm developed for Loihi 2 by default. Changing the backend to "Loihi2" will execute the problem on Loihi 2 neuromorphic chip if hardware access is available. The results are obtained by running LQS on an AMD EPYC 7702 virtual machine configured with 30 vCPU cores out of 64 cores. Moreover, the version of the Lava framework on which we ran the problem is Lava Optimization 0.2.2. We also considered updates on it until 2022 November 14.

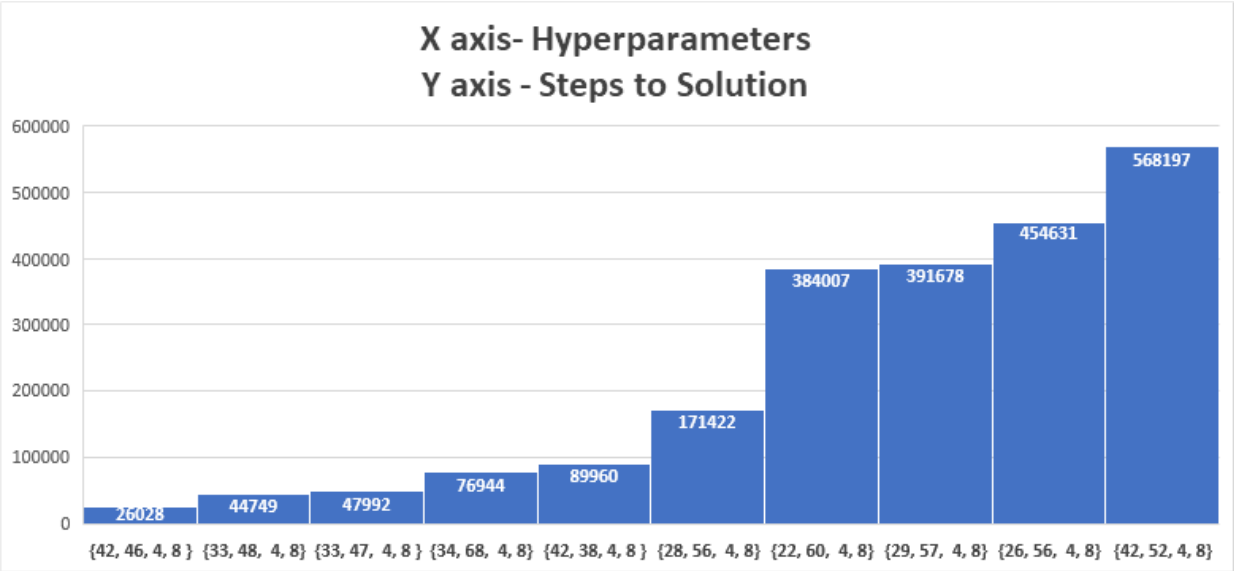

Figure 10 Impact of the Hyperparameters on the Number of Steps Needed to Reach an Optimal Solution.

4.3.3 Simulation Results

In this sub-section, the performance results are presented for different problem instances, as shown in Table 4. For comparison, we have considered four different problem instances P1-P4. P1 and P2 are composed of two users and two sub-carriers, while P3 and P4 have two users, three subcarriers, and three users and three sub-carriers, respectively. From Table 4, it is observed that problem instances P1 to P4 executed on DSA show around 6-7 factors gain against MATLAB, while TS shows a minimum of 3 to a maximum of 8 factors, and QSA shows approximately 1-5 times faster than MATLAB. However, the same problem is executed on LQS with CPU as the backend does not show any gains against other solvers. The estimated execution time on Loihi shows a maximum gain of 586x against P1 executed on MATLAB, while the least gain on P1 was against DSA with slightly below 80x gain. The least gain on Loihi 2 was against the P4 run on DSA, showing slightly above three orders of magnitude faster. While a maximum gain of more than 17000x was observed against P1 executed on MATLAB.

Our results show that executing optimization problems on the Lava framework with CPU as the backend does not offer any gains on performance. This can be explained as follows. First, Lava currently implements a spiking version of a Boltzmann machine inspired by the work of Jonke et al. [25], which is suitable for the asynchronous computing architecture of neuromorphic hardware but is not suitable for the CPU backend. In addition, this can be explained by the exponentially increased number of optimization variables resulting from the conversion of the ILP to a QUBO problem.

