Imperial College London
Department of Computing

FPGA-based network switches and their applications

Jiuxi Meng

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, 2023

Declarations

Statement of Originality

I Jiuxi Meng of Imperial College London hereby declare that this thesis is my original work, and all the sources used in this study have been appropriately cited and acknowledged. I have not submitted this work, or any part thereof, for any other academic qualification or assessment, and it has not been previously published in any form.

I affirm that the research presented in this thesis was conducted under my supervision, and all the data, results, and conclusions are genuine and authentic. I have adhered to the ethical guidelines and protocols of the institutions and organisations involved in this research, and I have obtained all necessary permissions and clearances for the use of copyrighted materials and human subjects.

In conclusion, I affirm that this thesis represents my original and independent research, and it is a true reflection of my academic abilities and intellectual potential.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-NonCommercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

This thesis investigates the potential of network switches based on FPGA (field-programmable gate array) technology for high-performance applications. There are three contributions. The first contribution quantifies the gap between current ASIC-based (application-specific integrated circuit) switches and FPGA-based switches. We propose a novel buffer-sharing design to make efficient use of the FPGA on-chip memory. We also develop techniques for predicting the performance and resource requirements of the proposed switch design. The second contribution is a novel architecture for random projection and ensemble learning that targets an FPGA-based switch, utilising spare computational resources on the FPGA. This design is capable of processing high-dimensional data effectively for training ensemble models. We also create a model for efficient load distribution between the switch and the downstream servers. The proposed architecture enables a reduction in ensemble training time while maintaining high accuracy, with up to 6.7 times speed-up. The third contribution explores the benefit of kNN (k nearest neighbour) classification using the Hamming distance metric. We introduce a new architecture that enables complete application offloading to an FPGA-based switch. Our results show an estimated speed-up of up to 8.7 times and a memory reduction of up to 27.7 times.

Acknowledgements

I would like to express my deepest appreciation and gratitude to Prof Wayne Luk, Dr Paolo Costa and Dr Ce Guo, for their invaluable guidance, support, and encouragement throughout my doctoral journey. Their expertise, dedication, and commitment to excellence have been instrumental in shaping my research and helping me reach this important milestone.

I am also grateful to my collaborators Nadeen Gebara for contributing to Section 3.3.1, Philippos Papaphilippou for contributing to Section 3.5 and Zhiqiang Que for contributing to Section 5.5. They have generously shared their knowledge, resources and expertise. Their constructive feedback, valuable suggestions, and collaborative spirit have enriched my academic experience and expanded my research horizons.

I would also like to extend my heartfelt appreciation to my friends Jiankang Deng, Pingchuan Ma, Yujiang Wang and Jing Yang, who have been my constant source of inspiration, motivation, and moral support, especially during the challenging times of the COVID-19 pandemic. Their company, kind words, and shared experiences have made this journey less lonely and more fulfilling.

I would also like to thank my lovely colleagues (Huxley 353) turned friends Nadeen Gebara, Ho-Cheung Ng and Ruizhe Zhao, and our special guest Marco Barbone, for their continuous support, encouragement, and laughter. Their presence in my life has made the journey more joyful and meaningful.

Last but not least, I want to express my deepest gratitude to my parents. Without their love, sacrifice, and unwavering support, I would not have been able to pursue my dreams and achieve this significant accomplishment. Their unwavering encouragement, guidance, and love have been my constant source of inspiration and motivation.

Dedication

I dedicate this thesis to my parents, who have been my pillars of strength and support throughout my academic journey. Their unwavering encouragement, guidance, and love have been instrumental in shaping me into the person I am today.

I am forever grateful for the sacrifices they have made to provide me with the best possible education and opportunities. Their unconditional love and support have been my constant source of strength and inspiration. I hope this thesis serves as a small token of my appreciation for everything they have done for me.

Publications

- Nadeen Gebara, Jiuxi Meng, Wayne Luk, and Paolo Costa. "Scheduling algorithms for high performance network switching on FPGAs: A survey." In *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 166-173, 2018

- Jiuxi Meng, Nadeen Gebara, Ho-Cheung Ng, Paolo Costa, and Wayne Luk. "Investigating the feasibility of FPGA-based network switches." In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160, pp. 218-226, 2019

- Philippos Papaphilippou, Jiuxi Meng, and Wayne Luk. "High-performance FPGA network switch architecture." In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 76-85. 2020

- Philippos Papaphilippou, Jiuxi Meng, Nadeen Gebara, and Wayne Luk. "Hipernetch: high-performance FPGA network switch." ACM Transactions on Reconfigurable Technology and Systems (TRETS) 15, no. 1 (2021): 1-31.

- Jiuxi Meng, Ce Guo, Nadeen Gebara, and Wayne Luk. "Fast and accurate training of ensemble models with FPGA-based switch." In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 81-84. 2020

- Jiuxi Meng, Zhiqiang Que, Ce Guo, and Wayne Luk. "CEARNA: FPGA-based Classifier with Ensemble Architecture for Nearest Neighbour Algorithm." Under review.

Glossary

FPGA Field Programmable Gate Array

ASIC Application-Specific Integrated Circuit

CPU Central Processing Unit

NIC Network Interface Card/Controller

GPU Graphics Processing Unit

LTL Lighweight Transport Layer

PCI-e Peripheral Component Interconnect Express

ECN Explicit Congestion Notification

SDN Software Defined Network

HDL Hardware Description Language

HLS High Level Synthesis

SSI Stacked Silicon Interconnect

HBM High Bandwidth Memory

ToR Top of Rack

IB Input Buffering

HOL Head of Line

VOQ Virtual Output Queue

OQ Output Queued

CIOB Combined Input and Output

CICB Combined Input and Cross-point

HC Hierarchical Crossbar

GCQ Grouped Crosspoint Queued

FMC FPGA Mezzanine Card

NoC Network on Chip

RTT Round Trip Time

BRAM Block Random Access Memory

RR Round-Robin

FF Flip-Flops

LUT Look-up Table

DSP Digital Signal Processing

SFP Small Factor Pluggable

QSFP Quad Small Factor Pluggable

RP Random Projection

MLP Multi-layer perceptron

SGD Stochastic Gradient Descent

GD Gradient Descent

TPU Tensor Processing Unit

DDR Double Data Rate

kNN k-nearest neighbour

LSH Locality Sensitive Hashing

Contents

Declarations **i**

Abstract **iii**

Acknowledgements **v**

Publications **ix**

Glossary **xi**

1 Introduction **1**

1.1 Motivation 2

1.2 Research challenges 4

1.3 Contributions and Thesis Organisations 5

2 Background **7**

2.1 Type of data centres 8

2.1.1 On-premise data centre 8

2.1.2 Hyperscale data centre (Cloud) 8

2.1.3 Edge data centre 9

2.2 Network switches in the data centre 9

2.2.1 Requirement of network switches 10

2.2.2 Existing switch fabrics designs 12

2.2.3 Common buffering techniques 14

2.2.4 Scheduling algorithms for crossbar fabric 17

2.3 Related Work on FPGA-based switches 17

2.4 Related work on applications 20

2.4.1 Dimension reduction 20

2.4.2 Ensemble Learning 21

2.4.3 Locality Sensitive Hashing (LSH) 22

2.4.4 kNN algorithm 23

2.4.5 Previous work on FPGA-based kNN design 23

2.4.6 kNN designs uses Hamming distance as the metric 24

2.5 Summary 25

3 FPGA-based network switches 26

3.1 Introduction 26

3.2 Architecture 29

3.3 Feasibility Study 33

3.3.1 Design setup 33

3.3.2 Performance analysis 35

3.3.3 FPGA Resource Utilisation 37

3.4 Analytical Model 40

3.4.1 Memory Resource Model 40

3.4.2 Port to port Latency model 42

3.4.3 Case study: Resource model use case 43

3.5 Sorting network-based high throughput switch design 45

3.5.1 Performance comparison with the crossbar-based designs 46

3.5.2 Resemblance to output-queued crossbar-based switches 47

3.5.3 Comparison to ASIC-based switches 52

3.6 Summary 54

4 Ensemble Model Training **56**

4.1 Introduction 56

4.2 System architecture 59

4.2.1 On-chip storage reduction for projection matrices 60

4.3 Theoretical model 61

4.4 Experiment setup 63

4.5 Performance estimation 64

4.5.1 System time modelling 66

4.5.2 Case study: Ad dataset training system speed-up 69

4.5.3 Tuning for optimised RP time 70

4.5.4 Time estimation for very large dimension size 71

4.6 Discussion 72

4.7 Summary 72

5 CERRNA **74**

5.1 Introduction 75

5.2 Proposed System Architecture 77

5.3 Software Modelling 81

5.4 Performance modelling 84

5.5 Dataset-specific case study 86

5.6 Result and Discussion 93

5.6.1 The relationship between speed-up and projected dimension 93

5.6.2 The effect of memory reduction on system scalability 94

5.7 Quantitative performance comparison 94

5.8 Summary 96

6 Conclusion **97**

6.1 Summary of Thesis Contributions 98

6.2 Future Work 99

6.3 Final Thoughts 101

Bibliography **101**

List of Tables

2.1 Comparison of high-end and low-end ASIC-based switches 12

3.1 Utilised custom packet format. 33

3.2 Design specification. 35

3.3 Buffer depth configurations (N denotes the number of ports). 35

3.4 Frequency and line rate scaling with switch size and data width. 36

3.5 Port-to-port latency scaling with switch size and data width. 37

3.6 Resource summary for Xilinx U280 Alveo 55

4.1 Estimated result. 66

4.2 RP time result with selected α value. 71

5.1 Details of the 4 types of RP matrices used in the experiment. 82

5.2 Experiment results 87

5.3 Sorter HLS implementation latency result. 91

5.4 Resource usage for Xilinx U280 Alveo. 91

5.5 Comparison with other designs. 95

List of Figures

1.1 Commonly used data centre structure (ToR: Top-of-Rack) 2

1.2 Current vs. future data centre structure with FPGA-based SmartNIC and Smart Switch 3

2.1 Common data centre topology 10

2.2 Existing switch fabrics 13

2.3 Buffering Techniques for Crossbar-based Switches 15

2.4 Hierarchical crossbar switch with a total radix of N and all sub-switches of radix p 17

2.5 The architecture of Hipernetch. Source: [1]. 18

3.1 BRAM implemented single queue for data width of 256 bit, larger data width results in higher BRAM usage as well as waste; a buffer depth of 64 provides zero packet loss as explained in section 3.3.1. 30

3.2 Shared memory architecture, similar to [2, 3] 31

3.3 Implemented device view of three architectures. The leaf cells highlighted in blue represent the memory resources used by buffers. The yellow cells represent the scheduler in the iSLIP design and the memory-based switch in the GCQ design. 37

3.4 Switch performance scaling results. 38

3.5 A base-10 log scale is used for the Y axis with the dotted horizontal line indicating the 100% resource usage of the target platform; For designs with usage over 100%, synthesis results are provided as they cannot be implemented on the FPGA; URAM is similar to BRAM but with larger unit block size and less flexibility, therefore shared memory module are implemented to first exhaust URAMs prior to using BRAMs to avoid wasting memory resources. 39

3.6 Memory model usage flow diagram. Note common parameters are parameters shared by both architectures and specific parameters are exclusive to GCQ only. 43

3.7 Abstract view of the design with 4x4 ports. As the detailed architecture and implementation are presented in Hipernetch [1], we thereby only show the abstract design and analyse its similarity to an output-queued design. 45

3.8 Comparison of 16×16 switches implementations on Xilinx Alveo U280. Source: [1]. 46

3.9 Average latency comparison of M/D/1 queueing model and simulation result. . . 50

3.10 Latency comparison with ASIC-based switches [4–11] (from left to right). 52

4.1 System flow 60

4.2 Dimensionality reduction data flow. 63

4.3 Simulation flow. 64

4.4 Time graph. Top: Forward mode (without RP). Bottom: Processing mode (with RP). 65

4.5 Experimental results showing the impact of the proposed system on training time and accuracy. 67

4.6 Time variable explained. 68

4.7 Column-wise matrix multiplication. 70

5.1 System architecture of FPGA-based system, building on top of an FPGA-based network switch. The Hamming distances are computed and sorted in the HS blocks (either on-FPGA or on-server), and T blocks prepare the top-k distances for aggregation. 78

5.2 Two aggregation methods for computing the class label based on the top-k result of each server. 82

5.3 Software simulation flow. The ensemble result is based on the aggregation type. 83

5.4 System architecture with time notation. 84

5.5 Detailed kNN flow for a single copy of the data in the ensemble, with Store phase (building the database) on the top and Query phase (computing kNN) on the bottom. 87

5.6 A simple circuit design for computing the Hamming distance between two binary vectors V1 and V2, which are extracted for the sign of the previous stage. 89

5.7 FPGA speed-up versus projected dimensionality. 90

Chapter 1

Introduction

A network switch is a device that operates at the data link layer of the OSI model, it connects all devices in the computer network through packet forwarding. With the rapid growth of technology (including the protocol, transceiver and semiconductor), network switches have evolved to operate in Terabit Ethernet [12], that is the speed of above 100 Gigabits per second. Due to the need for high performance, network switches are typically made with application-specific integrated circuits (ASIC).

Nevertheless, ASIC-based network switches with their fixed architecture allow no changes after taping out. This inflexibility slows down protocol/algorithm research which could potentially benefit from changing the underlying switch design. In a data centre where a large number of high-performance switches are used, the cost of updating the network switches is not trivial.

The thesis revolves around novel FPGA-based network switch design, as well as architectures around the FPGA-based network switches that provide benefits compared with the traditional ASIC switches. Apart from the design, performance models are also created to help application designers find the best parameters with the current FPGA board and to provide FPGA manufacturers with a reference for the next-generation FPGAs that focus on switch applications. Note that in this thesis, FPGA-based network switches, FPGA-based switches, and programmable switches are used interchangeably.

1.1 Motivation

Due to the increasing demand for both performance and variety of network services, new structures to overcome this problem are built. One good example is the smartNIC [13] designed by Microsoft as shown in Fig. 1.2a, this new structure involves using a bump-in-the-wire FPGA next to the NIC (Network Interface Card) as an accelerator, in contrast with a traditional NIC as shown in Fig. 1.1. The design provides high flexibility not only by allowing the FPGA to accelerate high-bandwidth flows but also by allowing the FPGA to assist data processing by the CPU via a PCIe connection. Moreover, FPGAs can generate their own packets which are independent of the CPU, in this way, FPGAs in the server can reach any other one in the data centre and together with the customised communication protocol LTL (Lightweight Transport Layer), idle FPGA resources of one server can be donated to the global pool for use by other servers. With the advancing of FPGA, the new structure has evolved from bump-in-the-wire to fully integrated design, further reducing the communication overhead, as shown in the middle of Fig. 1.2b. Such structure is best exemplified by Xilinx ALVEO U25 [14] and Intel N6000-PL and N5010 series [15].

Microsoft has shown that the implementation of smartNIC can bring such benefits to the data centre. The main motivation of this work is to explore whether similar benefits can be achieved by applying an FPGA-based structure to the network switch. The envisaged structure is illustrated in Fig. 1.2b.

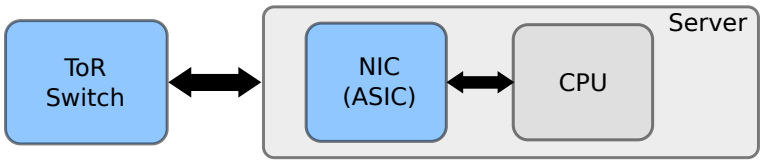
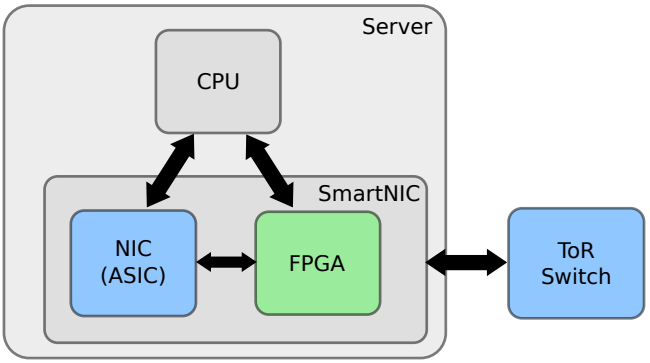


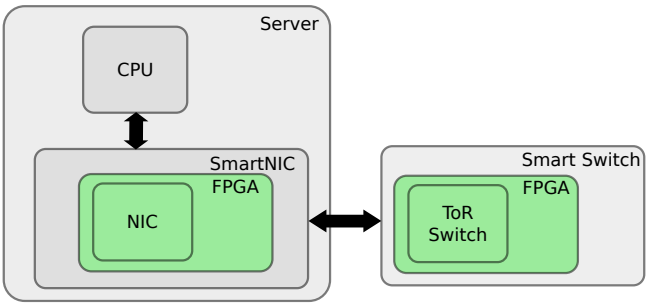
Figure 1.1: Commonly used data centre structure (ToR: Top-of-Rack)

Along with the main motivation, other benefits that motivate the FPGA-based smart switch are:

Cost saving. Nowadays network switches either have functions that are unusable to the customers or lack functions that customers need. Thus, there is always a mismatch between



(a) Microsoft cloud scale acceleration architecture with SmartNIC



(b) Envisaged future data centre structure

Figure 1.2: Current vs. future data centre structure with FPGA-based SmartNIC and Smart Switch

the need and the offer. To customise the product to suit customer requirements, the idea of building a customisable network switch is brought up. FPGAs show the way of building such devices, nowadays FPGAs are capable of handling 25Gbps (by the time of writing) Ethernet and the price of an FPGA that allows building a smart switch are much cheaper than before. This makes the future of a lower-cost network switch possible.

Ability to deploy custom functions/future-proof. Benefit from the programmability of FPGAs, it is possible to deploy customised network functions such as Explicit Congestion Notification Plus (ECN+) and pFabric [16], etc. to the smart switch. Moreover, these customised functions could also utilise the run-time reconfigurability to optimise their behaviour.

Offload application-specific tasks from the server. Because network switches connect multiple servers together, it is possible to offload tasks such as data aggregation [17] and Memcached-like caching to the switch. Additionally, with the extra computation resources packed within the FPGA, it is also possible to offload computational tasks such as dimension reduction, and data compression.

Future-proof. Although the Software Defined Network (SDN) switches with OpenFlow [18] interface provide a way of improving the network service at the software level with the fast-evolving algorithms and protocols, the hardware infrastructure cannot react fast enough to adapt to those new ideas due to the long manufacture cycle of ASIC. The mismatch of software and hardware update speed will not only slow down the development process but also leave more pressure on software developers. With FPGA-based switches, however, the programmable hardware can provide full support swiftly.

1.2 Research challenges

This section describes three challenges, C1, C2 and C3, that this research will address.

C1. Matching performance with ASIC-based switches. It is generally more difficult to achieve the same level of performance, in terms of aggregate bandwidth, with fully configurable logic such as FPGAs, which typically run at lower clock rates than Application Specific Integrated Circuits (ASICs). This is particularly challenging given that ASIC-based switches used in data centres today can support aggregate bandwidths of up to 12.8 Tbps, and even higher bandwidths of up to 51.2 Tbps are readily available [19]. This challenge would be addressed in Chapter 3.

C2. Resource limitation and application design. Due to the limited resources on the FPGA, implementing high-performance switches inevitably competes for hardware resources with other applications. It is challenging to design applications that have minimum impact on the switch design while providing competent performance. This challenge would be addressed in Chapter 4.

C3. Problem size and load distribution. For applications that can be offloaded to the switch, the load distribution ratio between the switch and the downstream server depends on the problem size and the available resources. Partially offloading an application introduces an extra level of complexity, thus it is challenging to alleviate or even eliminate this overhead. This challenge would be addressed in Chapter 4 and 5.

1.3 Contributions and Thesis Organisations

This section presents the first-authored publications, their relationship with each chapter and the related contributions to the thesis.

Chapter 2 provides the essential background for network switch design.

Chapter 3 describes the first contribution, which aims to address Challenge **C1**. It conducts a feasibility study that quantises the gap between the ASIC-based switches and the FPGA-based switches, this contribution is based on:

- Jiuxi Meng, Nadeen Gebara, Ho-Cheung Ng, Paolo Costa, and Wayne Luk."Investigating the feasibility of FPGA-based network switches." In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160, pp. 218-226, 2019

Chapter 4 presents the second contribution, which aims to address Challenge **C2**. It shows a novel FPGA-switch based architecture that enables the acceleration of ensemble learning. In addition, the usage of spare resources in computation is a potential benefit that FPGA-based switches can offer over ASIC-based switches. This contribution is based on:

- Jiuxi Meng, Ce Guo, Nadeen Gebara, and Wayne Luk."Fast and accurate training of ensemble models with FPGA-based switch." In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 81-84. 2020

Chapter 5 details the third contribution, which aims to address Challenge **C3**. It provides a novel architecture that allows a complete kNN application offloading the FPGA-based switch, this contribution is based on:

- Jiuxi Meng, Zhiqiang Que, Ce Guo, and Wayne Luk."CEARNA: FPGA-based Classifier with Ensemble Architecture for Nearest Neighbour Algorithm." Under review.

Chapter 6 concludes the studies we have carried out and the contribution we made. Future works are also discussed. In particular, we envisage the future hardware platform to better adapt to the FPGA-based switch concept, i.e. harden the switching fabric on the FPGA (making the switching fabric ASIC-based) to accelerate the forwarding speed as well as port-to-port latency.

Chapter 2

Background

This chapter starts with an overview of existing types of data centres, as well as different types of network switches used in the data centres. From this we identify three major requirements for network switches in the data centre, this serves as the guideline for designing FPGA-based network switches.

The investigation of the ASIC-based commodity network switches categorises the existing products, which allows a clearer comparison between the FPGA-based network switches and the ASIC-based counterparts.

Next, the related work on existing network switch design including switch fabric, buffering technique, packet scheduler and scheduling algorithm summarises the state-of-the-art works, this enlightened us on finding the design that suits the FPGA-based network switch.

Finally, since the flexibility of the FPGA-based switches enables customised applications on the network switches, and because the application running on the switch can also affect the switch design, we further provide backgrounds on applications that suit more on the FPGA-based network switch compared to ASIC-based commodity switches. These applications can benefit from the computational resources on the FPGA, which in turn reduces the workload for the downstream servers. Such applications are shown in Chapter 4 and Chapter 5

2.1 Type of data centres

Based on the purpose of the business, data centres can come in different shapes and sizes. Popular data centre types are:

2.1.1 On-premise data centre

An on-premise data centre is a physical facility that is owned and operated by an organisation to store and manage its IT infrastructure and data. It is located within the organisation's premises, typically within a secure building or data centre facility, and is managed by the organisation's IT staff.

On-premise data centres can be customised to meet the organisation's specific needs and requirements, and can provide greater control and security over sensitive data and applications.

Some of the advantages of an on-premise data centre include greater control and flexibility over the IT environment, the ability to customise and optimise the infrastructure to meet specific needs, and the ability to keep sensitive data and applications within the organisation's own premises. However, on-premise data centres also require significant upfront investment and ongoing maintenance and management costs, which can be a challenge for some organisations.

2.1.2 Hyperscale data centre (Cloud)

A hyperscale data centre is a large-scale facility that is designed to meet the high-performance computing needs of large enterprises and organisations. These data centres typically have a massive amount of resources, such as servers, storage systems, and networking equipment, and are capable of supporting massive workloads with high performance, reliability and scalability.

The term "hyperscale" refers to the ability of these data centres to scale up quickly and easily as demand grows. They are typically built using modular designs and can be expanded rapidly by adding more computing resources as needed. Tech companies can easily expand their operations

by leveraging the scalability of hyperscale data centres. Some examples of companies that operate hyperscale data centres include Amazon Web Services (AWS) [20], Microsoft Azure [21], Google Cloud [22], and Facebook.

2.1.3 Edge data centre

An edge data centre is a facility that is designed to bring computing and storage resources closer to the end-users and devices. These data centres are typically smaller than traditional data centres and are located closer to the edge of the network, and can be placed in a variety of locations [23].