4.3.4 Discussion

The execution time of the algorithms is highly impacted by the computer's underlying architecture. While we are already executing our problem on a 30-CPU core machine, we believe that adding even more cores would not provide the necessary speedup of the algorithms. The classical algorithms slow down more as the problem size increases. The real benefit of neuromorphic chips will be clear for large-scale problems, as they perform thousands of times faster and can easily outperform their classical counterparts [50]. Lava's QUBO optimization framework is undergoing continuous development to improve the performance of the QUBO solver. Currently, the Solver-Tuner utility function is used to optimize hyperparameters by random search. The solutions on Loihi 2 may differ due to slightly different noise models. The noise parameter value in the Lava CPU model for the stochastic integrate and fire process is set to the noise amplitude value multiplied by a randomly generated value. Thus, the number of timesteps required to reach solutions for the same input of hyperparameters is different for different random generation seed values. Therefore, finding the best hyperparameters values implies selecting the best seed values along with hyperparameters to obtain optimal solutions. The timesteps needed to reach solutions considering hyperparameters tuning are as listed in Table 4. Currently, tuning the

hyperparameters is a tedious task. Determining the optimal set of hyperparameters is even more difficult for large problem instances. Thus, we believe that improving the Solver- Tuner utility for hyperparameter tuning can help solve large problems and achieve performance gains, which is beyond the scope of this work.

The power consumption of Loihi for a wide range of applications is reported to be under 1W [49]. However, the default power consumption of the used AMD EPYC 7702 CPU is 200W, which is a typical power consumption of a server CPU these days. Thus, significant gains in power consumption are expected in addition to those in execution times when the execution is performed on Loihi and Loihi 2 chips.

4.3.5. Chapter Summary

In this chapter, we discussed the case study of radio resource allocation using QUBO optimization solvers. We explained the downlink OFDMA system model and formulated the resource allocation problem for the downlink OFDMA system as an ILP optimization problem which maximizes the total system throughput. Then, we reduced the ILP formulation to QUBO form to solve it on the Lava QUBO optimization solver. Next, we evaluate the performance of the LQS to solve the joint sub-carrier selection and rate allocation problem. The results are compared to the solutions obtained by solving the ILP and QUBO problems using ILP and other standard QUBO solvers executed on conventional CPUs. Currently, tuning the hyperparameters is a tedious task. Determining the optimal set of hyperparameters is even more difficult for large problem instances. Thus, we believe that improving the Solver-Tuner utility for hyperparameter tuning can help solve large problems. Also, our results show that executing optimization problems on the Lava framework with CPU as the backend does not offer any gains on performance. However,

significant gains are observed for estimated execution times on neuromorphic chips Loihi and Loihi 2.

5. Conclusion

In this chapter, we begin with a brief overview of the thesis' contribution and then discuss the potential directions for future research.

5.1 Contribution Summary

Solving resource allocation problems in wireless communication systems on traditional computer architectures is becoming increasingly challenging. In addition, meeting the real-time requirements of these systems, such as the transmission time interval (TTI) of one millisecond or less, is particularly difficult in this case. Neuromorphic computing offers a promising alternative approach to address these complex problems. It draws inspiration from the design and operation of the human brain. In comparison to conventional systems, neuromorphic architectures have the potential to achieve orders of magnitude increases in energy efficiency and performance. Therefore, there is a need to investigate the potential benefits and limitations of using neuromorphic software frameworks to solve radio resource allocation problems.

To address this need, we presented a motivation for investigating the use of the Lava neuromorphic framework for radio resource allocation and evaluating its performance in comparison to traditional approaches. The use of neuromorphic computing frameworks for wireless communication issues is gaining popularity, and research is ongoing in this field. We review state-of-the-art works on neuromorphic software frameworks and resource allocation in the OFDMA system. The possible advantages and disadvantages of adopting neuromorphic software frameworks to address radio resource allocation issues, however, have not been examined in earlier research works.