The purpose of edge data centres is to reduce latency and improve the performance of applications and services that require low latency, such as streaming media, autonomous vehicles, and IoT devices.

2.2 Network switches in the data centre

A Network switch is a special hardware that connects devices on the network by forwarding messages (packets) from one device to another. In a data centre environment, hierarchical topology is used to support large-scale connections. A common data centre topology is shown in Fig. 2.1.

This topology divides the network into three layers, namely the core layer, the aggregation layer and the access layer. The switches placed in these layers correspond to core switches (layer 3), aggregation switches (layer 2) and access switches (layer 2). Due to the difference in responsibility of these switches, the performance requirement varies. For example, the core switches aggregate data from the aggregation layer, and the aggregation switches aggregate data from the access layer; the core switch may only need to connect to a few aggregation switches but with higher throughput, and the access switches may need to connect to many servers with moderate throughput. As a rule of thumb, performance requirement decreases

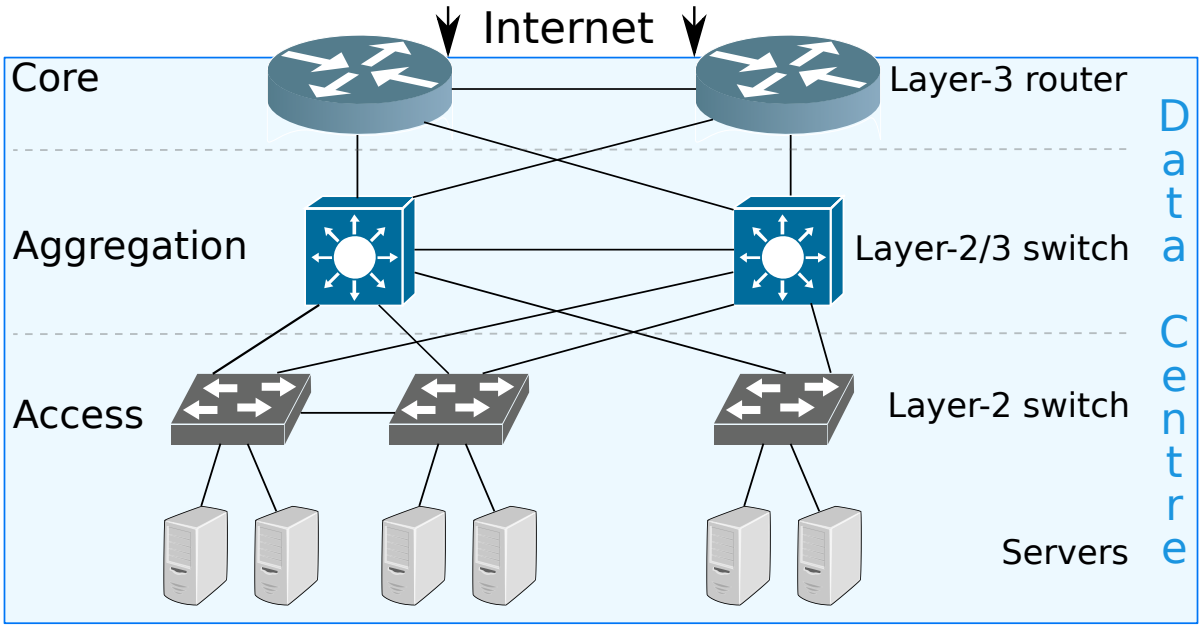


Figure 2.1: Common data centre topology

going down the topology. Due to the limited resources on the current FPGA chips, we put our focus on the layer 2 Top of Rack (ToR) switches, which are also called access layer switches when they are placed on the top of the server cabinet and are in charge of connecting the servers and the aggregation layer.

2.2.1 Requirement of network switches

The main requirements of network switches in a data centre include:

R1. High Performance. Network switches should be able to handle high volumes of traffic (high throughput) and provide low-latency connectivity between devices. A low-latency network switch should typically have a port-to-port latency of several hundred nanoseconds, the lower the better.

R2. Scalability. Network switches should be scalable to accommodate growing network demands. They should be able to support additional devices and traffic without significant performance degradation. The number of ports of a switch reflects this property. For a hyper-scale data centre that holds massive devices and heavy traffic load, the required number of ports of the top-of-rack (ToR) switch depends on the rack size. For an industrial standard 42U [24]

rack, if a 1U server with 2 network interfaces is used, this would require a switch with at least 82 ports to connect to a fully occupied rack. This number increases going up the topology as shown in Fig. 2.1. The number of ports needed for an edge data centre, which tends to have a smaller scale, can vary greatly depending on the specific use case.

R3. Compatibility. Network switches should be compatible with other network devices and protocols, including servers, storage systems, and networking technologies such as Ethernet, Fibre Channel, InfiniBand, etc.

Examples of ASIC-based switches ASIC-based switches come in different flavours. In this section, we briefly summarise commercial ASIC-based switches. The typical number of ports that can be found in a single switch are 8, 16, 24, 32, 40, 48, 64, 96, 128 and 256. In terms of the link capacity (port rate), typical values are $\leq$1Gbps, 10Gbps, 25Gbps, 40Gbps, 50Gbps, 100Gbps, 200Gbps and 400Gbps. In terms of port-to-port latency, it ranges from 90ns to 5000ns. The line between the low-end and high-end switch can be drawn by the total throughput of the switch. A higher throughput means more ports with a fixed link capacity or a higher link capacity with a fixed number of ports. Within the data centre, the physical size of the switch is usually measured by rack unit, which is the size of a single slot in a rack. Based on this, for a 1U (1 rack unit) design, the number of ports of the switch is usually 48 ports. The number of ports generally increases with the physical size of the switch. For high-end switches, the link capacity is at least 10Gbps and the number of ports goes as high as 2304 ports [25] through stacking multiple switches. Low-end switches on the other hand would either trade port numbers for link capacity or the reverse, with link capacity normally less than 10Gbps, and port numbers being as low as 8.

High-end switches are normally found in hyperscale data centres where traffic loads are massive, whereas low-end switches are normally found in smaller data centres such as edge data centres. Table 2.1 presents examples of both high-end and low-end switches and the requirements they meet.

A high-end switch is exemplified by the Cisco Nexus 9000 series aggregation layer switches. For the 9500 series modular switches, the modular design allows the switch to be configured from

Table 2.1: Comparison of high-end and low-end ASIC-based switches

Model	Type	Tier	# ports	Link capacity (Gbps)	Requirement met
Nexus 9500 Series [25]	Modular	high/low-end	<2304	10 - 400	R1, R2, R3
Nexus 9300-GX2 [26]	Non-modular	high-end	64	400	R1, R3
CBS350-48XT-4X [27]	Non-modular	low-end	48	10	R3

256 400-Gigabit Ethernet ports to 2304 1/10-Gigabit Ethernet ports, however, the latency can be as high as 5000ns. These switches provide the capability to use foundational layer 2/3 technologies, as well as modern technologies such as VXLAN, with a Border Gateway Protocol–Ethernet VPN (BGP-EVPN) control plane, Segment routing, Multiprotocol Label Switching (MPLS), and automation via NX-APIs [25]. For the non-modular switches, such as the Cisco Nexus 9300-GX2 Series [26], a maximum of 64 ports with 400Gbps link capacity is provided, supporting 51.2Tbps of bandwidth (the reported bandwidth is computed as *number of ports* $\times$ *link capacity* $\times$ 2). These switches support features such as VXLAN, IEEE 802.1ae MAC Security and intelligent buffer management, etc. A low-end switch is exemplified by the Cisco small business CBS350-48XT-4X (best in the CBS350 series) [27] with 48 10Gbps ports, supporting 1.04Tbps bandwidth. It supports features such as IPv6 support and Layer 3 traffic management.

2.2.2 Existing switch fabrics designs

Switch fabrics are in charge of making direct connections between any input and output port on a switch (for a network switch setting, we have the same number of input and output ports). The design of the switch fabric is tightly coupled with the traffic load and transistor technology. Fig. 2.2 summarises most categories of fabric designs that exist. In general, there are two categories, namely time division and space division.

Time division. If the traffic load is light or if the transistor speed is high, time division designs can be used. Time division essentially relies on the fabric to run N times faster than the data arrival speed, where N is the number of ports. In every cycle, if all input ports receive a slice

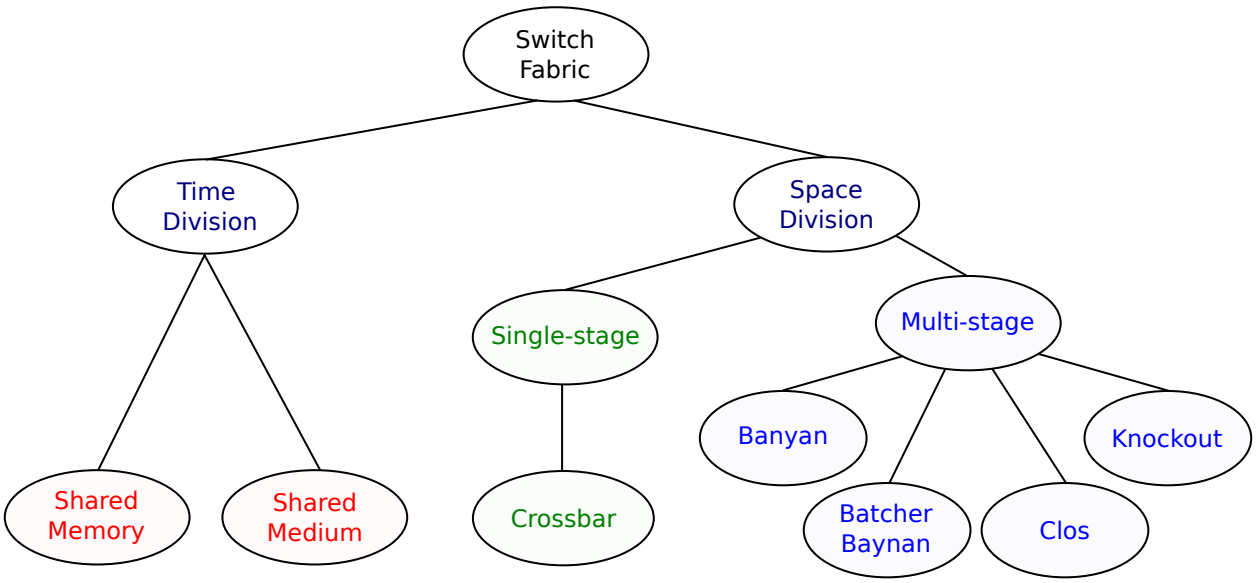


Figure 2.2: Existing switch fabrics

of data (N in total), then the fabric needs to move these N slices of data to their designated output ports. Time division can be implemented by either a shared bus or a shared memory design. Although its simplicity in design, time division scales poorly with the number of ports; this is due to the speed limit (operating frequency) of the transistor circuit.

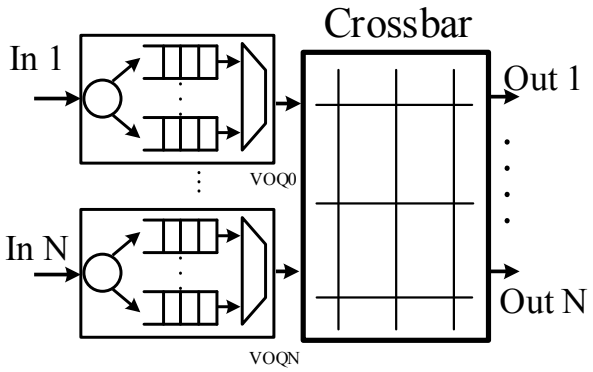
Space division. To avoid stressing the operating frequency of the circuit, space division fabrics allow multiple ports to transmit data at the same time; this is important for platforms such as FPGA where the operating frequency is low. Unlike time-division designs which only allow one pair of connections to be established at a time, space-division designs allow multiple connections to be made concurrently. Space division fabric can be realised with either single-stage designs such as the crossbar switch and the banyan switch [28], or multi-stage designs such as the Clos network [29], Batcher Banyan network [30] and the Knockout switch [31]. Furthermore, establishing optimised connections requires additional control of the fabric, i.e., the scheduler, and optimising the establishment requires well-designed scheduling algorithms. Single-stage design due to its simplicity requires less complicated scheduling algorithms compared to multi-stage designs. The multi-stage design takes multiple single-stage designs and forms a network of them. They are designed to relax a large number of interconnections (caused by either a high port number or high data width) in the single-stage design.

2.2.3 Common buffering techniques

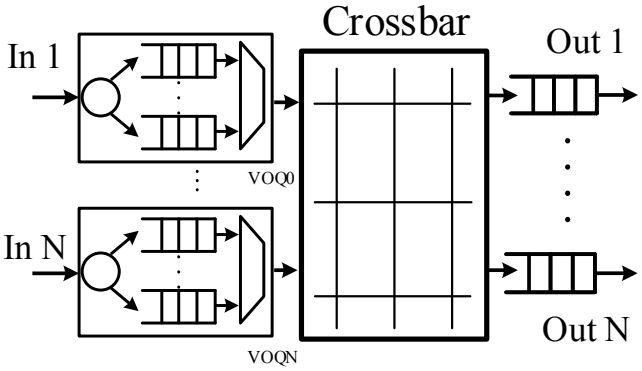
Buffering is crucial for resolving packet contention and reducing packet dropping rate. The choice of buffering is tightly coupled with the design of the scheduler as well as the scheduling algorithm. Various buffering techniques have been proposed for implementing high-performance switches, and these techniques differ both in terms of the location of the buffers and the scheduling algorithms used to eliminate resource conflicts that can arise in a switch. Fig. 2.3 presents three prominent buffering approaches. The strengths and weaknesses of each approach are described below.

(a) Input Buffering (IB): A limitation of placing one queue at each input port of a switch is Head of Line (HOL) blocking, which can significantly reduce the switch's maximum throughput [32]. This occurs when multiple input queues have packets destined for the same output. Those packets prevent subsequent packets destined to other output ports from traversing the switch, which ultimately leads to poor throughput.

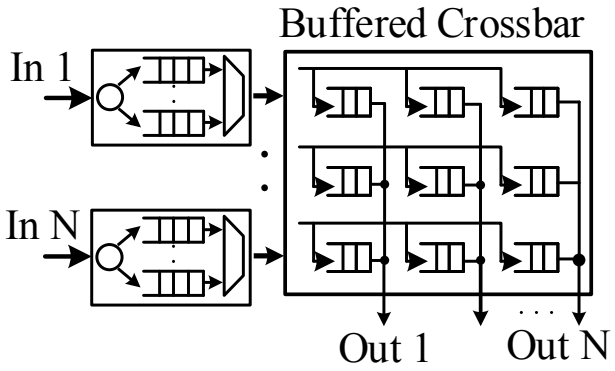
To avoid HOL blocking, Virtual Output Queues (VOQs) are used in input-buffered switches. The goal of VOQs is to have a queue for each output port at each input port such that packets no longer block each other when destined to different outputs as shown in Fig. 2.3a. Though VOQs eliminate HOL blocking, they introduce the need for simultaneous input and output conflict resolution. This means that the scheduler must not only determine input and output port matching, but also which VOQ to select from each input port. Therefore, for an N port switch, the scheduler must efficiently solve N^2 requests from VOQs, which greatly complicates the scheduling logic. This makes the choice of a scheduling algorithm that can achieve high throughput while still being simple enough to be implemented on an FPGA particularly challenging. Based on the outcome of our analysis of the complexity of implementing schedulers for input-buffered architectures on FPGAs [33], the scheduling algorithm iSLIP [34] is determined to be most suitable for an FPGA implementation. At a high level, iSLIP works as follows. At the beginning of each round, the input will send requests to the output grant arbiter based on the existence of the packet in the corresponding VOQs. Then the output grant arbiter will take the request vector (vector consists of all the requests from the input) and grant one request



(a) Input Buffering



(b) Combined Input and Output Buffering



(c) Combined Input and Cross-point Buffering

Figure 2.3: Buffering Techniques for Crossbar-based Switches

with a round-robin fashion starting from the port next to the previously granted port. Finally, when the input accept arbiter receives the granted request from the grant arbiter, it will accept the requests using the same round-robin method as the grant arbiter.

(b) Combined Input and Output Buffering (CIOB): Placing buffers at the output provides the best ideal switch performance and eliminates the need for scheduling by having the

buffers operating N times faster than the line-rate in an N-port switch. However, since the line-rate increases much faster than the memory bandwidth, output buffered switches scale poorly, and this is supported by available commercial products [35]. Combining input buffering and output buffering as shown in Fig. 2.3b allows the emulation of an output buffered switch using a broad class of packet scheduling algorithms while only requiring the scheduler, crossbar fabric, and output buffers to operate with a speed-up of $2 - (1/N)$ as opposed to N [36]. However, operating the scheduling algorithm with such a speed-up is still challenging for FPGAs especially when considering scaling to higher port counts (radix) and line rates. Therefore, we do not consider these architectures in our work.

(c) Combined Input and Cross-point Buffering (CICB): Buffers are located both at the inputs of the switch and are also made available for each input-output pair, as shown in Fig. 2.3c. This decouples the task of scheduling and allows the use of simpler scheduling algorithms at the switch's inputs and outputs, achieving a good performance in terms of both throughput and latency. However, the number of cross-point buffers scales quadratically with the number of ports and that makes cross-point buffering less attractive. To address this shortcoming, a Hierarchical Crossbar switch with CIOB was proposed by Kim [37] in order to retain the advantages of the CICB while making it more FPGA-friendly as we discuss next.

(d) Hierarchical Crossbar switch + CIOB (HC+CIOB): Kim et al. [37] proposes the use of a Hierarchical Crossbar architecture with CIOB to reduce the memory usage of cross-point buffered architectures as shown in Fig. 2.4. The author modified the CIOB architecture by replacing a $N \times N$ buffered crossbar with $(N/p)^2$ smaller $p \times p$ crossbars with CIOB. This modification reduces memory usage by 40% compared to a fully buffered crossbar. The idea of using HC switch is particularly beneficial for FPGAs: dividing the large radix (number of ports) switch into smaller ones allows for simpler scheduling algorithms and simplifies the overall design. Dai and Zhu built on this idea and proposed a Grouped Crosspoint Queued (GCQ) switch targeting FPGAs [38]. The idea of the GCQ switch is to replace the CIOB sub-switches with small-radix memory-based switches. Since BRAMs on the FPGA can run much faster than the fabric, the author chooses to speed up the block RAM 4 times faster than the fabric to emulate an 4×4 switch. We adopt a similar architecture in our analysis: we use

a round-robin scheduler for conflict resolution and investigate its hardware resource usage for higher radix switches.

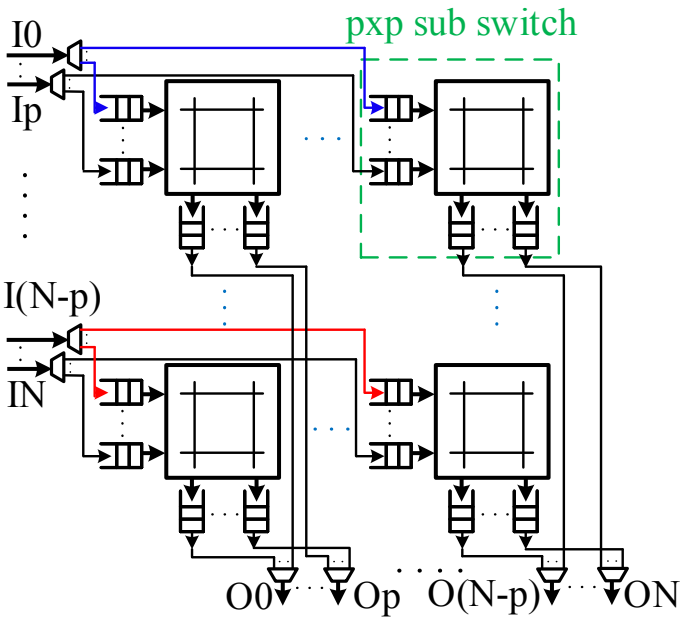


Figure 2.4: Hierarchical crossbar switch with a total radix of N and all sub-switches of radix p

2.2.4 Scheduling algorithms for crossbar fabric

Crossbar switches are used to connect multiple inputs to multiple outputs, allowing data to be routed between them. Scheduling algorithms are used to determine the order in which packets are transmitted through the crossbar switch. We focus on switching algorithms for crossbar fabric due to its simplicity and the fact that it is the basic component of most multi-stage designs. The FPGA-targeted scheduling algorithms are presented in the collaborated survey paper [39].

2.3 Related Work on FPGA-based switches

Yoshigoe et al. [40] implement a 24 port 10-Gbps switch with CIOB structure. However, multiple FPGAs are used due to the limitation of the past FPGA technology resulting in a high latency due to packet transmission across FPGAs. NetFPGA [41] is an open-source

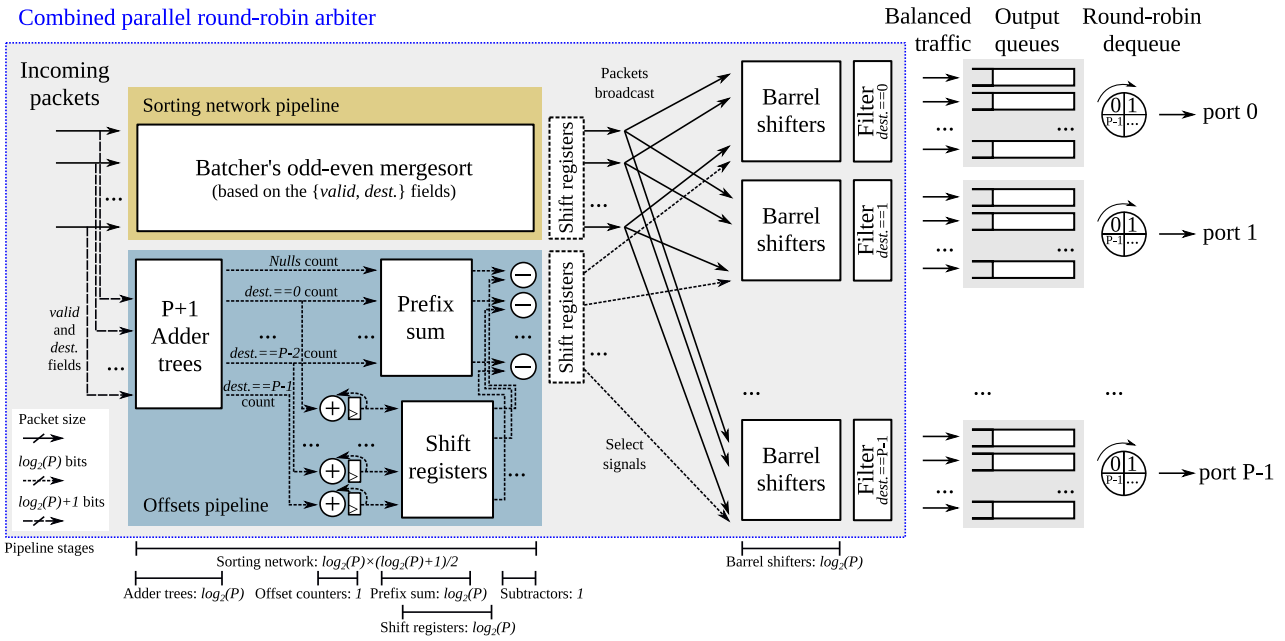


Figure 2.5: The architecture of Hipernetch. Source: [1].

hardware and software platform for network developers. The latest NetFPGA-SUME board provides four 10Gb/s ports and is capable of achieving 100Gb/s aggregate bandwidth by further utilising available FMC ports. It is further provided with a 4-port 10 Gb/s reference switch design that uses a CIOB fabric with a round-robin scheduler for conflict resolution. This results in a limited number of ports because the round-robin scheduler requires an internal data path that runs four times faster than the input interface to support all 4 ports.

More recently, there have been a few proposals to use hardened NoCs [42, 43] for FPGA switch implementations. These papers, however, only demonstrate the possibility of leveraging NoCs for implementing low latency and area-efficient network switches on the FPGAs through simulation since FPGAs with hardened NoCs are only recently made available [44].