Next, we design and formulate a radio resource allocation problem as an ILP optimization problem in the downlink OFDMA system. The objective of the resource allocation problem aims to maximize the total system throughput. Indeed, the sum-rate maximization is achieved by maximizing the individual user allocation rate in such a way that the total system throughput is increased. This is then solved optimally on MATLAB integer linear programming solver. Then, the ILP problem is converted into a QUBO problem to map the problem in the neuromorphic domain using the Lava framework.

This was followed by the implementation of the QUBO problem on other standard QUBO solvers executed on conventional CPUs. These classical solvers include Qbsolv - D-Wave's Simulated Annealing, Qbsolv - Tabu Search, and Qubovert's Simulated Annealing. For evaluations, we have considered four different problem instances P1-P4 where P1 and P2 are composed of two users and two sub-carriers. At the same time, P3 and P4 have two users, three subcarriers, and three users and three sub-carriers, respectively.

Finally, the results of the Lava QUBO solver are compared to the solutions obtained by solving the ILP and QUBO problems using ILP and other standard QUBO solvers executed on a conventional CPU. In addition, we also estimate the execution time on Loihi and Loihi 2 hardware. It should be noted that the estimated runtime is only an approximate value, as the actual duration is highly dependent on the complexity of the spiking neural network and its embedding on the hardware. Solving problems on Lava requires several hyperparameters as inputs. However, the current hyperparameters tuning process is very challenging for the execution of large problems since it is performed manually. Our results show that executing optimization problems on the Lava framework with CPU as the backend does not offer any gains on performance. However,

significant gains are observed for estimated execution times on neuromorphic chips Loihi and Loihi 2.

5.2 Future Work

In this work, we conducted experiments on small-scale resource allocation problems to demonstrate the efficiency of the LQS approach. However, to evaluate the practical applicability of the proposed approach, it is necessary to extend these experiments to large-scale problems. Therefore, the future works can extend the case study experiments to large-scale resource allocation problems once hyperparameters tuning is automated.

In addition, the work in this thesis did not include execution on neuromorphic hardware chip. We can execute the problem on a neuromorphic hardware chip Loihi 2 from the Lava framework in future works. One of the significant advantages of the LQS approach is its potential for implementation on neuromorphic hardware Loihi 2. Therefore, executing the proposed approach on the Loihi 2 chip from the Lava framework can demonstrate the feasibility of implementing the approach in real-world scenarios.

Moreover, with the emergence of several computing paradigms, such as quantum computing, it is crucial to compare the performance of neuromorphic computing with these emerging paradigms. The formulated QUBO problem can be executed on D-wave Quantum computers. Therefore, in future works, we aim to compare the performance of LQS with quantum computing approaches in solving resource allocation problems. This will provide valuable insights into the strengths and limitations of different computing paradigms in solving resource allocation problems and help identify the most effective approach for different problem scenarios.

In conclusion, the future work proposed in this section aims to extend this work and provides new insights into the potential of neuromorphic computing for solving resource allocation problems.

Bibliography

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963,* 2017.

[2] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE,* vol. 109, pp. 911-934, 2021.

[3] Lava team, *Lava Software Framework,* 2022, Accessed: 08-November-2022. [Online]. Available: https://lava-nc.org/

[4] H. Yin and S. Alamouti, "OFDMA: A broadband wireless access technology," in *2006 IEEE sarnoff symposium*, 2006.

[5] E. K. P. Chong and S. H. Zak, An introduction to optimization, vol. 75, John Wiley & Sons, 2013.

[6] G. Sierksma and Y. Zwols, Linear and integer optimization: theory and practice, CRC Press, 2015.

[7] F. Glover, G. Kochenberger, R. Hennig and Y. Du, "Quantum bridge analytics I: a tutorial on formulating and using QUBO models," *Annals of Operations Research,* vol. 314, p. 141–183, 2022.

[8] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu and Q. Qiu, "A Survey on Neuromorphic Computing: Models and Hardware," *IEEE Circuits and Systems Magazine,* vol. 22, pp. 6-35, 2022.

[9] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra and A. Wild, "NxTF: An API and compiler for deep spiking neural networks on Intel Loihi," *ACM Journal on Emerging Technologies in Computing Systems (JETC),* vol. 18, p. 1–22, 2022.