Hipernetch [1] presents a high-performance FPGA-based network switch. It replaces the crossbar with a combined parallel round-robin arbiter, resulting a high frequency, low latency switch design.

The primary feature of the design is the "Combined parallel round-robin arbiter" illustrated in Fig. 2.5. This is a hardware optimisation of a switching scheme that involves having P parallel round-robin arbiters, one for each output port, to balance the FIFOs' occupancy. This

is possible because the output of the pipeline for each port does not rely on packets destined for other ports. Therefore, instead of using P sorting networks for P separate parallel round-robin arbiters, the proposal only employs one sorting network, hence the name.

The architecture merges two pipelines: the first for the "sorting network" and the second for the "offsets," which calculates the rotation each barrel shifter needs to perform. The sorting network sorts a batch of P incoming packets based on their destination and validity, making it easier to route them by grouping those with the same destination. This reduces the number of possible permutations needed before forwarding the packets to the output ports to only rotations. The "offsets" pipeline contains everything below the sorting network in Fig. 2.5, and mainly consists of adders, such as adder trees, that count the packets going to each port and a prefix sum that finds the starting position of each consecutive packet group using this information. The two pipelines have different lengths and use additional shift registers to synchronise their output.

The arbiter architecture is fully pipelined and processes up to P packets at line rate. All inserted packets are distributed into the output queues at its output. The pipeline logic is non-blocking, and any dropped packets can only occur at the queue level.

FPGAs in the network. The success of Microsoft's project catapult [45] and Amazon AWS [20] has initiated a trend of bringing FPGA to the cloud. An FPGA, due to its flexibility and reconfigurability, is a perfect candidate for balancing the growing cost of hardware update and the fast-evolving algorithms. The Azure SmartNIC [46], which utilises FPGA as a bump-in-the-wire accelerator next to the ASIC NIC, shows a reduction in latency and an increase in throughput for Azure services. FPGA-based network switches take a step further in bringing FPGA to the data centre. Because the network switch is located at a higher level than the NIC, it is supposed to bring similar benefits to the network applications with the NIC whilst reducing the response latency. There are two major types of studies on FPGA-based network switches:

1. The infrastructure design targeting FPGAs such as [2, 47] focusing on designing high throughput and low latency switch fabric on the FPGA.

2. Applications utilising FPGA-based switches such as [48] focus on reducing the latency of distributed training by moving the aggregation step into the switch. [49] also conducts a series of experiments analysing the impact of FPGA accelerators based on their relative distance in an IoT network.

2.4 Related work on applications

2.4.1 Dimension reduction

Random projection is a technique used in high-dimensional data analysis and machine learning. The concept behind random projection is based on the Johnson-Lindenstrauss lemma, which was introduced by William B. Johnson and Joram Lindenstrauss in 1984 [50]. The lemma states that a high-dimensional dataset can be projected onto a lower-dimensional space while preserving the pairwise distances between the data points.

The idea of random projection is simple. Consider a high-dimensional dataset consisting of n data points, each of dimension m. To project this dataset onto a lower-dimensional space, we randomly select a set of k vectors, each of dimension m, and project each data point onto these k vectors. The resulting projection reduces the dimensionality of the dataset from m to k, which is typically much smaller than m.

The key advantage of random projection is that it allows us to reduce the dimensionality of a dataset while preserving the pairwise distances between the data points. This makes it an efficient technique for analysing high-dimensional datasets, as it reduces the computational complexity of many algorithms while preserving the accuracy of the results.

The most popular random projection methods are Gaussian random projection and Sparse random projection [51, 52].

Random projection on its own can perform both input feature manipulation and randomness injection to the training dataset, which suits well the purpose of generating ensemble classifiers.

Canning [53] investigates the accuracy of the ensemble method with random projection in a theoretical way, whilst our work focuses on a practical scenario where reconfigurable logic is used to deploy the concept at the server clusters level.

Fox. et al [54] proposes the first FPGA-based RP method to make training scalable on FPGAs. As their work involves training within FPGAs, they need to restrict the computational resources allocated for RP to make room for training. As a result, they resort to using RP matrices with higher sparsity to simplify the hardware requirements. This increase in sparsity introduces higher error for the projected subspace. Training on FPGA also limits their design to small feature datasets, whereas the effects of random projection are more significant on high feature datasets. Unlike their design, our work devotes all the idle computational resources to FPGA for random projection, allowing us to explore RP using Li's method [52] without extra error injection.

Nazemi. et al [55] introduces an algorithm that allows the deployment of different dimensionality reduction models in hardware, including random projection, principle component analysis (PCA), independent component analysis (ICA) and Equivariant Adaptive Separation via Independence (EASI). The design computes all dimensions concurrently, as a result, it only achieves dimension reduction for input with 32 features.

2.4.2 Ensemble Learning

Ensemble learning is a technique in machine learning where multiple models are trained and combined to improve the accuracy and robustness of the final prediction. The concept behind ensemble learning is based on the idea that a group of weak learners can combine their predictions to create a stronger learner.

Ensemble learning involves training multiple models using different algorithms, different training data, or different hyperparameters. The models are then combined to create a final prediction. Popular techniques for combining models in ensemble learning include:

- Bagging (Bootstrap Aggregating) [56]: In this technique, multiple models are trained

using different random samples of the training data. The final prediction is then based on the average or majority vote of the predictions made by each model.

- Boosting [57]: In this technique, multiple models are trained sequentially, with each model focused on learning from the errors made by the previous model. The final prediction is then based on a weighted combination of the predictions made by each model.
- Stacking [58]: In this technique, multiple models are trained and their predictions are used as input features to a meta-model that makes the final prediction.

Ensemble learning can improve the accuracy and robustness of the final prediction by reducing the variance and bias of the models. By combining the predictions of multiple models, ensemble learning can reduce the risk of overfitting the training data and improve the generalisation performance of the model. In our work, different from the techniques mentioned above, we used a simplified version of [53].

Related work on ensemble learning. A survey [59] on the ensemble method reviews several general-purpose methods for constructing ensembles, including Bayesian voting, manipulating training examples, input features as well as output targets and injecting randomness. It also points out that the ensemble method can achieve higher accuracy by combining the results of multiple less accurate models. Lou. et al [60] proposed an FPGA-based flexible computing architecture (through partial reconfiguration) that achieves a faster anomaly detection (compared against a CPU implementation) through an ensemble of individual anomaly detectors.

2.4.3 Locality Sensitive Hashing (LSH)

The hyperplane technique described in [61] gives:

$$Pr[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi} \tag{2.1}$$

where $\theta(\vec{u}, \vec{v})$ measures the angle between vectors $\vec{u}$ and $\vec{v}$. Since $cos(\theta)$ can be estimated by θ, this similarity on the right-hand side can be treated as cosine distance. Furthermore, Indyk

and Motwani [62] show that the similarity between two vectors $\vec{u}$ and $\vec{v}$ can be adapted to measure the Hamming distance between them.

2.4.4 kNN algorithm

Given a query data that has the same dimensionality as the training dataset. kNN computes the distance between the query data with samples in the training dataset. The distance is then sorted to obtain the k closest training samples. Based on the label of these top-k samples, the label of the query data can be derived. When computing the distance, there are a variety of metrics available. For example, Euclidean, Manhattan, Cosine and Hamming distance, etc. For vectors X and Y with M dimensions each, the Euclidean distance is computed as:

$$D_{Euc}(X,Y) = \sqrt{\sum_{i=1}^{M}(X_i - Y_i)^2} \tag{2.2}$$

The Cosine distance is computed as:

$$D_{Cos}(X,Y) = 1 - cos(\theta) = \frac{\sum_{i=1}^{M} X_i Y_i}{\sqrt{\sum_{i=1}^{M} X_i^2}\sqrt{\sum_{i=1}^{M} Y_i^2}} \tag{2.3}$$

where $cos(\theta)$ is the cosine similarity between vectors X and Y. The Hamming distance is computed as:

$$D_{Ham}(A,B) = \sum_{i=1}^{M}(A_i \oplus B_i) \tag{2.4}$$

where A and B are binary vectors of dimension M.

2.4.5 Previous work on FPGA-based kNN design

Most previous research on FPGA-based k-nearest neighbours focuses on designing specialised hardware to efficiently run kNN. Lu [63] implemented an HLS-based kNN accelerator that optimises memory access with multiple DDR and HBM memory banks, the design was tested on data with dimensions up to 128, using single-precision floating-point type for data representa-

tion. The high memory bandwidth comes at the cost of routing resources, especially in LUTs. This high resource usage does not suit the current FPGA switch based application design scenario. In addition, the high-precision data representation used in this work limits the data size that can be stored onboard.

An [64] proposed a highly parallelled FPGA-based coprocessor for computing the nearest Euclidean distance searching. This design also parallels memory access to gain high memory bandwidth and is tested on feature vectors with dimensions up to 256. Pu [65] proposed an OpenCL-based high-speed kNN accelerator utilising a modified parallel bubble sort, their design was tested on data with dimensions equal to 64, also using a floating-point representation. Hussain [66] implemented an ensemble kNN classifier using a systolic array and leveraging partial configuration, however, using the Euclidean distance metric limits the dimension of the input data even with the 16-bit datatype, and as a result, only small datasets are supported. Song [67] implemented an HLS-based memory-efficient kNN leveraging PCA filtering and low-precision data representation. Their design uses at least 4 bits to represent each feature, however, it was only tested on a single dataset. Viera [68] implemented a scalable kNN classifier that exploits various levels of parallelism. All aforementioned designs operate on datasets with relatively smaller dimensions and use relatively higher precision data types compared with our design.

2.4.6 kNN designs uses Hamming distance as the metric

Songhori [69] introduced an efficient method for privacy-preserving k-nearest neighbours search. Hamming distance is used in their sequential kNN circuit. The design works on data in the Hamming space and no accuracy is presented, hence parallel to our application. An [70] implemented an associative-memory based online learning method that uses fully-parallel associative memory to reduce query time for nearest neighbour finding. However, the data is pre-processed heavily before entering the FPGA. Besides, the tested dataset is a binary dataset which is sufficient to use Hamming distance without losing accuracy. Lee [71] proposed a similarity search design on an Automata Processor (AP) that uses Hamming encoding to solve the sorting phase of kNN in linear time. This ASIC design on the AP platform achieves great speed-up. How-

ever, this design assumes dataset vectors are processed offline, meaning it only needs to work on datasets with small dimensions.

2.5 Summary

This chapter first introduces different types of data centres and different types of network switches in the data centre, followed by a review of existing switch designs including switch fabric, buffering technique, scheduler and scheduling algorithms. Previous works on FPGA-based switches show that challenges remain in designing FPGA-based network switches, indicated by the performance gap between the FPGA-based switch and ASIC-based switch. In chapter 3, several FPGA-based switch design is presented, aiming to tackle challenge **C1** and making it more closely fulfil the requirements of network switches.

This chapter further reviews related applications that are adopted and applied to the FPGA-based switch system architecture. In chapter 4 and 5, we demonstrate the ability of our system architecture to accelerate machine learning related tasks. And how they address challenges **C2** and **C3**.

Chapter 3

FPGA-based network switches

FPGAs are being increasingly used on network interface cards (NICs) as offload units to accelerate packet processing tasks. The rationale is that by customising the NIC logic it is possible to achieve higher performance for the most critical tasks while eliminating unnecessary logic, thus improving overall efficiency.

In this chapter, we aim to investigate if similar benefits can also be extended to network switches. We compare different switch architectures and analyse their suitability for an FPGA implementation. We discuss several optimisation techniques to overcome the challenges of limited FPGA resources and assess the scalability of our designs up to 10, 25, and 50 Gb/s throughput per port, which are commonly found in data centre switches [72]. Finally, we make a comparison with a range of ASIC-based switches and discuss the gap between the FPGA-based product and ASIC-based ones with respect to the port-to-port latency and number of ports.

3.1 Introduction

FPGAs are often used in today's data centres to support compute-intensive workloads such as data analytics and machine learning as well as to accelerate network processing functions at

the end hosts. For example, Microsoft has shown that using FPGAs as a bump-in-the-wire between the CPU and the NIC can accelerate Bing search, machine learning inference tasks, and packet processing [13, 73, 74]. In this chapter, we explore whether FPGAs can be used to implement and customise network switches.

While modern network switches support some limited form of programmability, e.g. [75], most of their operations are implemented as fixed logic. This poses two main challenges. First, data centre operators can not arbitrarily modify this logic and implement custom network functions. For example, if a new packet scheduling policy is to be implemented, the only option is to engage with the switch manufacturer (if at all possible) and wait for the next chip tape out, which typically requires 18 months or longer. Second, today's cloud providers may not use all the functionality offered by network switches. This leads to a waste of silicon area as well and potentially higher costs and power consumption than necessary. An FPGA-based switch in contrast provides more efficient solutions as it allows the flexibility of only implementing the switch functions needed (including custom ones).

However, it remains unclear whether an FPGA-based switch can match the performance of ASIC-based switches and meet the requirements as mentioned in section 2.2.1. If we just consider the total transceiver bandwidth provided by FPGAs today, the gap between FPGAs and ASIC-based switches is shrinking. The latest generation Intel Stratix 10 TX FPGA has an aggregate transceiver bandwidth of 8Tb/s [76], which is almost two-thirds of the bandwidth provided by top-end data centre switches [77] released at the same time (12.8 Tb/s), providing evidence for meeting the **High Performance (R1)** requirement. However, this only reflects the *IO* capability and does not indicate whether the FPGA has sufficient memory and compute capabilities to switch packets at such rates, leaving the **Scalability (R2)** requirement unknown. FPGA as a flexible hardware platform automatically meets the requirement **Compatibility (R3)**.

This chapter makes a first step towards determining this by focusing on the core functionality required by any switch, i.e., the ability to forward packets from its input ports to its output ports. Such functionality can be provided by either a space-division fabric such as crossbar

(single stage) and crossover (multi-stage) along with some buffering and appropriate scheduling algorithms, or a time-division based fabric such as shared memory that operates at a rate proportional to the number of switch ports. In this chapter, we explore the trade-offs provided by different architectures for implementing the building blocks (buffers and scheduling logic) of such switches and reveal the challenges in meeting **R2** and **R3**. Such challenges on FPGAs include:

1. Limited on-chip memory and routing resources on the FPGA limit the size of the switch
2. FPGA usually runs at a much lower frequency than the ASIC design, thus it is challenging to match their performance

We make the following contributions:

1. The development of an efficient buffer-sharing technique that minimises memory resource usage on FPGAs for input buffered crossbar switch (Section 3.2). The use of the shared memory design reduces the usage of on-chip memory blocks, as a consequence reduces the amount of wiring and routing logic for these memory blocks as indicated in Fig. 3.3. This tackles the first challenge and improves the **Scalability**.
2. A feasibility study that shows the performance and resource utilisation trade-offs of considered switch architectures (Section 3.3). To address the second challenge, with the limited operating frequency, we focus on experimenting with different data width to gain the same line rate as the ASIC design.
3. An analytical model to predict the performance and resource requirements of the implemented architectures at various scales (Section 3.4).

To investigate the trade-offs of implementing an FPGA-based single-chip switch through design space exploration, we first identify designs best suited for FPGAs and then implement them while considering various optimisation techniques. Our work therefore provides a straightforward view of the pros and cons of the FPGA-based switches design. We found that switch

architectures that make use of input buffering with the iSLIP scheduling algorithms and the hierarchical buffering approach that leverages GCQ with a simpler scheduling algorithm are most amenable for FPGA implementation due to their scalability and switching performance. Therefore, this chapter focuses on their implementations and the corresponding investigation to demonstrate the feasibility of these two approaches. In particular, we study these approaches from a single-chip perspective and provide a comprehensive overview of their advantages and drawbacks. To the best of our knowledge, this is the first work that provides a direct comparison of these two architectures in the context of FPGAs. Additionally, another collaborative work (Hipernetch) on a high throughput switch is presented and compared with the aforementioned approaches in section 3.5.

3.2 Architecture

In this work, we restrict our attention to architectures that can fit within a single FPGA chip for the following reasons. First, multi-chip implementations introduce a significant latency overhead due to the long round-trip time (RTT). According to Shen, the measured average latency between FPGAs using the Interlaken IP was approximately 170 ns [78] which is roughly the same as the port-to-port latency of our entire single-chip implementation. Furthermore, multi-FPGA implementations are far more complicated than a simple, single-chip design. Because a single-chip design will not only reduce the latency introduced by the inter-FPGA connection but will also simplify the design. As the long RTT between the line card and crossbar will affect the switch performance. Shen [78] measured the average latency between FPGAs using the Iterlaken IP, which is 170ns and this is approximately the same level as the port-to-port latency of our implementation. This latency does not improve with the latest 2nd Gen Interlaken, according to the IP guide [79]; hence the multiple FPGA-based switch design will be out of scope.

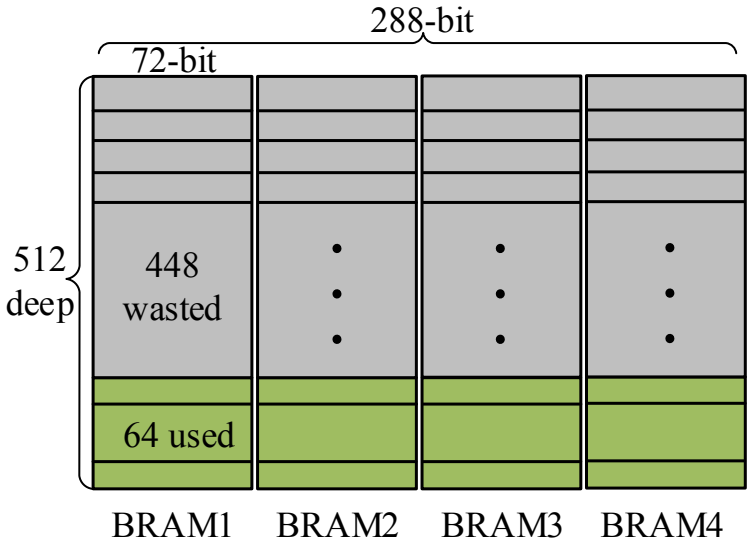


Figure 3.1: BRAM implemented single queue for data width of 256 bit, larger data width results in higher BRAM usage as well as waste; a buffer depth of 64 provides zero packet loss as explained in section 3.3.1.

Fabric

A crossbar-based switch consists primarily of buffers, a scheduler, and the crossbar fabric itself. Crossbar-based switches are non-blocking since any input port can be connected to any output port. In addition, the behaviour of the crossbar can be easily emulated with a group of multiplexers making them an attractive fabric choice for FPGA-based switches.

Buffering

Since low memory access latency is crucial to any switch design, fast on-chip memory resources must be used for buffering instead of the off-chip memory. Naively using BRAM FIFOs on FPGA for VOQs implementation, however, would limit the design's scalability and would make use of the memory resource available on-chip inefficiently.

Example: The Xilinx UltraScale+ XCVU9P consists of 2160 on-chip BRAMs. Assume that a line rate of 10Gb/s can be supported with a data width of 256 bits and a queue depth of 64 operating at a frequency of 40MHz, which is the same setup as shown in section 3.3.1. Since the 36 Kb BRAMs on the FPGA can be configured to have a maximum width of 72

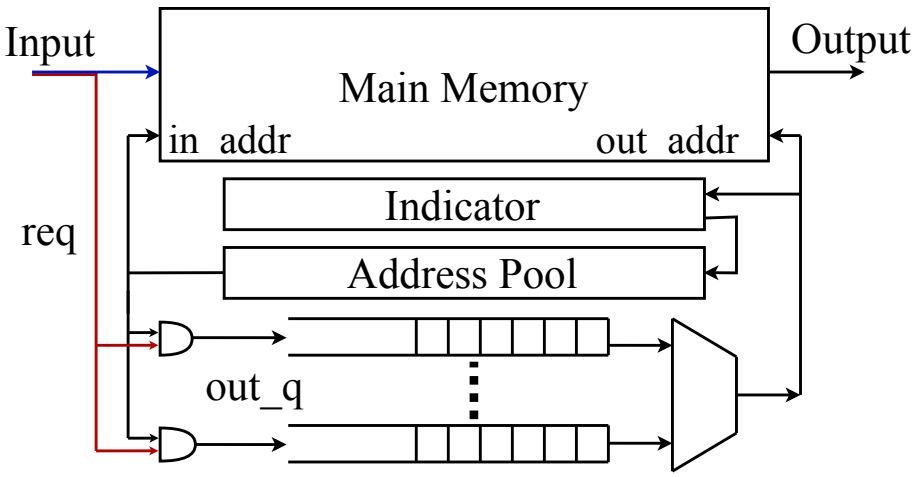


Figure 3.2: Shared memory architecture, similar to [2, 3]

bits, a total of four 36 Kb BRAMs would be required to write 256 bits in parallel. Since the number of VOQs required for an NxN switch is N^2, the following constraint must be satisfied: $4 \times N^2 \leq 2160$. This in turn limits the maximum number of ports that can be supported to 23. Additionally, most of the BRAM resources are also wasted as shown in Fig. 3.1. Utilising the distributed RAMs instead of the BRAMs relaxes the constraint to $256 \times 64 \times N^2 \leq 36.1$ Mb [80]. Nonetheless, the maximum port number is still limited to 46. While UltraRAMs present larger storage capacities, they are as limited as BRAMs in flexibility and utilising them for implementing VOQs would similarly be inefficient.

To efficiently use the on-chip FPGA memory resources, the shared memory architecture used by the IBM Prizma switch [3] is adopted to replace large VOQs. This architecture enables the flexible sharing of a single BRAM/UltraRAM-based memory across VOQs belonging to the same input port. Another advantage of this architecture is its high tolerance to bursty traffic, as the busy queues will be allocated with more buffer space in the shared memory space. The shared memory structure has similarly been used in the work of Dai and Zhu [38] when implementing a memory-based switch, and is shown in Fig. 3.2. Different from [38], which uses the shared memory as the internal switch, we use the shared memory to replace the virtual output queue.

The shared memory architecture is implemented using a dual-port RAM that supports concurrent read and write operations. The address pool is implemented as a simple FIFO populated

with available memory addresses. When a packet arrives at a shared memory block, the destination field is fed to the output pointer queues whereas the data get stored in the shared memory with the address provided indicated by the address pool. The indicator block shown in Fig. 3.2 is designed to handle multi-cast packets that are destined to multiple output ports. Instead of storing the same data multiple times in the shared memory, only one copy of the multi-cast data is kept. This minimises storage overhead since the number of bits required to represent the request is much smaller than the data itself. The indicator stores a bit-map indicating the output ports requested by a packet when it is first written into memory. The bit-map entry representing a certain output is cleared when the header is popped out of the corresponding output port, and the memory address is freed and returned to the address pool only after the bit-map value is 0, which indicates that all the multi-cast requests have been handled.

Although utilising an asymmetric dual port memory for implementing the indicator might appear optimal as it allows the modification of individual bits within the bit-map with a single clock cycle delay, such an approach would limit the scale of the shared memory. This is because the read and write width ratio of the dual port memory provided by Xilinx [81] memory is limited to 32. This in turn would restrict the maximum number of queues that can be shared. To remove this limit, we choose to implement the indicator block using a symmetric dual port memory at the expense of a cycle overhead required for updating the entire bitmap on each access. In other words, the symmetric memory now requires the entire old bit-map content to be read before writing back the new value thereby introducing an additional read delay.

Schedulers

Schedulers are among the core components of crossbar-based switches as they are needed to resolve potential port conflicts that could arise in a switch. We consider two schedulers in our work: **1)** a centralised iSLIP scheduler that implements the iterative maximal size matching scheduling algorithm proposed in [82] and guarantees 100% throughput for uniform Bernoulli traffic. **2)** a round-robin (RR) scheduler that can provide additional implementation simplicity

Table 3.1: Utilised custom packet format.

Field	Source	Data	Destination
Width	$\log_2 N$	$W - \log_2 N - N$	N

over iSLIP in architectures where complex centralised scheduling is not required.