[10] ROS team, *ROS wiki,* 2022. Accessed: 08-November-2022. [Online]. Available: https://wiki.ros.org

[11] D. F. M. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in python," *Frontiers in neuroinformatics,* p. 5, 2008.

[12] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker and C. Eliasmith, "Nengo: a Python tool for building large-scale functional brain models," *Frontiers in neuroinformatics,* vol. 7, p. 48, 2014.

[13] Nengo team, *A Python library for creating and simulating large-scale brain models,* 2022. Accessed: 31-December-2022.[Online]. Available: https://github.com/nengo/nengo

[14] H. E. Plesser, M. Diesmann, M.-O. Gewaltig and A. Morrison, *NEST: The Neural Simulation Tool.,* 2014.

[15] PyTorch team, *PyTorch website,* 2022. Accessed: 31-December-2022.[Online]. Available: https://pytorch.org

[16] Tensorflow team, *Tensorflow website,* 2022. Accessed: 31-December-2022.[Online]. Available: https://www.tensorflow.org

[17] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience,* vol. 11, p. 682, 2017.

[18] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet and P. Yger, "PyNN: a common interface for neuronal network simulators," *Frontiers in neuroinformatics,* vol. 2, p. 11, 2009.

[19] M. Davies, "New Tools for a New Era of Neuromorphic Computing," in *Neuro Inspired Computational Elements Conference (NICE 2022)*, 2022.

[20] M. Z. Alom, B. Van Essen, A. T. Moody, D. P. Widemann and T. M. Taha, "Quadratic Unconstrained Binary Optimization (QUBO) on neuromorphic computing system," in *International Joint Conference on Neural Networks (IJCNN)*, 2017.

[21] S. R. Kulkarni, M. Parsa, J. P. Mitchell and C. D. Schuman, "Benchmarking the performance of neuromorphic and spiking neural network simulators," *Neurocomputing,* vol. 447, pp. 145-160, 2021.

[22] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," *IEEE Letters of the Computer Society,* vol. 1, pp. 17-20, 2018.

[23] A. R. Voelker and C. Eliasmith, "Programming Neuromorphics Using the Neural Engineering Framework," *Handbook of Neuroengineering,* pp. 1-43, 2020.

[24] M. Morelli, C.-C. J. Kuo and M.-O. Pun, "Synchronization techniques for orthogonal frequency division multiple access (OFDMA): A tutorial review," *Proceedings of the IEEE,* vol. 95, p. 1394–1427, 2007.

[25] E. Avdotin, D. Bankov, E. Khorov and A. Lyakhov, "OFDMA resource allocation for real-time applications in IEEE 802.11 ax networks," in *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2019.

[26] X. Zhang, S. Chen and W. Wang, "Multiuser radio resource allocation for multiservice transmission in OFDMA-based cooperative relay networks," *EURASIP Journal on Wireless Communications and Networking,* vol. 2009, p. 1–13, 2008.

[27] S. Dimitrov, G. Boccolini, S. Jaeckel, D. Benfatto, N. Privitera, R. Suffritti, A. B. Awoseyila and B. G. Evans, “FFT-based waveforms for high throughput satellite communications: opportunities and challenges,” *Proc. KABAND-2014,* 2014.

[28] J. Mashino and T. Sugiyama, “Subcarrier suppressed transmission for OFDMA in satellite/terrestrial integrated mobile communication system,” in *2011 IEEE International Conference on Communications (ICC)*, 2011.

[29] J. Y. Won, S. B. Shim, Y. H. Kim, S. H. Hwang, M. S. Song and C. J. Kim, “An adaptive OFDMA platform for IEEE 802.22 based on cognitive radio,” in *2006 Asia-Pacific Conference on Communications*, 2006.

[30] P. Korrai, E. Lagunas, S. K. Sharma, S. Chatzinotas, A. Bandi and B. Ottersten, “A RAN Resource Slicing Mechanism for Multiplexing of eMBB and URLLC Services in OFDMA Based 5G Wireless Networks,” *IEEE Access,* vol. 8, pp. 45674-45688, 2020.

[31] P. Zhang, H. Tian, P. Zhao and S. Fan, “Context-aware Mobile Edge Resource Allocation in OFDMA Downlink System,” *IEEE Transactions on Network Science and Engineering,* 2022.