3.3 Feasibility Study

To investigate the scalability of FPGA-based crossbar switches, two architectures are considered. The first switch architecture, ***SMiSLIP***, uses shared memory input-buffering with a centralised iSLIP scheduler and is chosen to represent the broad class of input-buffered switches with centralised schedulers. The second design, ***GCQ***, is the switch architecture proposed by Dai and Zhu in [38], and is chosen to be representative of CICB and HC+CIOB architectures since the CICB architecture is an extreme case of the HC switch with S=1 as explained in Section 2.2.3. We also implemented a simple crossbar switch architecture with an iSLIP scheduler and with no buffer sharing to serve as a baseline. We specifically focus on simple scheduling algorithms since they are more likely to scale to higher line rates and the number of ports while achieving timing closure on FPGAs.

In the rest of this section, we first introduce our design setup and then discuss the results of our study. Specifically, we present the trade-offs of the various architectures in terms of their maximum achievable line rate and the number of ports (aggregate bandwidth), port-to-port latency, and FPGA resource utilisation based on the simulation results obtained.

3.3.1 Design setup

Design Flow & Tools

Vivado Design Suite 2017.2 is used for hardware implementation, and our results are obtained targeting Virtex Ultrascale+ XCVU9P FPGA with `out_of_context` flag to prevent IO in-

sertion. Our results do not include any transceiver logic and are limited to the switching architectures themselves.

Packet Format

All implementations assume uniform traffic and a fixed packet size representing a single transfer unit, namely the cell. Though this approach increases the control resource overhead, it allows for a simpler flow control logic. We simply use buffer occupancy flags (full or empty) as a notifier to the sender. A customised packet format summarised in Table 3.1 is chosen since our primary goal is investigating the basic switching functionality. The radix of the switch is represented by N, and W is the total packet width. Each bit in the destination field represents the request to the corresponding output port. $Log_2(N)$ bits are used to represent the input port, and are only included to provide a more realistic packet format even though they are not necessarily required for correct switch operation.

Design specification

We consider switches with the three line rates shown in Table 3.2. The reason why the operating frequency is fixed at 40MHz is twofold: First, 40MHz is the boundary operating frequency for the iSLIP scheduler to support 128 inputs based on our experimental results. Second, the GCQ switch requires four times speed-up in the design, using 40MHz not simultaneously eases achieving timing and provides a fair comparison with the previous work.

Buffer Depth

For zero packet loss, a VOQ buffer depth of 64 is used for the iSLIP-based switch. The GCQ switch requires a smaller VOQ depth for zero packet loss and a VOQ depth of 16 is sufficient as presented in [38]. The utilised buffer depths for the various switching architectures considered are further summarised in Table 3.3.

Table 3.2: Design specification.

Platform	Operating frequency	Line rate	Data width
Virtex UltraScale+ xcvu9p	40MHz	10Gb/s	256
	40MHz	25Gb/s	640
	40MHZ	50Gb/s	1280

Table 3.3: Buffer depth configurations (N denotes the number of ports).

	iSLIP	SMiSLIP	GCQ
VOQ depth	64	n/a	16
Shared memory depth	n/a	64N	64N
Memory pool depth	n/a	64N	64N
Output address pointer queue depth	n/a	64N	64N

3.3.2 Performance analysis

Design scalability

To get a better view of the scalability of the crossbar-based network switch, designs targeting 10Gb/s, 25Gb/s and 50 Gb/s (Data Width 256-bit, 640-bit and 1280-bit) were implemented for all three designs. Without memory sharing the naive iSLIP can only achieve a maximum of 32 ports with 10Gb/s and a maximum of 8 ports with 50Gb/s. The SMiSLIP on the other hand can scale to 64 ports with 10Gb/s and 32 ports with 50Gb/s.

The maximum frequency and link capacity (maximum running frequency $\times$ data width) that could be supported for each data width and the port count is determined and provided in Table 3.4. Note that GCQ consists of two clock domains, with one being 4 times faster than the other. The slower clock frequency is displayed in Table 3.4 because this clock determines the link speed of the switch.

As can be seen from Fig. 3.4a, for switches with 32 or fewer ports, SMiSLIP can support a higher link capacity than GCQ. This is because no speed-up is required for SMiSLIP so that the entire switch can run at the highest frequency. However, due to the centralised architecture of SMiSLIP, the maximum supported frequency decreases more rapidly than that of the GCQ

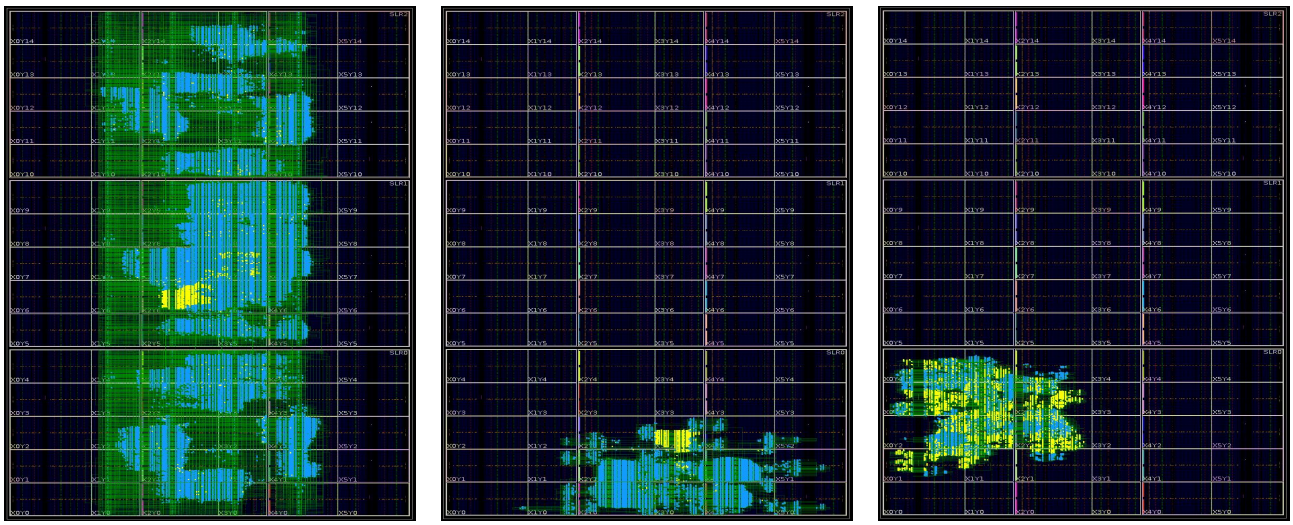
Table 3.4: Frequency and line rate scaling with switch size and data width.

	Maximum Frequency (MHz)						Link Capacity (Gbps)	
Data Width	256 bit		640 bit		1280 bit			
# Ports	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ
8	228.68	55.49	206.31	54.14	206.65	48.31	264.52	61.83
16	201.49	54.29	192.53	52.69	183.79	43.67	235.25	55.88
32	135.65	51.28	109.89	50.64	56.72	46.42	70.33	59.42
64	52.46	49.29	-	48.56	-	-	13.43	31.08

switch. This is because the iSLIP scheduler contains three stages in one scheduling iteration and the input size of the scheduler is proportional to the port number squared. Larger input size leads to a higher combinational logic delay and eventually reduces the maximum running frequency. While inserting registers could help increase the operating frequency, this would also introduce cycle overhead to the scheduler and as an iterative scheduler, this overhead would be multiplied by the number of iterations, which makes its overall efficiency less clear. We leave a more thorough exploration of this possible optimisation to future work. GCQ on the other hand has a stable performance due to its hierarchical structure which makes it more suitable for higher radix switch designs.

Average port-to-port latency

Table 3.5 reports the port-to-port latency obtained for each implementable configuration. The latency of all the designs increases with both the port number and the data width. Fig. 3.4b contrasts the average port-to-port latency of the SMiSLIP and GCQ switch designs for different port speeds. To provide a fair comparison against ASIC-based switches, the latency of the transceivers is further accounted for in the result. According to [83], the latency of the transceivers ranges from 13.65ns to 96.75ns for 10 Gb/s line rate resulting in an average latency of 55.2ns. The shaded area represents the latency boundary of ASIC-based switches [84–86]. Overall, both design performs well in latency compared with the ASIC-based products, and SMiSLIP has much lower latency compared with GCQ. This is because GCQ switches require clock domain crossing at both input and output interfaces as well as the use of pipeline registers to break the long wires that inter-connect the memory-based switches.



(a) iSLIP 10G 16-port **(b)** SMiSLIP 10G 16-port **(c)** GCQ 10G 16-port

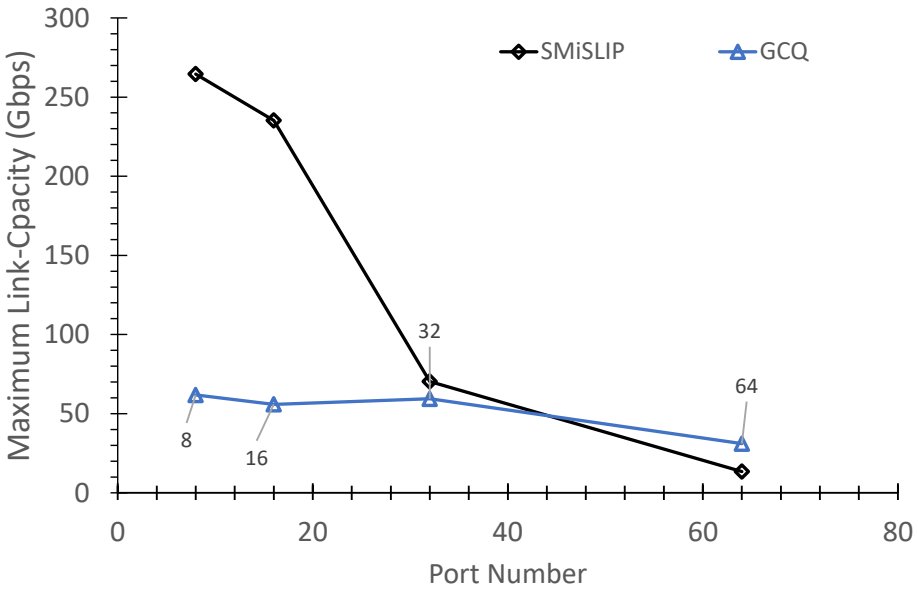
Figure 3.3: Implemented device view of three architectures. The leaf cells highlighted in blue represent the memory resources used by buffers. The yellow cells represent the scheduler in the iSLIP design and the memory-based switch in the GCQ design.

Table 3.5: Port-to-port latency scaling with switch size and data width.

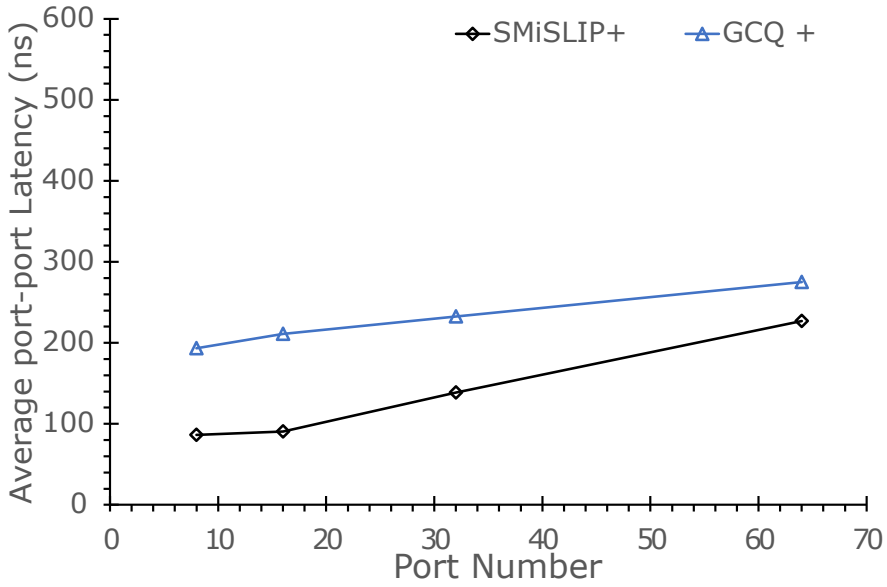
	Port-to-port (p-p) Latency (ns)						Avg. p-p latency (ns)	
Data Width	256 bit		640 bit		1280 bit			
# Ports	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ	SMiSLIP	GCQ
8	26.23	130.57	32.082	133.92	32.03	150.07	31.11	138.21
16	33.77	142.75	35.164	147.01	36.64	177.48	35.19	155.76
32	49.23	170.63	59.6	172.74	141.02	188.47	54.41	177.30
64	171.53	218.09	-	221.37	-			219.73

3.3.3 FPGA Resource Utilisation

Three floorplans of a 16 port switch are shown in Fig. 3.3 to demonstrate the scaling of each architecture in terms of resource utilisation. As expected, the basic crossbar switch with no buffer sharing has a high memory utilisation and is limited in scalability. The resource usage versus port number for various port line rates is presented in Fig. 3.5a and Fig. 3.5b for the SMiSLIP switch and the GCQ switch respectively. As can be seen, GCQ requires a higher number of logic resources compared with the SMiSLIP switch (FFs and LUTs). However, comparing the utilisation of the other resources is not as straightforward. Since the compiler views the rest of the available resources as a collective pool of memory resources, it attempts to make optimal use of the pool of resources while compensating for the lack of one resource with another based on the total resources available on the target device. This, in turn, obfuscates the



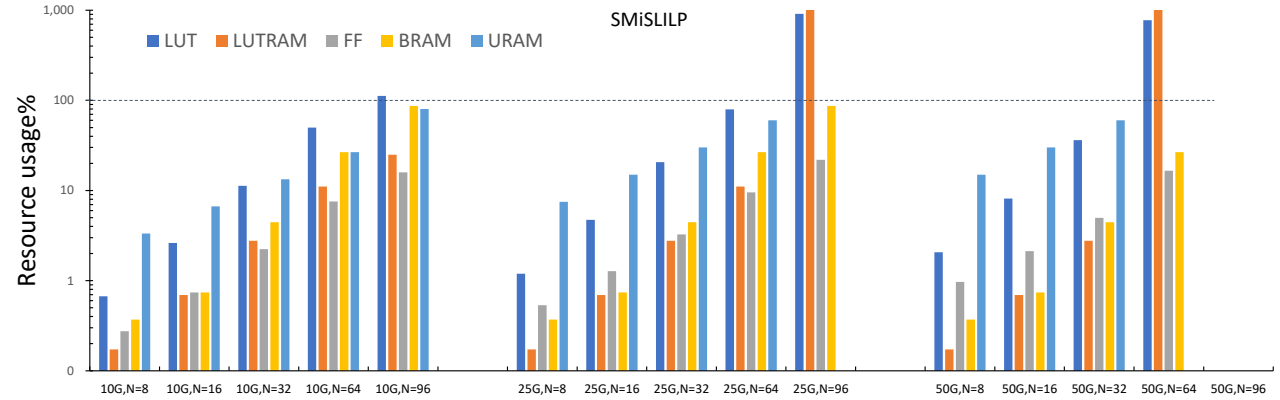
(a) Maximum link capacity vs. port number.



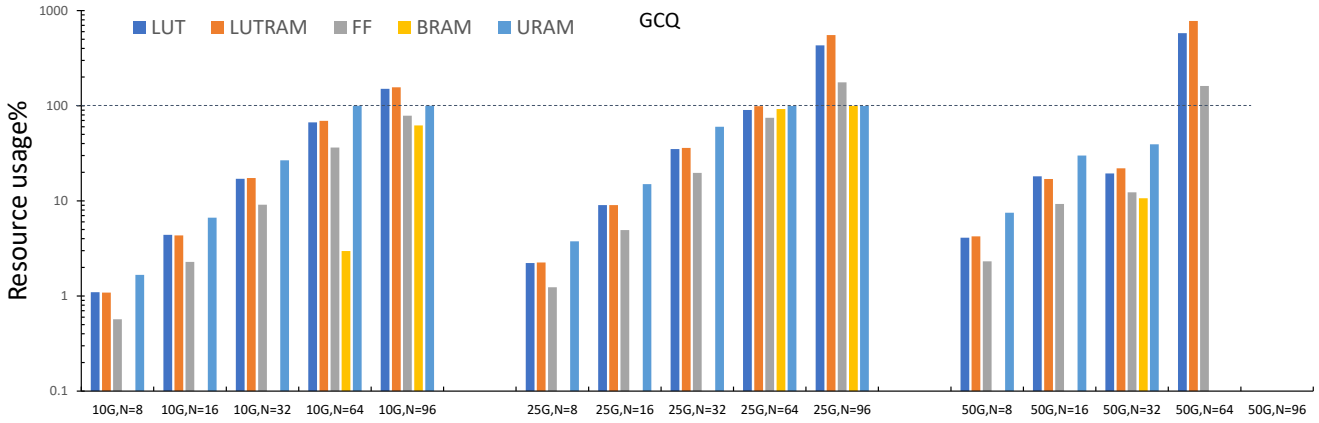
(b) Average port-to-port latency vs. port number, "+" means with transceiver latency added.

Figure 3.4: Switch performance scaling results.

relationship between the scale and the resource requirements of the design, as it is now subject to target-dependent optimisation constraints. We therefore propose an analytical model in Section 3.4.



(a) SMiSLIP resource usage vs. port number and line rate.



(b) GCQ resources usage vs. port number and line rate.

Figure 3.5: A base-10 log scale is used for the Y axis with the dotted horizontal line indicating the 100% resource usage of the target platform; For designs with usage over 100%, synthesis results are provided as they cannot be implemented on the FPGA; URAM is similar to BRAM but with larger unit block size and less flexibility, therefore shared memory module are implemented to first exhaust URAMs prior to using BRAMs to avoid wasting memory resources.

3.4 Analytical Model

We present a target-independent model that can be used to estimate the resource requirement of our proposed architectures at various scales for the future adoption of next-generation FPGAs. This enables us to estimate the resource requirements of each architecture and predict whether they can be implemented on a target FPGA while relying on the FPGA datasheet. This approach reduces design time by eliminating the need to repeatedly implement the designs while targeting different boards and scales. We also estimate the expected latency of each architecture to show their trade-offs in terms of both latency and resource utilisation.

3.4.1 Memory Resource Model

We rely on our existing implementations of the SMiSLIP switch and the GCQ switch to obtain a memory model that can be used to predict the memory resource requirements without being subject to target device constraints. When we consider the SMiSLIP switch, the shared-memory VOQs contribute to all the memory resource utilisation resulting in the equality shown in (3.1), where U_{sm} is the memory resources used by the shared memory VOQs.

$$U_{SMiSLIP} = U_{sm} \tag{3.1}$$

We further break down the shared memory VOQ architecture into the utilisation of its four main sub-components: **(a)** a shared memory buffer (U_{shared_mem}), **(b)** the memory pool (U_{mem_pool}), **(c)** the output queues (U_{out_q}), and **(d)** the indicator ($U_{indicator}$). Consider an N port switch with a data path width of W and queues with a depth of D. The shared memory buffer is expected to hold a total of N VOQs. This results in $U_{shared_mem} = DNW$ as shown in Fig. 3.2. The memory pool should contain all the addresses for the shared memory. Given the depth of $D \times N$, the input width of the memory pool equals $log_2(DN)$; hence the usage for memory pool $U_{mem_pool} = DNlog_2(DN)$. The output queues store the addresses issued by the memory pool with a depth of D, so the usage for output queues $U_{out_q} = DNlog_2(DN)$ The indicator

module should have the same depth as the memory pool and the data width of N to indicate the request status for N outputs. This results in $U_{indicator} = DN^2$. Finally, a total of N shared memory units are needed. In summary, the total memory usage U_{sm} for the shared memory can be expressed as:

$$U_{sm} = N \times (\underbrace{DNW}_{shared_mem} + \underbrace{2DNlog_2(DN)}_{mem_pool\&out_q} + \underbrace{DN^2}_{indicator}) \tag{3.2}$$

Similarly, the GCQ switch memory utilisation can be broken down into 3 sub-components: **(a)** the memory-based switches (U_{MBS}), **(b)** the VOQs (U_{VOQ}), and **(c)** the output buffers (U_{outbuf}). The core of the GCQ switch is the memory-based switch (MBS) which extends the shared memory architecture by adding multiplexers and de-multiplexers at the beginning and end respectively. As a result, the memory usage model for a MBS is very similar to that of U_{sm}, with the exception that the size of the MBS used in GCQ is fixed. With a speed-up of S, the total number of MBS needed in the GCQ design is $(\frac{N}{S})^2$, hence U_{MBS} can be expressed as:

$$U_{MBS} = (\frac{N}{S})^2 \times (\underbrace{DSW}_{shared_mem} + \underbrace{2DSlog_2(DS)}_{mem_pool\&out_q} + \underbrace{DS^2}_{indicator}) \tag{3.3}$$

The GCQ switch reduces the number of queues at the input to $(\frac{N}{S})$. The U_{VOQ} usage can then be calculated as follows with D_{VOQ} representing the depth of the VOQ at the input:

$$U_{VOQ} = N(N/S) \times W \times D_{VOQ} \tag{3.4}$$

GCQ requires additional asynchronous output buffers at the output whose usage can be expressed as the product of the number of buffers and a factor F to compensate for the resource overhead when using the IP generator. Factor F is calculated by the ratio of actual memory resources used by the IP to the theoretical memory usage without the IP. Because factor F maps the theoretical memory usage to the actual memory usage, the model reflects a more

accurate memory usage. The memory utilisation of a GCQ switch can then be given by:

$$U_{GCQ} = \underbrace{(\frac{N}{S})^2 \times (DSW + 2DSlog_2(DS) + DS^2)}_{U_{MBS}} + \underbrace{F \times N(N/S) \times W \times D_{VOQ})}_{U_{VOQ}} + \underbrace{F \times (N/S)^2 \times SWD}_{U_{outbuf}} \tag{3.5}$$

3.4.2 Port to port Latency model

SMiSLIP

The latency of iSLIP-based switches is dominated by the minimum clock period T of the scheduler. According to [34], the iSLIP scheduler needs $log_2(N)$ iterations to converge, which means each cell waits for at least $log_2(N)$ cycles in the buffer before it is scheduled to the output port. The latency of the iSLIP scheduler can then be calculated as $log_2(N)T$. Therefore, the latency of SMiSLIP in clock cycles can be given by:

$$D_{SM_iSLIP} = TC_{MBS} + log_2(N)T \tag{3.6}$$

where C_{MBS} is the number of clock cycles that are required for the cell to pass through the shared memory architecture.