[32] A. Chernov, V. Matyukhin, M. Somov, E. Barashov, J. Chernova and V. Mishin, “Radio Resource Management in 5G networks,” in *International Conference Engineering and Telecommunication*, 2020.

[33] Y. M. Allawi, G. Y. Kim, J. Cho, A. G. Reza, Y. Na and J.-K. K. Rhee, “Optimal dynamic subcarrier allocation for dual-band OFDMA-PON supporting integrated fronthaul and backhaul in 5G networks,” *ICT Express,* vol. 4, p. 138–143, 2018.

[34] L. Saraswat, M. Gupta, S. Singh and P. Rana, “OFDMA BASED RCO MAC PROTOCOL FOR TERAHERTZ WIRELESS COMMUNICATION IN VEHICULAR 6G NETWORK,” *International Journal of Advanced Research in Engineering and Technology,* vol. 12, pp. 1060-1072, 2021.

[35] H. B. Salameh, H. Al-Obiedollah, R. Mahasees and Y. Jararweh, “Opportunistic non-contiguous OFDMA scheduling framework for future B5G/6G cellular networks,” *Simulation Modelling Practice and Theory,* vol. 119, p. 102563, 2022.

[36] S. Ismail, C. K. Ng and N. K. Noordin, “Fairness resource allocation for downlink ofdma systems,” in *IEEE 9th Malaysia International Conference on Communications (MICC)*, 2009.

[37] M. O. Kabaou, N. Zoghlami and J. Sghaier, “A Novel Fusion Approach Bandwidth Allocation for OFDMA Systems: Application to New Generation Mobile Networks,” in *2022 5th International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, 2022.

[38] S. Zeng, H. Zhang, B. Di and L. Song, “Trajectory Optimization and Resource Allocation for OFDMA UAV Relay Networks,” *IEEE Transactions on Wireless Communications,* vol. 20, pp. 6634-6647, 2021.

[39] T.-T. Nguyen, Q.-V. Pham, V.-D. Nguyen, J.-H. Lee and Y.-H. Kim, “Resource allocation for energy efficiency in OFDMA-enabled WPCN,” *IEEE Wireless Communications Letters,* vol. 9, p. 2049–2053, 2020.

[40] T. Q. Wu and H. C. Yang, “On the Performance of Overlaid Wireless Sensor Transmission With RF Energy Harvesting,” *#IEEE_J_JSAC#,* vol. 33, pp. 1693-1705, August 2015.

[41] A. A. Nasir, X. Zhou, S. Durrani and R. A. Kennedy, “Wireless-Powered Relays in Cooperative Communications: Time-Switching Relaying Protocols and Throughput Analysis,” *#IEEE_J_WCOM#,* vol. 63, pp. 1607-1622, May 2015.

[42] M. Saravanan and R. Pratap Sircar, “Quantum Evolutionary Algorithm for Scheduling Resources in Virtualized 5G RAN Environment,” in *IEEE 4th 5G World Forum (5GWF)*, 2021.

[43] D-wave team, *Qbsolv solver introduction page,* 2022. Accessed: 31-December-2022.[Online]. Available: https://docs.ocean.dwavesys.com/projects/qbsolv/en/latest/

[44] D-wave team, *Simulated annealing index page,* 2022. Accessed: 31-December-2022.[Online]. Available: https://docs.ocean.dwavesys.com/projects/neal/en/latest/index.html

[45] J. T. Iosue, *Qubovert's documentation page,* 2022. Accessed: 31-December-2022.[Online]. Available: https://qubovert.readthedocs.io/en/latest/index.html

[46] C.-h. Guan, Y. Cao and J. Shi, “Tabu search algorithm for solving the vehicle routing problem,” in *2010 Third International Symposium on Information Processing*, 2010.

[47] M. M. Keikha, “Improved Simulated Annealing Using Momentum Terms,” in *2011 Second International Conference on Intelligent Systems, Modelling and Simulation*, 2011.