GCQ latency model

The cross-clock domain logic and the pipeline stages in the GCQ design introduce latency overhead to the design. The latency model for the GCQ switch with N ports in clock cycles is shown below:

$$D_{GCQ} = T_sC_{wr} + T_sC_{rd} + T_fC_{MBS} + T_sC_p \tag{3.7}$$

where T_s and T_f are the clock period of slow and fast clock regions respectively. C_{wr} and C_{rd} are the write and read delay in terms of clock cycles. Additionally, C_p is the number of pipeline stages inserted to break the long wire connections within the switch and it is equal to the

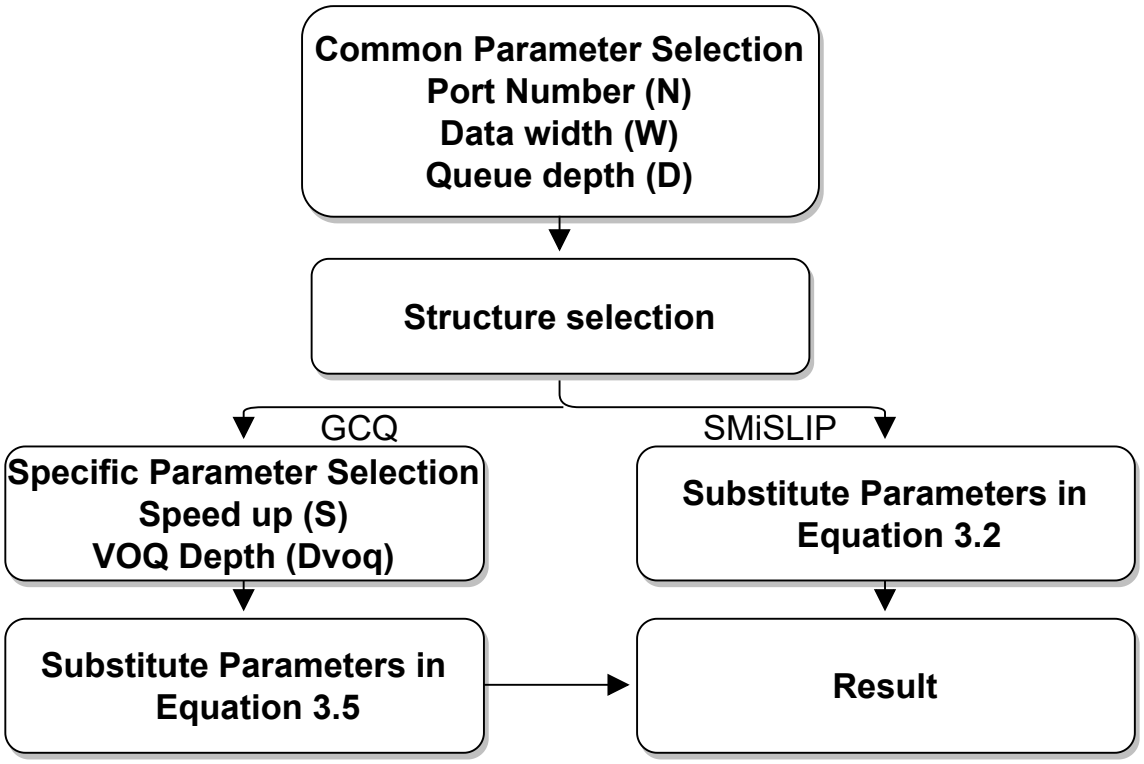


Figure 3.6: Memory model usage flow diagram. Note common parameters are parameters shared by both architectures and specific parameters are exclusive to GCQ only.

number of memory-based switches in a row.

3.4.3 Case study: Resource model use case

As an example of how our model could be used in practice, we sketch a possible workflow in Fig. 3.6. While we do not expect our model to give us the exact number of memory resources that end up being used on the FPGA by each architecture, we can use that to answer the following questions.

1. Is an SMiSLIP or GCQ switch implementation likely to fail due to insufficient memory resources on a target board?

We answer this question by computing the expected memory resource requirements for the points at the boundary conditions where the first design fails implementation due to insufficient memory resources. This corresponds to a 64-port design with a line rate of 50 Gb/s. Based on our model, we estimate that 356.515 Mb and 368.738 Mb are required for the SMiSLIP and

GCQ architectures respectively. Considering that the target board has a total of 382 Mb of cumulative memory resources (LUTRAM, BRAMs, and UltraRAM), our estimates are within a threshold (6%) for SM-SILP and (3.4%) for GCQ from the overall memory capacity. Based on those numbers, we hypothesise that designs are likely to fail due to insufficient memory resources when the model-based memory resource estimations are within close proximity of the determined thresholds. To verify our hypothesis, we run a number of tests for design sets that start to fail implementation. The results indicate that on average, the design fails with memory variation of 7.98% over for SM-iSLIP and 2.75% over for GCQ of the overall memory capacity. This implies that the synthesis tools might end up using more memory than actually needed. Therefore, while the model does not provide an exact matching number of resources, it can be used to predict the likelihood of failing implementation due to memory constraints, although more work is needed for rigorous statistical validation of the determined thresholds. Next, we use our model to predict the lower bound of FPGA memory resource requirements for extrapolating the memory requirements of FPGA switches at various scales. An example is as follows.

2. What is the minimum memory requirement for implementing a 128-port SMiSLIP or GCQ switch operating at a line rate of 50 Gb/s ? Based on our model, we determine that SMiSLIP and GCQ need 1503.65 Mb and 1474.9 Mb memory respectively for a switch of such a scale. Considering this, and the amount of on-chip memory resources available on FPGAs, we observe that the memory requirements of both designs cannot be supported by the current generation of UltraScale+ FPGAs, which has a maximum of 502.8Mb RAM (consists of Distributed, Block and Ultra) as reported in [80]. For FPGAs with on-chip HBMs (with a typical size of 4GB, 8GB and 16GB), despite the large size, the access latency is high compared to the other on-chip memory types, making them less suitable for low-latency switch design.

3.5 Sorting network-based high throughput switch design

This section is part of the collaborative work with Papaphilippou [1]. The material presented below, unless noted otherwise, is my own work.

Different from the crossbar-based switches with VOQ as a buffer, which requires more complex control logic for the scheduling algorithm, the sorting network-based design as shown in Fig. 3.7 can be treated as an output queued switch design that works with simple round-robin algorithm, with the number of queues at each output port equals the input port number, this space division queuing method avoids the speeding up of a single shared output queue. The multistage nature of this design enables heavy pipelining, which suits the FPGA platform well. As a result, this design achieves 1.7 Tb/s throughput for a 16-port design with 512-bit data width.

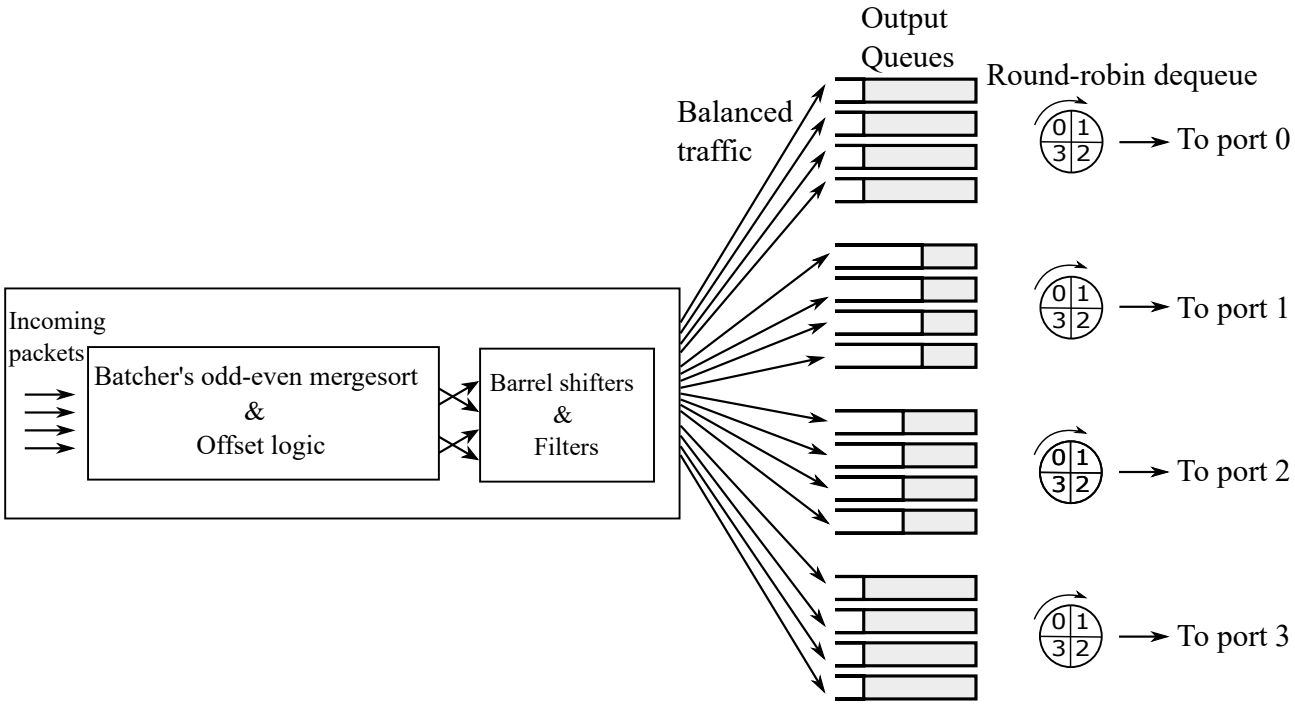


Figure 3.7: Abstract view of the design with 4x4 ports. As the detailed architecture and implementation are presented in Hipernetch [1], we thereby only show the abstract design and analyse its similarity to an output-queued design.

In this section, we first present the comparison between the sorting network-based design and the aforementioned crossbar-based designs. Then based on the result of the comparison, we

provide an analysis of sorting network-based switches and show their resemblance to output-queued crossbar-based switches, which explains why this design outperforms the aforementioned crossbar-based designs. Because Output-queued (OQ) network switches have been shown to perform better than input-queued (IQ) switches [36].

3.5.1 Performance comparison with the crossbar-based designs

For a fair comparison, we re-run the experiment on a newer U280 board and use the same wrapper logic for a fair comparison. The result is shown in Fig. 3.8 The implementations here were redeveloped for the same target FPGA (Xilinx Alveo U280). Note that this comparison is mostly indicative, as some assumptions were made for each implementation, such as to be out-of-context as well (no actual transceiver is populated) and featuring different buffer lengths and memory types. There could also be additional effort to optimise the implementation of the competing designs.

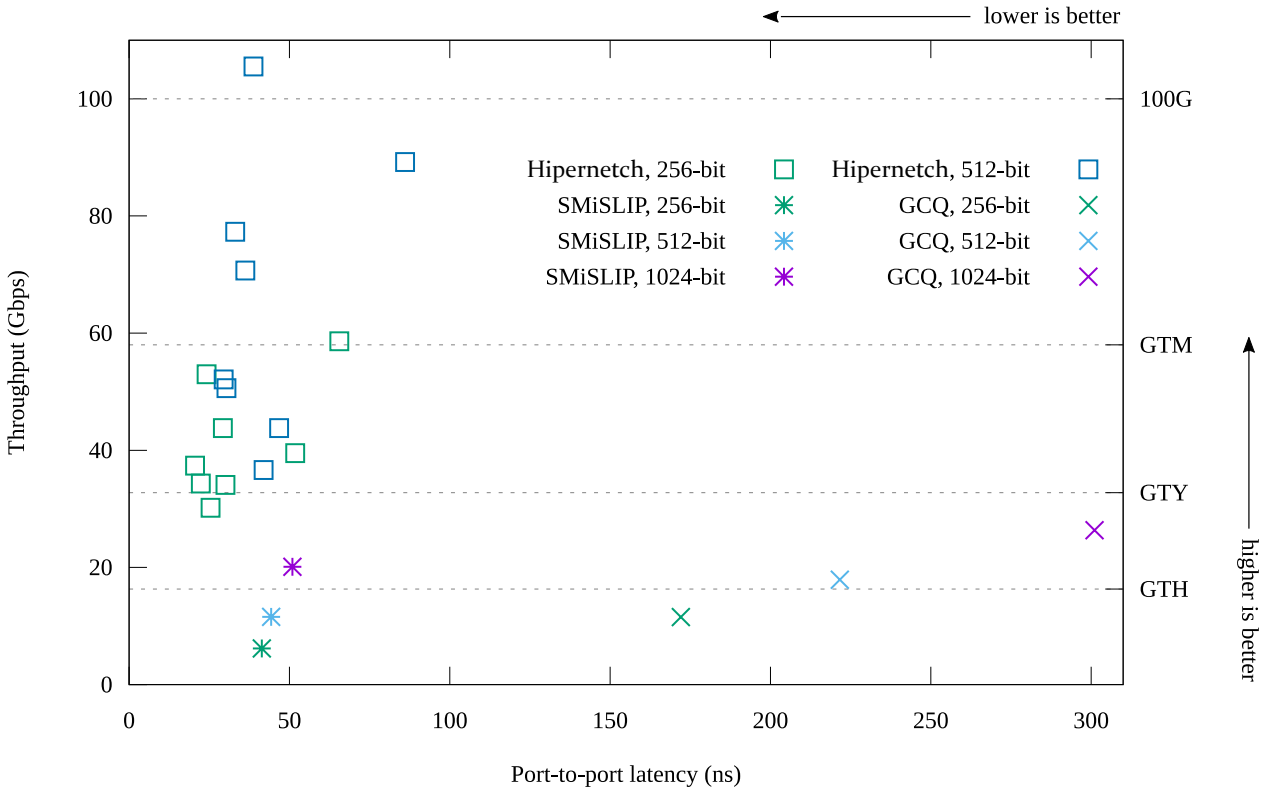


Figure 3.8: Comparison of 16×16 switches implementations on Xilinx Alveo U280. Source: [1].

Fig. 3.8 shows how the design space for the proposed design performs against the related work,

based on the throughput and port-to-port latency. For a packet size of 256 bits, the sorting network-based design achieves a throughput of 53 Gbps, which is around 8.5 and 4.6 times the throughput of the 256-bit versions of SMiSLIP and GCQ respectively. SMiSLIP seems to sometimes marginally win for port-to-port latencies, but our design space features choices that are better in both the throughput and latency, including the aforementioned data point, which has a latency of 24.2 ns. Also, since SMiSLIP uses the iSLIP scheduling algorithm, there will be additional latency for high input rates, as found in our simulations. GCQ yields the worst latencies irrespective of packet size. For a packet size of 512 bits, we achieve the highest throughput overall at 105.5 Gbps and a latency of 39 ns, saturating the UltraScale+ GTY and GTM transceiver, as well as for 100 Gbps or equivalent.

In terms of hardware resource usage, the sorting network-based design requires less memory but requires more LUTs and FFs compared with the crossbar-based designs. Although performs better in terms of throughput and latency, the sorting network-based design is less scalable, with a maximum number of ports of 32. This can be explained by the heavy usage in LUTs and FFs of the arbiter logic.

3.5.2 Resemblance to output-queued crossbar-based switches

Output-queued crossbar switches (OQ-xbar) are considered as the ideal switch architecture and most alternative architectures strive to emulate their performance [36, 87, 88]. They consist of a crossbar fabric and a FIFO-based queue for each output port. In crossbars, conflicts happen when packets from different input ports are targeting the same output port at the same cycle. To solve this problem, both the crossbar and the queues use P times higher clock frequency than the base clock (fabric speed-up of P times, where P is the number of ports). This gives the illusion that packets arriving at the input ports are forwarded "immediately" to the appropriate destination port queue.

In practice, the main performance bottleneck of the output-queued crossbar switch is the need for a high memory accessing speed. Alternatives have been researched to match the performance of an output-queued switch. These solutions, however, are all based on the speed-up of the

switch fabric. In [89], a speed-up of 2 in a CIOQ switch was proven to be sufficient to emulate an output-queued switch in theory. However, the need for scheduling algorithms that run with a speed-up of 2 for switches with high port rates is challenging in general, and even more challenging on FPGAs due to their low operating frequencies. In this section, we demonstrate that our proposed design approaches the performance of the output-queued crossbar, though it does not necessarily behave identically to an OQ-xbar switch. More specifically, two packets arriving on the same cycle with the same destination can appear in a different order at the output port, but this does not affect the common performance metrics.

In high-level, an OQ-xbar switch with P ports can enqueue up to P packets to a single output port buffer immediately. The order in which these packets are written to the same destination port buffer depends on an arbitration policy used which can be as simple as a round-robin arbitration policy. Our proposed switch in contrast relies on a sorting network to perform this arbitration, and therefore provides no deterministic order in which packets are written to the destination port buffer. This is due to the fact that sorting networks are not stable sorters. Another reason the behaviour of our design is not exactly identical to that of an OQ-xbar is that packet drops can cause our optimised output arbiters to get desynchronised. In the rest of this subsection, we explain how these differences in behaviour impact the performance of our proposed design, and how it compares to an OQ-xbar in terms of the average latency, worst-case latency, throughput and packet loss rate.

Latency Two metrics for measuring the switching performance are the *average* and the *worst-case* latency. The average and worst-case latency is measured by counting the average and maximum number of cycles respectively a packet needs to be served from the input port to the output port.

Without loss of generality, we study the events in a single output port out of the P available. With respect to the latency of packets destined for the same output port, there are two cases to consider: (1) packets arriving on different cycles and (2) multiple packets arriving on the same cycle.

For (1), any two packets A and B arrive on different cycles t_A, t_B for the arrival time of A and B respectively, where $t_A < t_B$ (B precedes A). Both the *combined parallel round-robin arbiter* and the output-queued crossbar have a fixed processing latency. Thus it is guaranteed that the order of packets A and B will be conserved (each being served on the arrival time plus the processing latency) and (1) is not a differentiating factor.

For (2), multiple packets (up to P packets) arriving on the same cycle, it is shown that different permutations do not affect the latency metrics. Let $t_1, t_2, ..., t_p$ the timestamps for the $i \in (1, 2, ..., P)$ consecutive packets to be dequeued. Note that these timestamps are one cycle apart from each other, due to the round-robin arbiters in the end. t_0 denotes the common timestamp of arrival in the input. The worst-case (t_{worst}) and average (t_{avg}) latencies are given by:

$$\begin{aligned} t_{worst} &= max((t_1 - t_0), (t_2 - t_0), ..., (t_i - t_0)) \\ &= max(t_1, t_2, ..., t_i) - t_0 \end{aligned} \tag{3.8}$$

$$\begin{aligned} t_{avg} &= ((t_1 - t_0) + (t_2 - t_0) + ... + (t_i - t_0))/i \\ &= (t_1 + t_2 + t_3 + ... + t_i)/i - t_0 \end{aligned} \tag{3.9}$$

Due to the associative and commutative properties of both the max operation and summation, the queuing order at the output will not affect the worst-case or average latency for both our design and the OQ-xbar.

As the average latency is shown to be identical, we can verify this finding by comparing each set of output queues to an M/D/1 model, for example, as it is shown to model the OQ-crossbar as well [90, 91].

In our design, the highest arrival rate for the queue group in the output is 1 packet per cycle (corresponding to 100% input rate under uniform Bernoulli traffic), and the highest serving rate is also one packet per cycle per port. The queue utilisation is given by $\rho = \frac{\lambda}{\mu}$, where λ (or r, ranging from 0 to 1) is the arrival rate and μ is the serving rate. For instance, achieving the

steady state of the model requires $\lambda \leq \mu$, otherwise, the queue size would grow indefinitely.

As the average latency metric comparison assumes infinite queues, we choose to set μ to 1 due to the dequeue policy of processing one packet per cycle (as with the OQ-xbar). The average latency ω_Q of a packet in such an M/D/1 queue can then be given by:

$$\omega_Q = \frac{\rho}{2\mu(1-\rho)} = \frac{\frac{\lambda}{\mu}}{2\mu(1-\frac{\lambda}{\mu})} = \frac{\lambda}{2(1-\lambda)} \tag{3.10}$$

By mapping the average latency to the y-axis and the input rate to the x-axis we can compare this model to our simulation results. This simulation dataset represents the average latency for the entire switch (including the queues for all output ports), using uniform Bernoulli traffic. There is a minor displacement due to a convention to wait to empty the queues, marginally affecting the effective input rate in the simulation. The average latency ω_Q of multiple M/D/1 queues equals to the average latency of a single M/D/1. Thus, observing this juxtaposition in Fig. 3.9 indicates that our switch design follows the output-queuing model.

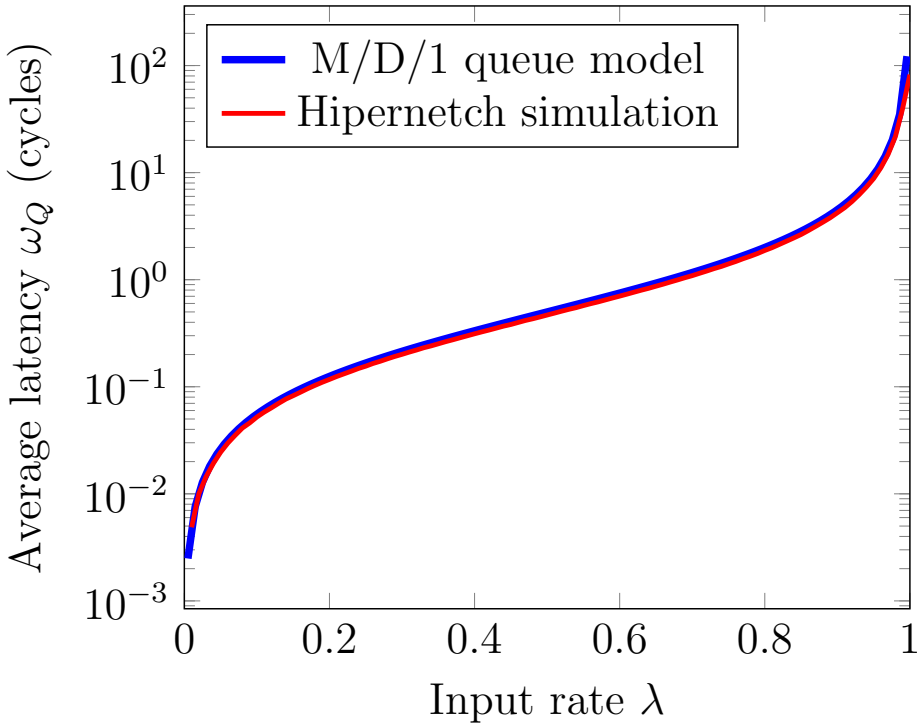


Figure 3.9: Average latency comparison of M/D/1 queueing model and simulation result.

Throughput. The throughput is measured as the number of exported packets for a given time

period. Both the output-queued crossbar fabric and the *combined parallel round-robin arbiter* are collision-free, i.e. given at most P packets targeting the same port, arriving on the same cycle, where P is the number of ports, they can all be processed. In the case of the crossbar, this is achieved by expanding the time domain (i.e. speeding up both the fabric and the queues P times). In the case of our design, this is achieved by rearranging the packets in line rate using non-blocking components and increasing the parallelism of the space domain (i.e. accessing the FIFOs in banks). They both support the line rate, sometimes referred to as "supporting 100% of the throughput", although this holds as long as there is adequate free space in the queues. As network switches do not have an infinite amount of queue space, the throughput is reduced by packet losses.

Packet loss rate. In terms of the packet loss rate, our proposal performs similarly to OQ-xbar with queues of the similar total size. As this metric relates to a fixed queue size, the implementation of the output arbiter and desynchronisation workaround also affects the packet loss behaviour. For brevity, we show that our proposal with any simpler output arbiter choice is no worse than an OQ-xbar with $(M - P)/M$ of the total queue space for a port, where $M = P * D$ is the space of an output queue group and D is the queue depth. Note that this is a worst-case and also between the workarounds there is, for instance, the backpressure solution which would perform the same as an OQ-xbar with the same total queue size. The longer the FIFOs, the less important this difference becomes.

On each cycle, due to the round-robin enqueue and dequeue policies in our solution, we notice that the available spaces will be consecutive (cyclic) and evenly distributed across the P output queues of the port. Therefore, regardless if the output arbiter for the port is synchronised or not, when there are P or more free spaces ($F \in P, P + 1, ..., P \times D$), all packets arriving on that cycle will find a place in the FIFO group of the port and will not be dropped. Therefore, we notice that it performs at least better than an OQ-xbar with $(M - P)/M$ the queue space (with P fewer spaces).

3.5.3 Comparison to ASIC-based switches

To compare the latency of our design with ASIC-based switches, we collect latency values reported in the datasheet from well-known switch manufacturers, and the comparison is presented in Fig. 3.10. Note that for ASIC-based switches, most vendors provide the best latency among a series of products, and the relationship between the best latency and the number of ports as well as the line speed is unknown. Thus, our reported best latency is chosen from designs of different line speeds, given a fixed number of ports. The numbers in parentheses are the available number of ports supported by the corresponding switch.

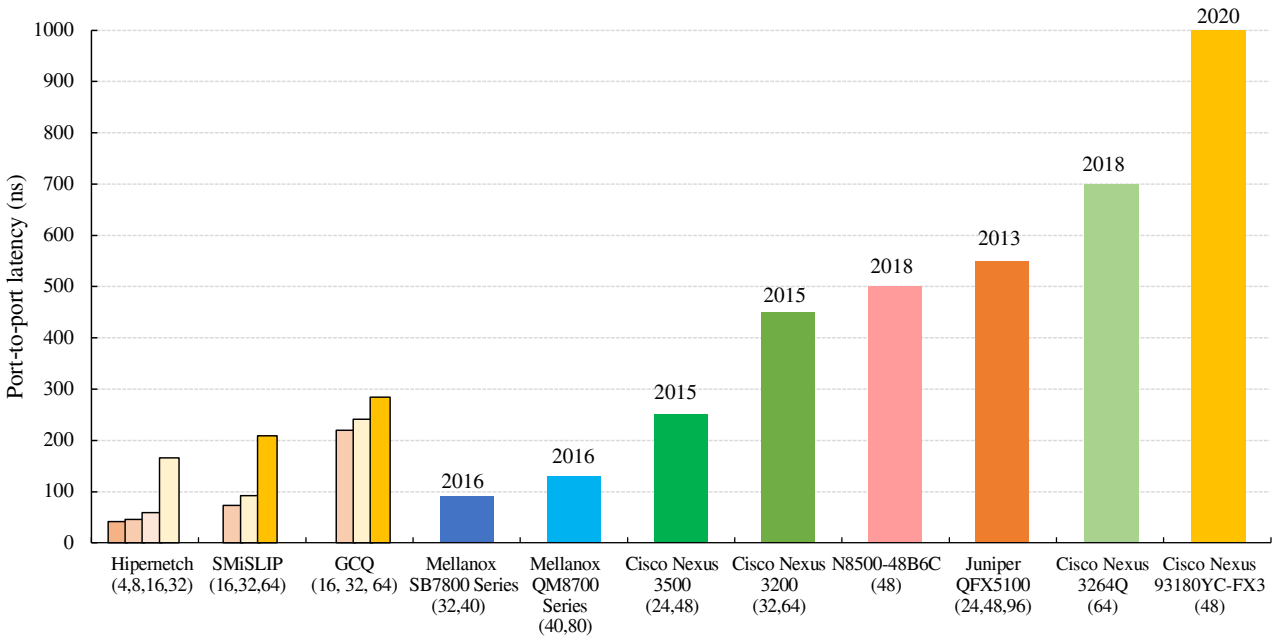


Figure 3.10: Latency comparison with ASIC-based switches [4–11] (from left to right).

Our design is shown to yield very low latencies, but it has the most basic port forwarding function, and is implemented in our switch with a custom packet format. As our design only considers the latency inside the FPGA, it is not appropriate to compare with ASIC-based switches without the same interface logic (SFP+/QSFP+). Thus, we add the corresponding latency of such an interface to our design. Taking the Ethernet switch as an example, we make an estimation using the 10G/25G Ethernet Subsystem from Xilinx [92], which gives the lowest latency of 36.8 ns and 46.08 ns for 10GE and 25GE design respectively. To achieve the lowest latency, we choose 36.8ns for our estimation.

In order to compare with other Ethernet switches, the port-to-port latency was updated as explained and corresponds to a very basic implementation with no additional functionality. As can be seen in the figure, the Hipernetch and SMiSLIP designs have the lowest latency among the ASIC-based switches, but for up to 16 ports. For 32 ports, Hipernetch shows a higher latency than the SMiSLIP; this is because of the noticeable performance overhead from approaching the limits of the FPGA device related to the Stacked Silicon Interconnect (SSI).

Due to the long iteration time of ASIC products, ASIC-based switches tend to pack functions that might not be used by the customer in the switch. These additional functions potentially affect the latency of the switch. In contrast, for an FPGA-based switch design, it is the choice of the user to add their desired functions. Therefore, FPGAs can minimise the latency overhead introduced by the redundant functions. Observing the latency difference between our design and the slowest ASIC-based switch, our switch provides a time window of approximately 600 ns for the user to include additional functions like packet processing, and still able to compete with the ASIC-based switches. Moreover, there will potentially be a frequency boost when transitioning from FPGA to ASIC implementations [93].

One limitation of FPGA-based switches is the number of ports that a single FPGA can support. Although our design shows the ability to saturate the fastest transceiver bandwidth on FPGAs, it can be challenging to utilise all the available transceivers on board, due to routing and other resource limitations. However, it is possible to overcome such limitations by utilising multiple switches, or in different FPGA, hierarchically. Network topologies such as Clos networks [29] and fat-trees [94] are typical methods to construct large switching networks with smaller switching modules. Such methods can increase the scalability of FPGA-based switch designs at the expense of latency. These can also be regarded as a workaround for bigger switches before improved generations of FPGA devices are released.

3.6 Summary

In this chapter, we study the feasibility of FPGA-based switches to answer our primary question of whether such architectures can be used to implement FPGA-based switches that can match the performance of ASIC-based switches used today. Our experiments show that the maximum number of ports achieved is 64, this makes FPGAs today more suitable for implementing switches with lower port count requirements. We further determine the trade-offs of each proposed architecture in terms of performance and scalability. Among the three designs presented, Hipernetch has the lowest port-to-port latency for up to 16 ports. Beyond 16 ports, the SMiSLIP architecture shows the lowest port-to-port latency. Though the SMiSLIP architecture has lower port-to-port latency than the GCQ switch, it exhibits limited link capacity for high radix design. This is due to its higher resource utilisation as shown by the target independent model, as well as to the difficulty in achieving higher operating frequencies due to its centralised architecture. The GCQ switch, instead, demonstrates better scalability in terms of memory resource requirements and consistency in supported link capacity, although scaling it further is currently limited by routing resource congestion. Further, we observe that achieving timing closure became more challenging as the size of the switch increases since it is possible for a switch fabric to cross two super logic regions (in Xilinx FPGAs) to connect with the transceiver.

In terms of meeting the requirements mentioned in chapter 2, FPGA-based switches meet **R1** in port-to-port latency when compared with ASIC-based switches, making them good candidates for edge data centres where latency is of vital importance. As for **R2**, FPGA-based switches can support a range of port counts. Hipernetch is more suitable for less than 32 ports design. SM-iSLIP and GCQ more suitable for less than 64 ports design; this meets **R2** when compared with relatively lower-end ASIC-switches [6] as presented in section 2.2.1.

In terms of resource usage, we measure the usage percentage for a 16x16 switch on the Xilinx U280 Alveo. The result as presented in Table 3.6 is obtained by fixing the same throughput for all three designs. All three designs show similar usage in LUTs and FFs, this is because Hipernetch due to its pipelined design runs at a higher frequency and therefore requires smaller

Table 3.6: Resource summary for Xilinx U280 Alveo

	Memory	LUTs	FFs
Hipernetch	<1%	14%	6%
SMiSLIP	<23%	17%	1%
GCQ	<24%	17%	4%

data width to achieve the same throughput. For memory usage, because the type of on-chip memory used depends on the compiler optimisation strategy, we only show the total usage percentage. The Hipernetch design uses a shallow buffer and uses the least memory resources. Additionally, none of the designs uses DSP resources.

In conclusion, current FPGA can match ASIC-based switches in either link capacity, port number or switching latency. When combining them, they are more likely to match with low-end ASIC switches as mentioned in Chapter 2. However, similar to ASIC-based switches, it is possible to scale the port number by connecting multiple FPGA-based switches, with a sacrifice in switching latency. This opens an opportunity for FPGA-based network switches in hyperscale data centres when latency is not the most crucial requirement.

The rapid growth in the size of FPGAs and their performance can help circumvent these gaps and enable larger and faster FPGA-based switches that do not rely on a centralised scheduling architecture. With 3D stacking [95], more resources and short routing can be achieved on the FPGA. Also, with hardened Networks-on-Chips (NoCs) finding their way into FPGAs, the challenges of meeting timing closure can be alleviated, thus enabling even lower latency and resource area usage, which in turn could help bring the performance of FPGA-based switches closer to commercial switches.

In addition to packet switching, FPGA-based switches can handle computation-intensive operations which is not offered by the ASIC-based switches. In the next chapter, we discuss applications that can benefit from FPGA-based network switches.

Chapter 4

Ensemble Model Training

Random projection is gaining more attention in large-scale machine learning. It has been proved to reduce the dimensionality of a set of data whilst approximately preserving the pairwise distance between points by multiplying the original dataset with a chosen matrix. However, projecting data to a lower dimension subspace typically reduces the training accuracy. This chapter proposes a novel architecture to support ensemble learning on an FPGA-based network switch. This architecture enables reducing training time while maintaining high accuracy. Our initial result shows a speed-up of 2.1-6.7 times on binary classification using four different high-dimensionality datasets. The speed-up is defined by the ratio between the time taken for a single model to train the data to the time taken for the ensemble model to train the data, as shown in Eq. 4.10.

4.1 Introduction

Due to the location of the network switch and the resource usage of the switch design, not all application suits the FPGA-based switch. We, therefore, identify three traits for an application to be well-mapped to the FPGA-based switch. Such traits are:

1. lightweight, e.g., with streaming algorithm support, heavy-load applications would com-

pete for on-chip resources and affect the performance of the switch. Besides, the resources for the applications should not exceed the spare resources obtained from Table 3.6.

2. requiring frequent communication with other servers, e.g., weight update in distributed machine learning. Located at the centre of communications, handling collective operations such as Gather or Reduce at the switch reduces the communication cost compared to handling those operations using a dedicated server. Another type of application that falls in this scope is data compression within the switch, a reduced data size reduces the bandwidth usage and improves communication speed.

3. requiring low response latency (high response speed), e.g., distributed memory caching, so that response time-sensitive applications could save time by only accessing the switches instead of the servers. The earlier a request can be answered in the data centre topology the better, in other words, the preference is switch > NIC > server.

This chapter presents an example application namely accelerate ensemble learning with an FPGA-based switch that leverages trait 2 through the ensemble of multiple servers. Chapter 5 presents another application that leverages the remaining two traits.

Recent years have witnessed a significant surge in the size of data available for machine learning tasks. Though this is great for improving training accuracy, it is computationally challenging. Reducing the high dimensionality of the data becomes crucial for accelerating machine learning. Random Projection (RP) has been proved to be a powerful method for dimensionality reduction. However, increasing the sparsity of the RP matrix and thereby reducing the projected dimensionality inevitably results in the loss of information within the dataset. This, in turn, impacts the training accuracy of machine learning applications.

An ensemble classification system comprises of multiple classifiers. These classifiers can be diverse in model type as well as predictive accuracy. The diversity of the classifiers allows them to correct and compensate each others' shortcomings, thereby achieving a collective accuracy that is higher than that of the individual classifier. Combining RP and ensemble systems allows classifiers to gain diversity from data in different sub-spaces, and the system to recover

the accuracy loss with the ensemble.

Network switches as the pivot of data centre networks are possible places to introduce diversity, however, ASIC-based network switches are fixed in functionality which is not suitable for computational tasks. Previous work [39, 47, 96, 97] shows the capability of FPGAs for implementing network switches. Using FPGA-based switches enables network function customisation, on-path aggregation for traffic reduction and network application acceleration. However, such design typically leaves computational resources such as Digital Signal Processing (DSP) blocks unused, which can be utilised to implement additional functions. We therefore propose an FPGA-based network switch architecture to deploy RP ensemble method across multiple servers, with the switch section responsible for distributing training data and the DSPs for random projection.

The challenges of designing such systems are: **(C1)** Processing large dimensional data requires a large amount of FPGA resources, thereby limiting the scalability of the system. **(C2)** Performing fast computation requires storing the RP matrices on-chip, and this becomes challenging as the dimensionality of input data increases due to limited on-chip storage. Through model analysis, we tackle the aforementioned challenges, and provide a generalisable solution for various scenarios. The main contributions of this chapter are:

1. A novel architecture for random projection ensemble method that utilises the spare resources (as shown in Table 3.6) of an FPGA-based switch (section 4.2).

2. Potential speedups in training whilst maintaining or even surpassing the single model accuracy(section 4.5). As presented in Fig. 4.5, the single model accuracy is represented by the dotted line.

3. An analytical model that benefits future deployment of our system (section 4.5.1). The performance model helps predict the speed-up for data with different dimensions. The individual component analysis in Fig. 4.4 helps to tune the optimised RP time (section 4.5.3).

4.2 System architecture

The system comprises two major parts as shown in Fig 4.1, the FPGA-based switch for data preprocessing and the server nodes for training. Within the FPGA, on top of the switch functionality, multiple DSP-based matrix multipliers are used for the RP process. The system starts with a network storage server, sending the training data to the FPGA through a high-speed interface. Upon entering the FPGA-based switch, the data within the packets are either sent directly to the output port (forwarding mode), or enter the DSPs where matrix multiplication is performed with the pre-stored RP matrices (processing mode) before being sent to the output port depending on the request field in the header. The server nodes then use the received data to perform training. We make no assumptions on the type of server nodes, and they could range from simple development boards to server clusters. Finally, the outputs of the switch stream into the servers where training happens independently. Servers are platforms capable of running machine learning training.

Note that the ensemble method allows servers to hold different types of models, however the slowest node will affect the completion time of the system, making training tasks difficult to schedule. For the sake of consistency, we choose to have the same type of model running in all the servers, for example, multilayer perceptrons (MLPs) in our case. Each output port is assigned a unique projection function. Therefore, the servers receive data projected into different sub-spaces, ensuring the diversity of trained models in the ensemble.

Our design can be deployed as an **extra layer** of switching between the ToR switch and server nodes, illustrated by the right half of Fig. 4.1. This approach allows a reduction of the number of servers connected to each FPGA and keeps the functionality of the traditional rack. As a result, more resources on the FPGA can be utilised to compute random projection. Replacing the ToR switch with FPGA-based switches, in contrast, would require the FPGA to scale with the number of servers connected to the ToR switch and this would limit the resources available for computation. Such a hierarchical approach allows us to address **C1** as the number of server nodes increases.

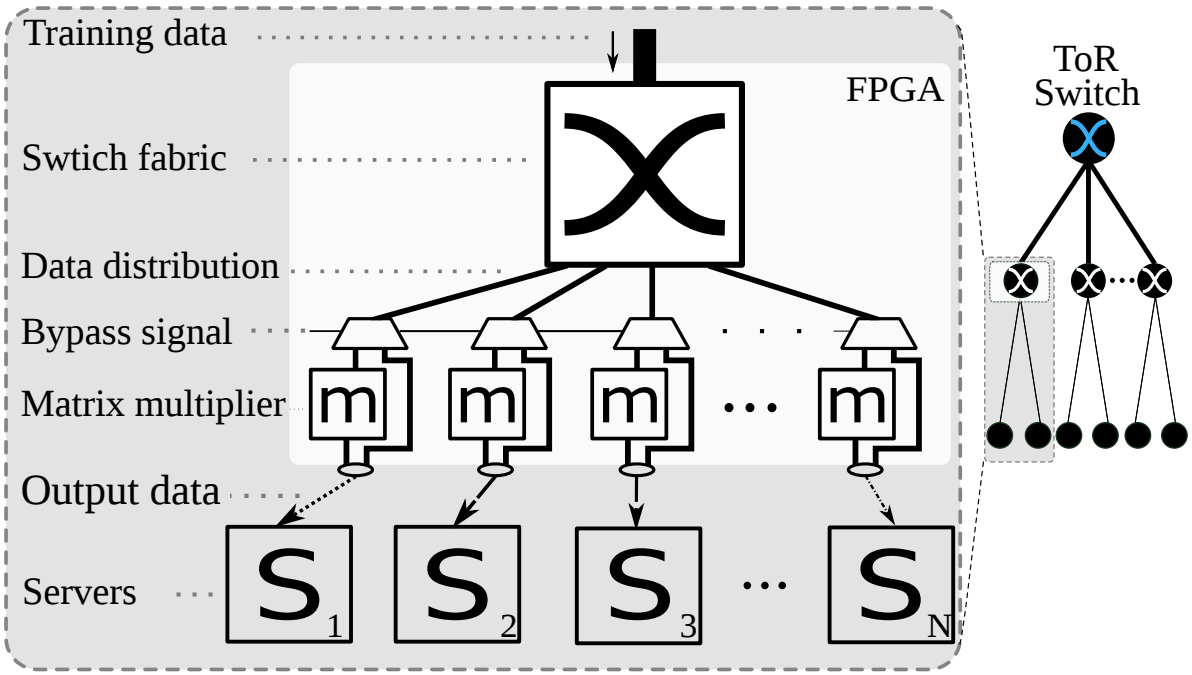


Figure 4.1: System flow

4.2.1 On-chip storage reduction for projection matrices

On-chip storage avoids extra data transfer latency of the RP matrix from the off-chip memory and is necessary to perform fast data projection within the FPGA-based switch. The size of the projection matrix becomes considerably large when the data dimension increases, which can become too large for the on-chip memory resources. Observing that the sparse matrix consists of only three values (positive, negative and zero) as shown in [52], we propose to represent matrix value with only 2 bits and restore its value only when being used, this is because the absolute value of the positive and negative value is the same. To restore the value, a 2-to-n-bit lookup table is needed for each DSP-based multiplier, where n depends on the data type, for example, n equals 16 for a 16-bit fixed-point number. This process only takes one cycle to complete and incurs minimal resources overhead. This method allows us to store the RP matrix on-chip with only a little resource overhead, thereby addressing challenge **C2**. Even for a dataset with extremely large dimensions that cannot fit within on-chip memories, this method still reduces the transmission time between off-chip memories and the FPGA.

4.3 Theoretical model

Our theoretical model analyses the system time using boundary conditions. Aiming to optimise the system time, it provides conditions when maximum computation power can be utilised by the system. However, The theoretical model is constructed under the condition that FPGA has enough resources to compute the RP in a fully pipelined way, its result could deviate with different implementations.

Random projection can be performed by the switch and/or by the server nodes. Let α be the fraction random projection computed by the switch, the rest $(1-\alpha)$ computed by the servers. The overall system time consists of three components (a) the random projection time $T_{project}(\alpha)$ (b) the release time of output ports $t_{release}(\alpha)$ and (c) the training time of the server T_{train}. It is possible to implement the random projection with systolic matrix multiplication. In this case, the output ports can send data to the servers while the random projection is in progress. In other words, the random projection time and data release time overlap. Therefore, we take the greater time of $T_{project}(\alpha)$ and $T_{release}(\alpha)$. Ignoring the latency caused by the network and the pipeline, the total system time is as follows:

$$T(\alpha) = \max(T_{project}(\alpha), T_{release}(\alpha)) + T_{train} \tag{4.1}$$

$$T_{project}(\alpha) = \frac{\alpha X}{\theta^{switch}_{project}} + \frac{(1-\alpha)X}{\theta^{server}_{project}} \tag{4.2}$$

where X is the number of data points for training a sub-model in the ensemble; $\theta^{D}_{project}$ is the throughput of random projection for device D measured by the number of data instances projected per second.

The release rate of data should stay below the bandwidth of outgoing links due to congestion control. The release time to a server is the total data traffic $F(\alpha)$ divided by the bandwidth b from the switch to the server. The data traffic for each server includes the original data and

projected data sent to the server, i.e.

$$F(\alpha) = \alpha Xk + (1-\alpha)Xm \tag{4.3}$$

where m and k are respectively the number of dimensions of the original data and the projected data. Assume that all servers have the same bandwidth b measured by the number of data entries transferred per second, the release time is

$$T_{release}(\alpha) = \frac{F}{b} = \frac{\alpha Xk + (1-\alpha)Xm}{b} \tag{4.4}$$

We propose to handle all calculations for random projection in the FPGA-based switch with spared resources, which corresponds to $\alpha = 1$. In contrast, in a conventional system with an ordinary switch, the random projection works solely on the servers, which corresponds to $\alpha = 0$. The difference in performance is as follows:

$$T(1) - T(0) = X\big[\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) - \max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b})\big] \tag{4.5}$$

The proposed system outperforms the conventional one if and only if

$$T(1) - T(0) < 0 \tag{4.6}$$

This inequality holds if and only if

$$\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) - \max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b}) < 0 \tag{4.7}$$

Note that inequality 4.7 is independent of the number of data points X or the time spent on sub-model training T_{train}.

Inequality 4.7 is very likely to hold under practical settings. In particular, there can only be four cases for the maximum operations.

1. $\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) = \frac{k}{b}$ and $\max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b}) = \frac{m}{b}$. Since $k < m$ is the fundamental setting of

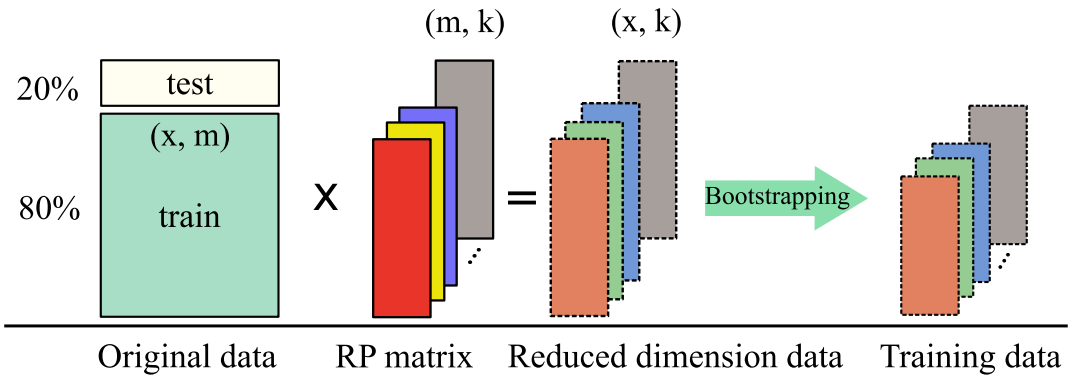


Figure 4.2: Dimensionality reduction data flow.

random projection and $b > 0$, we have $\frac{k}{b} < \frac{m}{b}$. Therefore inequality 4.7 obviously holds.

2. $\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) = \frac{k}{b}$ and $\max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b}) = \frac{1}{\theta_{project}^{server}}$. In this case, $\frac{k}{b} < \frac{m}{b} \leq \theta_{project}^{server}$. Therefore, similar to the previous case, inequality 4.7 still holds.

3. $\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) = \frac{1}{\theta_{project}^{switch}}$ and $\max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b}) = \frac{1}{\theta_{project}^{server}}$. In other words, the random projection time is the bottleneck of both systems. In this case, the inequity holds if $\theta_{project}^{switch} > \theta_{project}^{server}$, which holds as long as DSPs on the FPGA have a speed advantage over the server in terms of matrix multiplication.

4. $\max(\frac{1}{\theta_{project}^{switch}}, \frac{k}{b}) = \frac{1}{\theta_{project}^{switch}}$ and $\max(\frac{1}{\theta_{project}^{server}}, \frac{m}{b}) = \frac{m}{b}$. In this case, whether inequality 4.7 holds is more data-dependent and hardware-dependent. However, we found that the case is very unlikely to appear practically.

4.4 Experiment setup

The experiments are primarily carried out in simulation with several binary classification datasets from the UCI repository [98]. The tested model is a two-layer MLP classifier from Keras [99]. The simulation code and data are open sourced[1].

Based on Eq. 4.8, training time is collected from software simulation and switch time is calculated manually for better estimation. The simulation flow is illustrated in Fig 4.3. Aiming to

[1]https://github.com/jiuxi-pub/Ensemble_learning

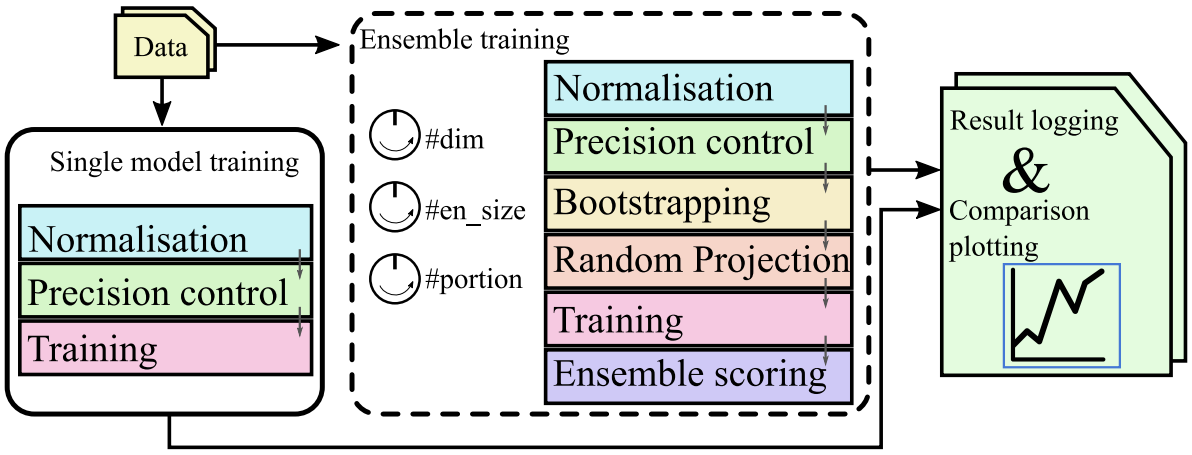


Figure 4.3: Simulation flow.

find the impact of dimension reduction on the training time and accuracy, different numbers of servers (*#en_size*), size of dimensionalities (*#dim*) and the portion size of bootstrapping (*#portion*) are passed to the simulator as parameters. As for accuracy evaluation, the ensemble process is achieved through simple majority voting among the predictions of all servers.