[48] J. D. Smith, W. Severa, A. J. Hill, L. Reeder, B. Franke, R. B. Lehoucq, O. D. Parekh and J. B. Aimone, “Solving a steady-state PDE using spiking networks and neuromorphic hardware,” in *International Conference on Neuromorphic Systems 2020*, 2020.

[49] Intel, *New Technologies newsroom page,* 2022. Accessed: 31-December-2022.[Online]. Available: https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf

[50] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, A. Beigh and S. Douglass, “High speed cognitive domain ontologies for asset allocation using loihi spiking neurons,” in *International Joint Conference on Neural Networks (IJCNN)*, 2019.

[51] C. Yakopcic, N. Rahman, T. Atahary, M. Z. Alom, T. M. Taha, A. Beigh and S. Douglass, “Spiking Neural Network for Asset Allocation Implemented Using The TrueNorth System,” in *IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*, 2019.

[52] Z.-H. Wu, B. Wang, C. Jiang and K. R. Liu, “Downlink MAC scheduler for 5G communications with spatial focusing effects,” *IEEE Transactions on Wireless Communications,* vol. 16, p. 3968–3980, 2017.

[53] V. Vahidi and E. Saberinia, “OFDM high speed train communication systems in 5G cellular networks,” in *IEEE Proc. CCNC*, 2018.

[54] V. Vahidi and E. Saberinia, “OFDM high speed train communication systems in 5G cellular networks,” in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2018.

[55] K. Seong, M. Mohseni and J. M. Cioffi, “Optimal resource allocation for OFDMA downlink systems,” in *IEEE International Symposium on Information Theory*, 2006.

[56] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay and others, “Opportunities for neuromorphic computing algorithms and applications,” *Nature Computational Science,* vol. 2, pp. 10-19, 2022.

[57] C. D. Schuman, J. S. Plank, A. Disney and J. Reynolds, “An evolutionary optimization framework for neural networks and neuromorphic architectures,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

[58] O. Rhodes, P. A. Bogdan, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. R. Lester, M. Mikaitis, L. A. Plana, A. G. D. Rowley and others, “sPyNNaker: a software package for running PyNN simulations on SpiNNaker,” *Frontiers in neuroscience,* vol. 12, p. 816, 2018.

[59] J. Nocedal and S. J. Wright, “Quadratic programming,” *Numerical optimization,* p. 448–492, 2006.

[60] J. P. Mitchell, C. D. Schuman, R. M. Patton and T. E. Potok, “Caspian: A neuromorphic development platform,” in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020.

[61] Z. Jonke, S. Habenschuss and W. Maass, “Solving constraint satisfaction problems with networks of spiking neurons,” *Frontiers in neuroscience,* vol. 10, p. 118, 2016.

[62] X. Huang and R. de Vegt, *The Benefits of OFDMA for Wi-Fi 6,* 2021. Accessed: 01-December-2022. [Online]. Availbale: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/ofdma_white_paper.pdf

[63] J. D. Hidary and J. D. Hidary, Quantum computing: an applied approach, vol. 1, Springer, 2019.

[64] A. G. Gotsis, D. I. Komnakos, D. D. Vouyioukas and P. Constantinou, “Radio resource allocation algorithms for multi-service OFDMA networks: the uniform power loading scenario,” *Telecommunication Systems,* vol. 56, pp. 467-480, 2014.

[65] D-wave team, *Tabu solver introduction page,* 2022. Accessed: 01-December-2022. [Online]. Availbale: https://docs.ocean.dwavesys.com/projects/tabu/en/latest/intro.html

[66] Lava library team, *Lava optimization library notebooks,* 2022. Accessed: 01-December-2022. [Online]. Availbale: https://github.com/lava-nc/lava-optimization/blob/main/tutorials/tutorial_02_solving_qubos.ipynb

[67] PyNN team, *A Python package for simulator-independent specification of neuronal network models.,* 2022. Accessed: 01-December-2022. [Online]. Availbale: https://neuralensemble.org/PyNN/

[68] J. Aimone, P. Date, G. Fonseca-Guerra, K. Hamilton, K. Henke, B. Kay, G. Kenyon, S. Kulkarni, S. Mniszewski, M. Parsa and others, "A review of non-cognitive applications for neuromorphic computing," *Neuromorphic Computing and Engineering,* 2022.