As depicted in Fig 4.2, original training data are randomly sampled using bootstrapping [56] with predefined proportions to form different sub-training data for each server. After random projection, sub-training data of shape (x, m) is reshaped to a lower dimension space with shape (x, k), where x is the number of training data and m, k being the dimensionalities before and after reduction respectively. Note that for each sub-training dataset generated from the original training data, the remaining data are used as a validation dataset to estimate the performance of the individual model.

4.5 Performance estimation

Fig 4.4 illustrates the system time flow in a more practical way, with the top half showing the time for training without RP and the bottom half with RP. t_0 represents the time for the first element in the dataset to leave the switch. t_{Ia} is the time interval for each packet to arrive at the server. T_t is the total training time taken by the server. Without random projection, training data are simply forwarded to the server without processing, resulting in shorter t_0 and t_{Ia} in the top graph than the ones in the bottom. Packet forwarding during random projection

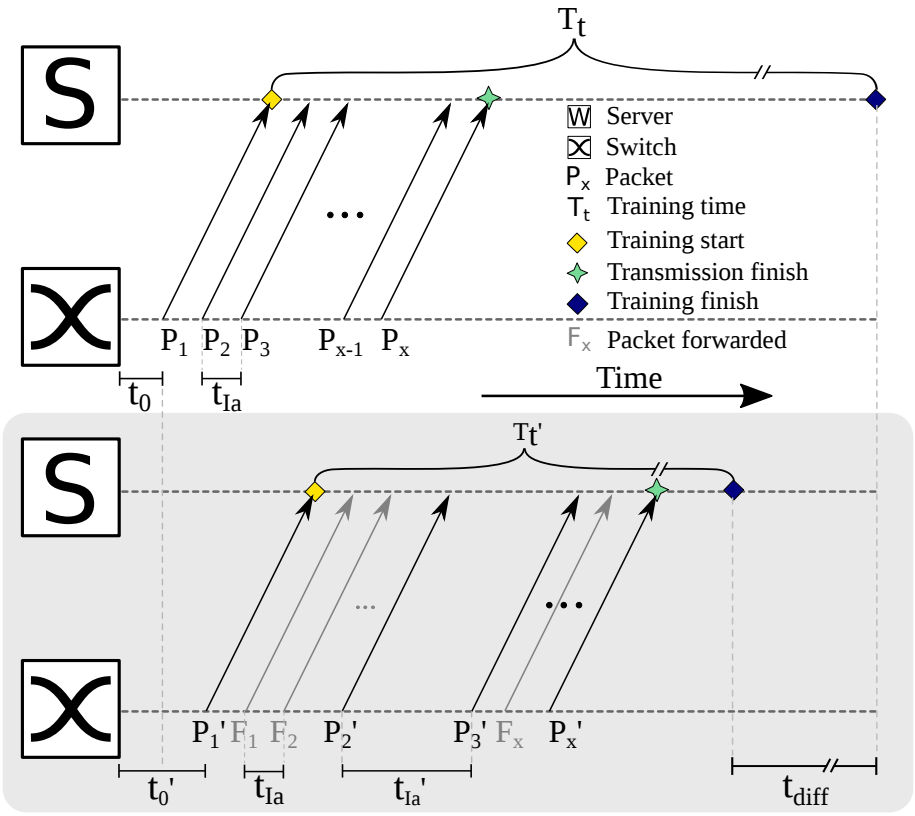


Figure 4.4: Time graph. Top: Forward mode (without RP). Bottom: Processing mode (with RP).

is also shown in the bottom graph, indicated by the grey arrows in between the packets with RP. Note that training can start immediately after *training start*, this applies to Stochastic Gradient Descent (SGD), Gradient Descent (GD) and mini-batch SGD. From the time graph, it is not hard to observe that a speed-up from our design requires t_{diff} to be greater than zero.

To prove our point, four high-dimensionality datasets are tested with their attributes and performance improvements are shown in Table 4.1. Take the realsim dataset for example, 10% of the dataset is picked randomly for our experiment, this translates to 5783 training data. As a proof of concept, we assume a 4x4 10Gbps (256-bit data width) switch is implemented on a Xilinx VU9P FPGA with 6840 DSPs, as presented in [47, 96]. A previous 4x4 network switch designed by the NetFPGA-SUME project [97] shows less than 15% of on-chip memory usage and less than 10% logic usage on a Virtex-7 chip. The same implementation on an Ultrascale+ chip will leave us with more flexibility for the extra RP function.

Only DSPs are considered as multipliers in our analysis at this stage to avoid overestimating

Table 4.1: Estimated result.

Name	Size used	Original dimension	Reduced dimension	Estimated Speed-up
Ad [100]	3279	1558	200	3.2
epsilon [101]	55000	2000	1000	2.1
gisette [98]	6000	5000	1000	6.7
realsim [102]	5783	20958	3500	3.5

the outcome, and to minimise the interference with the switch logic. We assume a 16-bit fixed-point number for data representation as it only requires a single DSP to implement a 16x16 multiplier. Floating point multiplication should be avoided as it requires much more resources on an FPGA. The impact of using fixed-point rather than floating-point is modelled by a conversion precision loss in the simulation. In the future, we will investigate the effect of varying the number of bits in the fix-point representation.

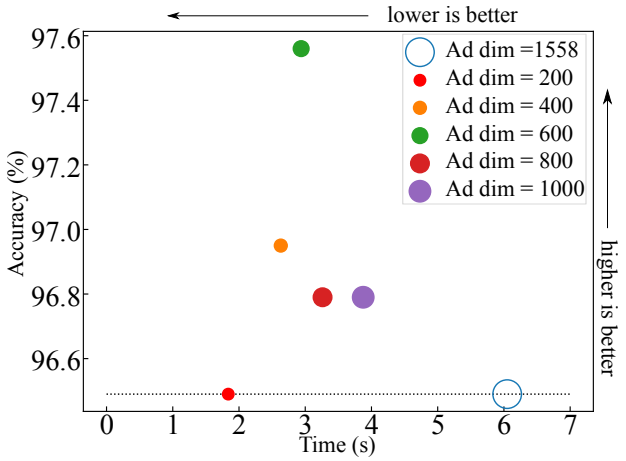
4.5.1 System time modelling

Overall system time is represented by the sum of time spent in the switch (T_{switch}) and the training time of servers (T_t) as shown in Eq. 4.8:

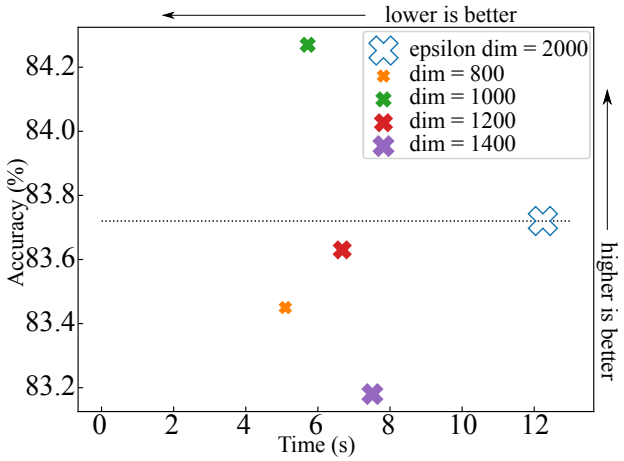
$$\begin{aligned} T_{single} &= T^F_{switch} + T^F_t, \\ T_{ensemble} &= T^P_{switch} + T^P_t \end{aligned} \tag{4.8}$$

where superscripts F and P stand for the forwarding mode and processing mode of the switch respectively. On the switch side, time is measured by the difference between the time the first packet enters the switch and the last packet leaves the switch for both modes. T^P_{switch} in terms of clock cycles can be expressed as:

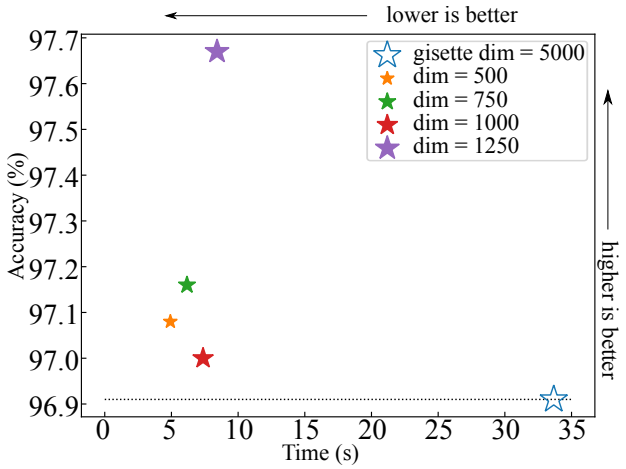
$$T^P_{switch} = \underbrace{t_{in} + t_{mult}}_{t_0} + (X_{sub} - 1) * t_{Ia} + t_{out} \tag{4.9}$$



(a) Ad dataset.



(b) epsilon dataset.



(c) gisette dataset.

Figure 4.5: Experimental results showing the impact of the proposed system on training time and accuracy.

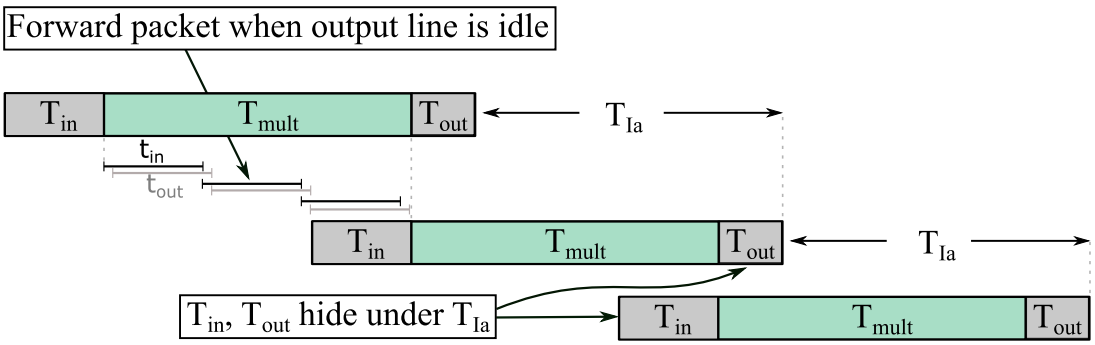


Figure 4.6: Time variable explained.

where X_{sub} is the number of elements in the dataset for each sub-model; t_{in} is the time for the first element to enter the switch t_{out} is the time for the last element to leave the switch, it is shorter than t_{in} due to the dimension reduction; t_{mult} is the time for multiplying a single element of the dataset with the RP matrix. t_{Ia} equals to t_{mult}, as shown in Fig 4.6.

Packet forwarding time for single model training is shorter as no multiplication time is required. The switching latency is typically a few hundred nanoseconds and is a one-time expense for continuous packet transfer, which becomes negligible compared with the total forwarding time. It can be modelled as $T^F_{switch} = X_s * t_{out} = X_s * t_{in}$, where X_s is the number of elements in the single model training dataset.

For worst-case analysis, we assume data flow into the switch as if the data flow through the Ethernet port. This speed could be higher with PCIe and remaining high-speed transceivers. Given the switch data width of D and the input dimensionality of m and output dimensionality of k, we have $t_{in} = 16 * m/D$ and $t_{out} = 16 * k/D$, where 16 is the width of the fixed-point number. The speed-up S of our proposed system compared with a single model can then be calculated by

$$S = \frac{T_{single}}{T_{ensemble}} = \frac{T^F_{switch} + T^F_t}{T^P_{switch} + T^P_t} \tag{4.10}$$

System flexibility

FPGAs can vary in resources and speed so can the servers, and case 3 in the theoretical model will not always hold. It is possible for our system to adapt to different categories of FPGAs and servers [49]. This can be achieved by tuning α, which shifts some portion of RP to the

downstream servers. For servers like IoT or robotics whose computing power is usually low, we expect a higher portion (α close to 1) of RP to be done by the FPGA. For servers like GPU or TPU which are optimised for matrix operations, a smaller portion (α close to 0) of RP should be done by the FPGA. For those that lie in between, like FPGAs or some CPUs, fine-tuning of α is required for best performance.

4.5.2 Case study: Ad dataset training system speed-up

The Ad dataset under testing has a total element of 3279, in which 80% ($X_s = 2623$) is used for single model training and the rest 20% is used for testing. The training time and model accuracy of this single model are then severed as the baseline of the ensemble method. After bootstrapping, sub-training data is sampled for each server. The sub-training data can be less in quantity compared with the training data, and achieves the same model accuracy with less training time. A portion of 80% is chosen for this case study, which makes the sub-training data $X_{sub} = 2098$.

Each element in the Ad dataset has 1558 dimensions. Transferring an element in a 16-bit fixed point over a 10Gbps switch with 256-bit data width requires $t_{in} = 1558 * 16/256 = 97.375 \approx 98$ cycles. Assume the switch is running with a 200MHz clock, and let C denotes its period which is 5ns. We then have $t_{in} = 98C = 0.49us$. This latency only needs to be considered once as matrix multiplication takes longer than receiving a packet. As mentioned in Fig 4.6, t_{in} can be hidden under t_{mult}.

With limited DSP resources on the FPGA, matrix multiplication is broken down into vector multiplication. Each element in the dataset will be multiplied by the RP matrix column-wise. For a system with each FPGA switch connected to two servers, assume only DSPs-based multipliers are used for computation for worst-case analysis. To simplify the calculation, we allocate 1558*2 DSPs to each matrix multiplier module, which results in a total DSP usage of 6232 out of 6840.

As shown in Fig 4.7, the time for the multiplication process depends on the number of DSPs

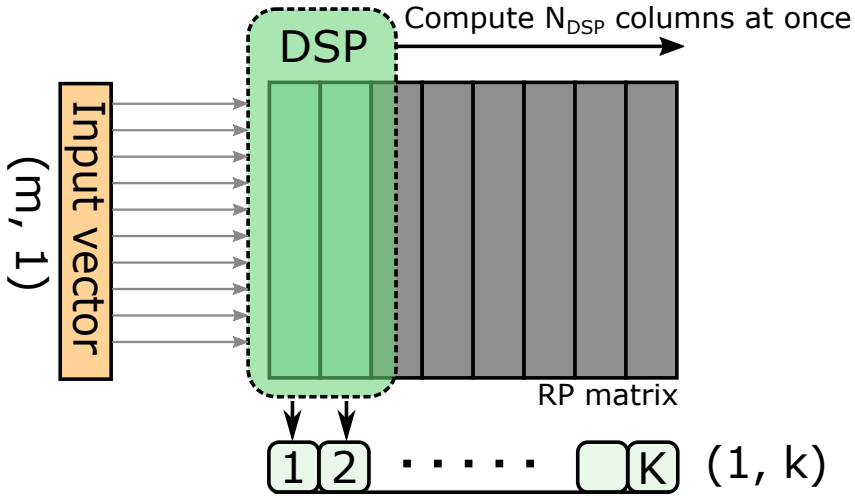


Figure 4.7: Column-wise matrix multiplication.

allocated for multiplication, which can be calculated as $m*k/N_{DSPs}$. Therefore, multiplication time $t_{mult} = 1558/(1558*2*k*C) = 1/2kC$. Multiplication time t_{mult} equals to t_{Ia} in this case as multiplication takes longer than transmission. The remaining elements take $t_{mult} \times (X_{sub} - 1) = 209700C$. On the output side of the switch, the reduced dimension data requires fewer cycles to transfer. After dimension reduction, a vector of 200 dimensions takes $T_{out} = 200 \times 16/256 \approx 13C$ cycles to transfer. In total, $T^P_{switch} = (98 + 209700 + 13)C = 1.049e - 3s$. Finally, the training time for the thin data set is 1.838s measured by the server, in this case, an Intel 7700k CPU. Therefore, $T_{ensemble} = 0.001049 + 1.838 = 1.8390s$.

With the FPGA switch serving as a simple forwarding switch, time spent on the switch simply equals the total number of cycles that the dataset requires to be transferred. Therefore, can be calculated as $X_s * t_{in} = 2623 * 98C = 0.00128s$. Together with a training time of 6.049s obtained from the server, $T_{single} = 0.00128 + 6.049 = 6.0502s$. Substitute the calculated value into Eq. 4.10 and obtain the speed-up $S \approx 3.2$.

4.5.3 Tuning for optimised RP time

Small though compared with training time, RP still takes a significant amount of cycles to finish. To highlight the impact on RP time by tuning α, t_{train} is removed from comparison. Use the Ad dataset again as an example, finding the optimum α requires the knowledge of time

Table 4.2: RP time result with selected α value.

	RP time (s)		α_{opt}
Name	$\alpha = 0$	$\alpha = 1$	
Ad	0.1145	0.0021	0.980
epsilon	0.1867	0.1760	0.515
gisette	0.3686	0.0320	0.920
realsim	0.7012	0.7756	0.475

taken on both the FPGA switch and the server, the optimised alpha can then be found based on their speed. Tab 4.2 summarises the RP time and the optimised α value.

4.5.4 Time estimation for very large dimension size

For datasets with high dimensions, for example, the *realsim* dataset, the m×k RP matrix becomes substantial in size, causing insufficient on-chip memory in small FPGAs. To solve this problem, off-chip DDR memory needs to be added to the system as a method of offloading the on-chip memory usage. However, extra latencies introduced by the DDR memory can cost 100-200 cycles. The data width of the memory controller can be chosen from 32-64bit for DDR4 and 32-512bit for DDR3 [80]. To estimate the worst case, we assume a transfer latency of 200 cycles and a memory interface of 512-bit wide. Moreover, RP matrices are assumed to be stored completely off-chip, despite the spare on-chip memory remaining idle. Note that our precomputed RP matrix is stored in the DDR memory in a compressed manner. Transfer for each column of the RP matrix from DDR to FPGA costs 20958*2/512 = 81.86 ≈ 82 cycles. Assume multiplication finishes as soon as one column of the RP matrix is transferred. In contrast with the case where no DDR is involved, it takes 82 instead of 20958/1000 ≈ 21 cycles to compute the one column of the RP matrix. Taking this into consideration, the new speed-up becomes $S_{DDR} = 1.89$.

4.6 Discussion

In general, our software simulation result shows an increase in training accuracy and a decrease in training time with the ensemble method, the reduced dimension is typically less than half of the original. As proof of concept, several experiments are conducted using our proposed system. A number of 15 members are chosen for the ensemble method, as it achieves stable performance for most tested datasets. Fig 4.5 shows the training time against accuracy for different binary classification datasets. The original data is represented with hollow markers whereas experimental data are represented with solid-filled markers. The more to the right of the x-axis the longer the training time and the higher the y-axis the higher the accuracy. Take the Ad classification dataset for example, our method uses only 12.83% of the original dimension and 30.38% of the original training time to achieve the same accuracy as single-model training. Increasing the dimensionality to 50% extends the training time but results in even higher accuracy than single-model training. For the epsilon data set, the proposed method experiences a small variance in accuracy, but with only less than 0.5% lower in accuracy for some dimensions. This can be explained as the result of an insufficient ensemble member size. The gisette dataset gives the highest time reduction among all the tests. Note that the training time of lower dimension data is measured from the slowest member. Another benefit of our proposed method is the reduction in training data, since the training data for each member is generated using 80% of the training data in the single model.

4.7 Summary

Our approach achieves a higher training accuracy for ensemble learning with reduced training time compared with single-model training. Small datasets are manageable on current FPGAs, whereas for datasets with very large dimensionality, current FPGAs may not have sufficient resources to exploit our method. Our proposed architecture is not limited to a single FPGA-based switch; it can be deployed with multiple FPGA-based switches to form larger ensemble clusters. Moreover, ensemble machine learning is not the only application that can benefit from

our proposed method; further work will explore how our design can benefit other demanding computations, and how such computations can be implemented automatically and efficiently on the switch.

Because training the MLP-based model requires a significant amount of resources and time due to the iterative optimisation method. This chapter focuses on the scenario in which all the training is carried out in the downstream servers. In the next chapter, we approach the classification problem using a kNN model which requires no training. Considering the limited resources on the FPGA, we choose to use hamming distance to replace the cosine distance as the metric. Combining with the random projection, we show that it is possible to fully offload the classification task from the servers, without losing classification accuracy.

Chapter 5

Ensemble Classifier for Nearest Neighbour Algorithm

Chapter 4 proposes an FPGA-based architecture for ensemble learning. In this chapter, we extend the aforementioned architecture and propose CERRNA, a novel FPGA-based ensemble architecture that explores the benefit of the Hamming distance metric for KNN-based classification, this design leverages the remaining two traits mentioned in chapter 4.

Recent years have witnessed a blooming in data-centric applications. k-nearest neighbour search (kNN) as a simple yet effective algorithm has been widely applied to data mining and machine learning applications. Popular distance metrics such as Euclidean and cosine rely on multiplication operations; they are computationally expensive when either the dataset or the dimensions of the dataset are large. Past FPGA-based kNN designs [65] revolve around these distance metrics and their performance is limited by either the memory bandwidth or on-chip memory storage, due to the usage of high bit-width (16 for example) data representation to avoid accuracy degradation.

Hamming distance, which uses only 1 bit to represent each feature in a dataset, has been proven to be able to approximate cosine similarity. As a binary encoding, hardware design could greatly benefit from computing the Hamming distance. For example, compared with a kNN design using a 16-bit data type for storage, using Hamming encoding alone will make

the bandwidth 16 times more efficient. However, little research on it has been studied. One reason could be that converting to Hamming space causes information loss, as a consequence, reduces the accuracy. In this chapter, we present a novel FPGA-based ensemble architecture that explores the benefit of the Hamming distance metric for a kNN-based classifier. Through software and hardware co-design, and with minimum hardware optimisation, we are not only able to gain an estimated speedup of 2.3-8.7 times whilst retaining the prediction accuracy, but also achieve a memory reduction of 3.5-27.7 times, compared with the baseline CPU-based kNN classifier using cosine distance, on a variety of high dimensional datasets. Combining random projection with Hamming encoding, we show it is able to store large datasets (as indicated by the trigger size in Table 5.2) in the onboard memory, making the FPGA solely in charge of the kNN classification, i.e., completely offloading the task from any downstream server in a large-scale system.

5.1 Introduction

With the explosive increase in data, more and more applications are moving their way to the cloud to gain better performance. As a consequence, cloud providers continue to update their infrastructure to accommodate more applications and provide better services. FPGA as a flexible hardware platform has shown great success in the cloud in recent years, from PCIE-connected accelerator cards [20] to smart Network Interface Controllers (NIC) [46]. In-network computing is gaining increasing attention due to its benefits in reducing the response latency of cloud applications, especially in a distributed environment.

FPGA-based network switches due to their customisability can bring multiple benefits to the data centre. For example, support custom transport protocol, in-network acceleration and aggregation [48, 49], network function customisation, etc. Recent study [1] also shows that the gap between FPGA-based network switches and ASIC-based switches is shrinking. The challenges of supporting applications on an FPGA-based network switch are: a) The need for high-performance applications that avoid interfering with the dataflow. b) The available

hardware resources for the application are limited by the specification of the switch, e.g., number of ports, data transferring speed and switching latency, etc. This limits the type of applications that can be implemented on the FPGA-based switch. Although it is possible to implement a part of the application within the switch and rely on the downstream platform to solve the rest of the problem, it introduces extra complexity as it is non-trivial to find the optimal balance of the workload.

kNN has been broadly applied to machine learning and data mining tasks over the past few decades due to its simplicity and outstanding performance. The simplicity comes from the fact that, unlike other supervised learning methods, kNN does not have a training phase. This eliminates potential training issues such as gradient explosion/vanishing in deep neural networks.

The use cases for kNN cover three main areas: data clustering, classification and regression. Additionally, kNN can serve as an important stage in some application-specific machine-learning systems. When building a recommendation system, kNN can be used in the collaborative filtering system to retrieve recommendations based on the similarity of certain user features. It can also be used for data imputation [103, 104], a classic way of dealing with missing values in the training dataset, compressed communication methods for distributed DNN training [105, 106], and accelerating DNN interpretation query [107]. This chapter focuses on classification applications using kNN.

A typical kNN operation consists of two steps: first computing the distance of a query with all the pre-stored training data, then taking the top-k (first k items with the shortest distances). Depending on the types of data, different distance metrics can be used. Pairwise distance Metrics such as Euclidean and cosine distance are the default choices in most cases. However, these metrics are computationally expensive, especially for large datasets; this can mean large in either the length (number of data) or the width (dimension).

While using hamming distance is proven to have similar efficacy as cosine distance and is a hardware-efficient distance metric, its impact on hardware design has rarely been studied, especially for FPGA-based network switches. In this chapter, we propose an FPGA-based

system architecture that combines the power of in-network computing, random projection, ensemble learning and customised hardware design, which allows us to explore the benefit of replacing cosine distance metric with Hamming distance. Through hardware software co-analysis, we address the following challenges. **a)** How to support kNN on an FPGA-based network switch with high performance and minimised resources, as shown in the proposed system architecture. **b)** How to completely offload the kNN tasks from downstream servers, as discussed in section 2.2.1. The contributions of this chapter are as follows:

1. A novel system architecture that enables kNN classification within FPGA-based network switches through an ensemble of classifiers with Hamming distance as the metric (section 5.2). The system can be easily adapted to FPGA-based NICs due to their similar architecture.

2. Achieving high memory reduction on high-dimensional dataset whilst retaining accuracy for kNN classifiers through random projection, random sampling and ensemble learning (section 5.5).

3. A parameterised system model for hardware-software co-analysis that allows prediction and analysis of the system performance, for datasets of different sizes or FPGAs with different resources.

5.2 Proposed System Architecture

With the memory-efficient Hamming encoding method for dataset compression and computation-efficient Hamming distance metric for computing the distance of nearest neighbours, our design enables the possibility of removing the server from the system, having the output database completely stored within the FPGA board. Build upon chapter 4, the distinct features and the challenge they aim to tackle are:

1. In-FPGA top-k computation with sorting.

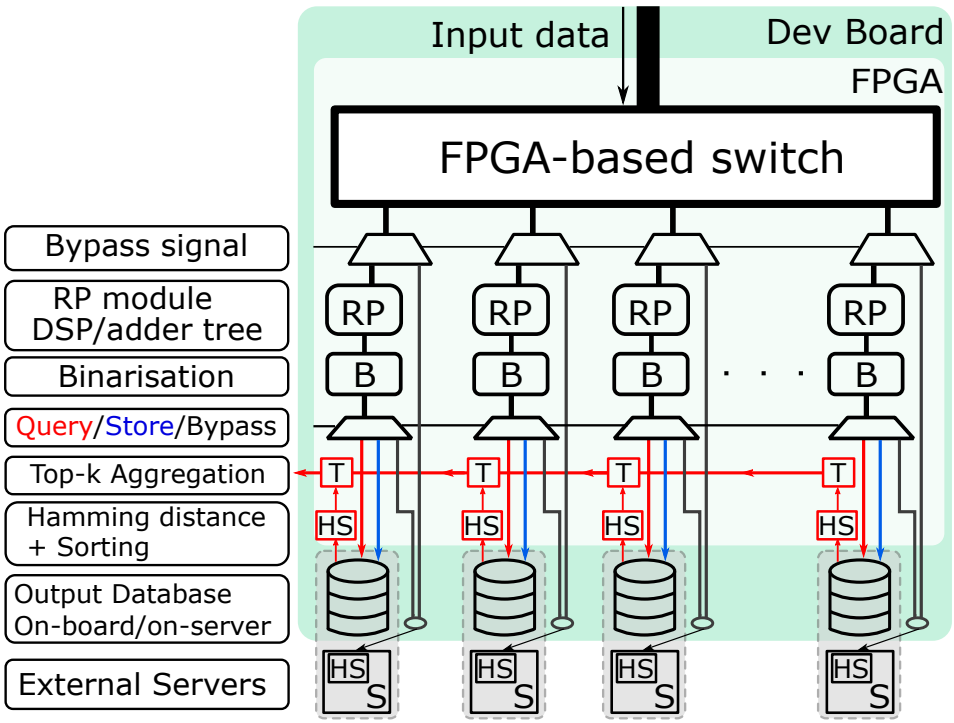


Figure 5.1: System architecture of FPGA-based system, building on top of an FPGA-based network switch. The Hamming distances are computed and sorted in the HS blocks (either on-FPGA or on-server), and T blocks prepare the top-k distances for aggregation.

2. The top-k result is partially aggregated within the FPGA.

3. An additional binarisation step to convert data to Hamming code, this step extracts the sign of each dimension to form a new Boolean vector (challenge a).

4. An additional level of multiplexing that supports the forward, store and query operations (challenge a).

5. The combination of dimension reduction from random projection and the Hamming code extraction gives the switch the ability to store all data within the FPGA board, instead of relying on the server. (challenge b)

6. The multiplier module is not restricted to a DSP-based design, which enables support for different types of RP matrices.

The above features distinguish our approach from other methods which are also based on random projection and ensemble [108].

Algorithm 1 Algorithmic expression of the system flow.

$Group \leftarrow [1...n]$ ▷ group contains n_split servers
$memory \leftarrow \varnothing$
$res \leftarrow \varnothing$
for each member $m \in Group$ **do**
    $data \leftarrow$ incoming data
    $data_{rp} \leftarrow data \cdot MATRIX_{RP}$
    $data_b \leftarrow BINARISE(data_{rp})$
    if *Store* is True **then**
        $memory \leftarrow memory \cup data_b$
    else if *Query* is True **then**
        $distance \leftarrow \varnothing$
        for each data $d \in memory$ **do**
            $idx \leftarrow$ index of data
            $label \leftarrow GET_CLASS(d)$
            $ham_d \leftarrow HAMMING_DISTANCE(data_b, d)$
            $distance \leftarrow distance \cup (idx, ham_d, label)$
        end for
        $top_k \leftarrow$ TOP K ELEMENTS $\in SORT(distance)$
        $res \leftarrow res \cup top_k$
    end if
end for
# Result Aggregation
if $freq_reduce$ is True **then**
    return the class label of the most frequent index
else if $majority_vote$ is True **then**
    return class label of the majority
end if

The proposed system architecture is shown in Fig. 5.1, along with Algorithm 1 detailing the operations in the system flow. In Fig. 5.1, the descriptions of the blocks are presented on the left. In particular, the external servers are downstream platforms that carry out further data processing tasks. In the data centre scope, they are normally referred to as the servers in the rack. Looking from top to bottom, first, data will be fed to the FPGA-based switch from an upper-stream storage source. Then based on the type of data, identified by the header field of the packet, the packet will be either directly forwarded to the corresponding output port or distributed to the computing pipeline through broadcasting. Each copy of the data will enter the RP module to be projected (through matrix multiplication) to a different latent space. Note in our design, based on the type of RP matrix used, the RP module can be implemented with either Digital Signal Processing (DSP) units or simple adder trees, which brings more

flexibility to the resource usage. After the RP process, the binarisation module converts the data to the Hamming space by taking the sign bit of the projected data. This compresses the data from high bit-width representation to binary representation. Since datasets vary in all sizes and shapes, the hardware solution should also be flexible to support such variation. Therefore another stage of bypassing is introduced in our design. This solves the challenge when on-chip/on-board memory is limited in the development board, and allows us to analyse the impact of our method under different circumstances.

If bypass is enabled, the compressed dataset will be forwarded to the downstream server for storage and computation of the nearest neighbours. Turning off the bypass is equivalent to removing the servers from the system, leaving storage and computation all on the development board. For the initialisation phase, i.e., building the database with the store operation, data will be stored in the onboard DDR RAM. For the prediction phase, i.e., query operation, the incoming binarised data will be compared with the database, and the result will be aggregated to form the result. The detail of the aggregation methods is presented in Fig. 5.2.

Let us consider the scalability of the proposed approach. Limited by the manufacturing technology, current FPGA chips remain tight in resources after implementing a high throughput/high port count network switch. This challenges the types of applications that can be implemented on the FPGA-based switch, which potentially cuts out the benefit of in-network computing. To scale the system, a hierarchical design that contains two layers of switching should be used. The extra layer of switching will inevitably introduce additional latency overhead. We argue that this fixed latency is trivial when compared with the computation cost, therefore it is reasonable to adopt this architecture. In fact, when the dataset is large, the problem will be bounded by the computation with current FPGAs. In section 5.5, we quantify this gap with an example dataset.

5.3 Software Modelling

Our baseline design computes kNN using cosine distance for the original dataset, operating on a multi-core CPU server. The CPU time for the baseline is measured on the official KNeighborsClassifier library, which is implemented as a Cython extension that compiles down to C, leveraging OpenMP-based parallelism and allowing multi-core processing on the CPU. We implement our own kNN function with the Hamming distance metric for the experiment for two reasons:

1. The official library implements the Hamming distance with normalisation and returns a double data type, the normalised Hamming distance is the ratio of the Hamming distance to the length of the vector, whereas it is sufficient to use the un-normalised Hamming distance in our experiment. The use of un-normalised Hamming distance simplifies the computation, and only an integer data type is required for storing the result.

2. When the Hamming distance metric is used, little optimisation is applied in the official library, causing an unfair comparison with the cosine distance metric.

Our implementation first generates a distance list by comparing the query with all the data in the database, then returns the top k elements with the smallest distance (largest similarity) after sorting on the distance list.

Fig. 5.3 summarised the software flow. The proposed method uses an ensemble of models to recover the accuracy loss, if any, that is introduced by the Hamming space similarity. All data are first preprocessed, this includes scaling all feature values to the same range, e.g., between zero and one. The impact of using fixed-point rather than floating-point is also modelled by a conversion precision loss function. Then the processed data are projected to multiple latent spaces before being converted to their Hamming encoding. These new sets of data are then stored in separated models waiting for queries to take place, shown as the right-most "Train Data" in Fig. 5.3. The queries will go through the same process before turning into a Hamming vector in their corresponding latent space, shown as the right-most "Test" in Fig. 5.3. Each model in the ensemble will return the top k most similar elements in the dataset.

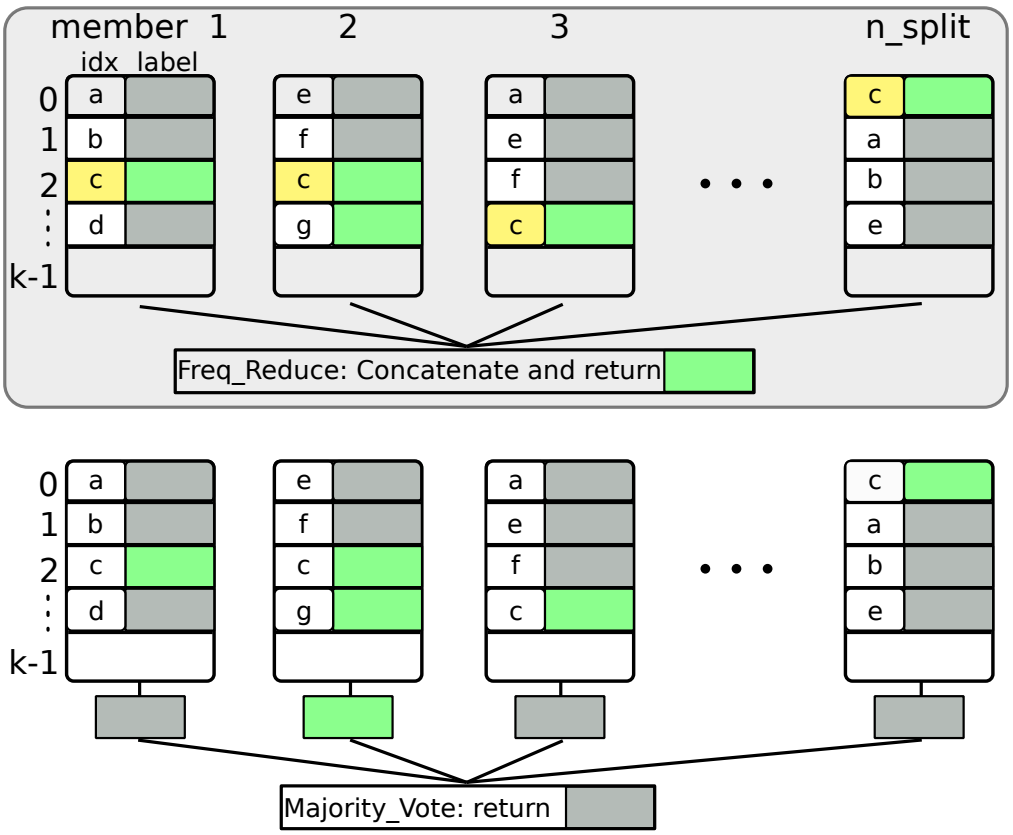


Figure 5.2: Two aggregation methods for computing the class label based on the top-k result of each server.

Based on the aggregation type, each model can return either the label (label in Fig. 5.2) of the majority class, or the index (idx in Fig. 5.2) of the top k element in their subspace. For example, for classification problems, each model could return its predicted label, which is done through a majority vote, followed by another majority vote across all the predicted labels from all models. Each model could also return its top-k indices, then we aggregate these indices and find the most frequency k indices before the majority vote. We will discuss the impact on the result of these two aggregation methods later.

Table 5.1: Details of the 4 types of RP matrices used in the experiment.

RP Matrix type	Multiplier	Sparsity	Data type/value
Gaussian	DSP	Dense	floating-point
GaussianInt	Adder tree	Dense	integer (0, 1, -1)
Sparse	DSP	Sparse	floating-point
SparseInt	Adder tree	Sparse	integer (0, 1, -1)

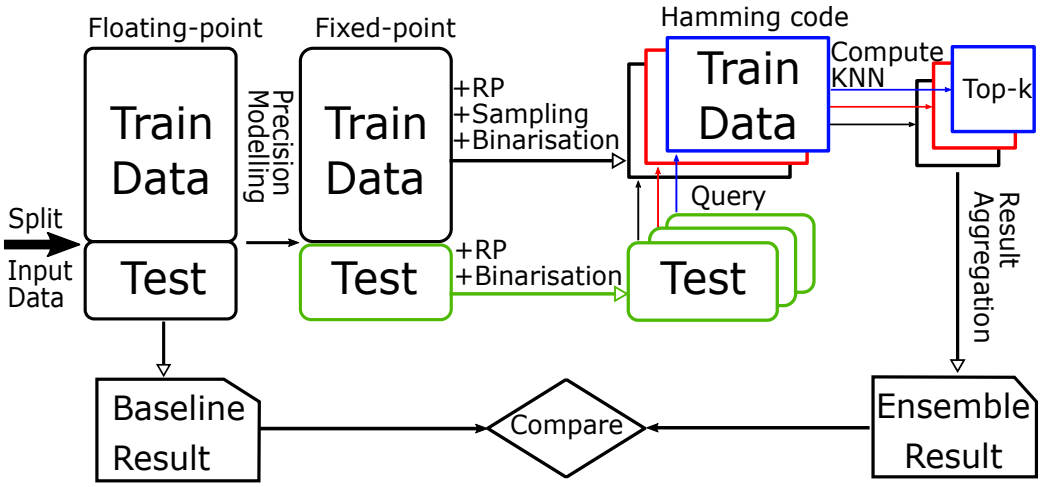


Figure 5.3: Software simulation flow. The ensemble result is based on the aggregation type.

All datasets used are large datasets with high dimensionality, with a mixture of sparse and dense datasets. Due to the lack of dense high-dimensional datasets online, we create two dense datasets. The first one is created from a stock price dataset (Alcoa Corporation 1970-2017), the dataset is created with a window size of 2000 and each window corresponds to the trend of the market for the day after. The choice of window size can be random, we avoid picking dimensions that are too small because dimensionality reduction suits better for high-dimensional datasets. The second dataset contains the image embedding of cats and dogs created using the openAI CLIP model [109], which outputs an embedding length of 512.

As for the experiment setup, we run the experiment for all datasets with 4 types of random projection matrices. Gaussian, GaussianInt, Sparse and SparseInt, which correspond to dense and sparse matrices and their compressed version (using -1, 0, 1 to replace positive, zero, and negative values, identified by the Int suffix). Note that the dense RP matrix is used to compare with the DSP-based multiplier and the compressed version is used to compare with the adder tree-based multiplier. This is summarised in Table 5.1.

For the number of data used, we split the dataset with a fixed train test split ratio value of 0.8. For the ensemble, we provide a further sample portion of the training dataset to reduce the number of data to be stored.

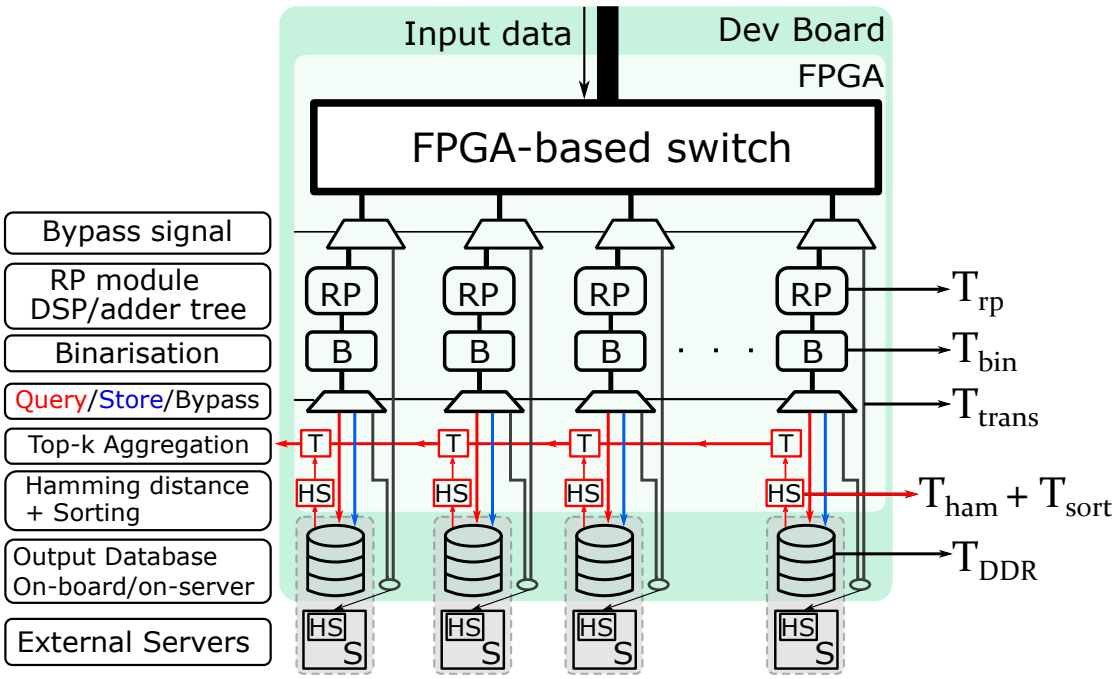


Figure 5.4: System architecture with time notation.

5.4 Performance modelling

The estimation is done using a combination of hardware estimation and software simulation. At a higher level, the time for our system can be summarised as the computation time plus the transmission time. We denote the time taken for the baseline as T_{single}, which represents kNN using the cosine distance metric operating on a single CPU without processing the dataset. The time taken with our proposed design is denoted as $T_{ensemble}$.

The time in the following equations is presented in Fig. 5.4. The time taken for the T block is negligible because it only relates to simple pointer operations. For the baseline model, the breakdown of the time can be expressed as:

$$T_{single} = T_{trans} + T_{cos} + T_{sort} \tag{5.1}$$

where T_{trans} is the time taken to forward the data from the data server to the server, T_{cos} is the time taken for the CPU to compute the cosine distance, T_{sort} for sorting the obtained distance table. For worst-case analysis, we assume no overlap between these operations, theoretically, sorting can start immediately after receiving a single data point if we maintain a priority queue,

similar to a streaming top-k algorithm.

For the proposed model, the system time bifurcates into two expressions depending on the size of the dataset. They are:

$$T_{ensemble} = T_{rp}^{fpga} + T_{bin}^{fpga} + T_{trans}^{fpga} + T_{DDR}^{fpga} + T_{ham}^{fpga} + T_{sort}^{fpga} \tag{5.2}$$

if the data size is smaller than the onboard memory, otherwise,

$$T_{ensemble} = T_{rp}^{fpga} + T_{bin}^{fpga} + T_{trans}^{fpga} + T_{ham} + T_{sort} \tag{5.3}$$

where T_{rp} is the time taken for random projection, T_{bin} is the time taken for binarising the projected data, T_{DDR} is the time for reading and writing data from the on-board DDR memory and T_{ham} represents the time taken to compute the Hamming distance between the query and the dataset. Note that parameters with superscript represent operations that happen on the FPGA, whereas those without represent operations on the CPU. If the size of the database after compression can fit in onboard memory, then the top-k is performed on the FPGA, otherwise, performing top-k on the CPU server. For simplicity, we skip the case that would store the data partially on the FPGA and partially on the servers, as this would require non-trivial algorithmic adaptation.

For worst case analysis, we assume only two columns of multipliers are implemented on the FPGA, i.e., generating two output dimensions at the same cycle. This reduces the execution time of the multiplier by half. Now $T_{rp} = M{*}m/N{*}C$, with M being the original dimensionality and m being the projected dimensionality. Note that N represents the number of multiplication units, which is the same for the adder tree and the DSP multiplier.

$T_{ensemble}$ contains two phases, namely the Store and Query phases. They can be calculated with:

$$T_{store} = sr \times sp \times n_{data} \times t_{rp} \tag{5.4}$$

$$T_{query} = n_{data} \times (1 - sr) \times (T_{rp}^{fpga} + T_{ham}^{*} + T_{sort}^{*}) + T_{init} \tag{5.5}$$

where n_{data} is the total data size, T_{init} is the initial latency of the sorter pipeline, sr is the train test split ratio and sp is the sample portion. Note that the sample portion only applies to the training data. The superscript $*$ indicates the time taken in either a CPU or an FPGA. We will elaborate on Eq. 5.4 and 5.5 with an example dataset in the next section.

The platforms used for our experiment are:

1. Xilinx U280 board [80] with a total DDR capacity of 32GB and 9027 DSP slices.
2. AMD EPYC 7543 32-Core Processor running at an average 2.8GHz, with 504GB total memory.

Because the bypass signal separates the packet forwarding and data computing path, different clock domains can be applied to the FPGA-based switch and the computing logic. This allows switch designs that run at a lower frequency (e.g., SMiSLIP and GCQ) to be used. For switch designs that run at a higher frequency (e.g. Hipernetch), only one clock domain is needed. For simplicity, we assume the FPGA runs at 200 MHz, which is equivalent to a clock period of 5ns.

The following section presents the result of the performance model evaluated on several high-dimensional datasets.

5.5 Dataset-specific case study

The epsilon dataset [101] is a dense dataset with high dimensionality, the original dataset contains 400000 entries with 2000 dimensions. To reduce the simulation time, a portion of the dataset is taken for simulation. In this case study, 7000 data entries are randomly sampled from the dataset for training and testing, and a train-test split portion of 0.8 is used.

To calculate T_{single}: According to Eq. 5.1, the transmission time is simply the time spent forwarding the packet by the switch. This time is determined by the design of the network switch and the length of the packet. For simplicity, we assume that the packet is of fixed size, and the network switch operates the same as the NetFPGA-SUME [97] base design, but to

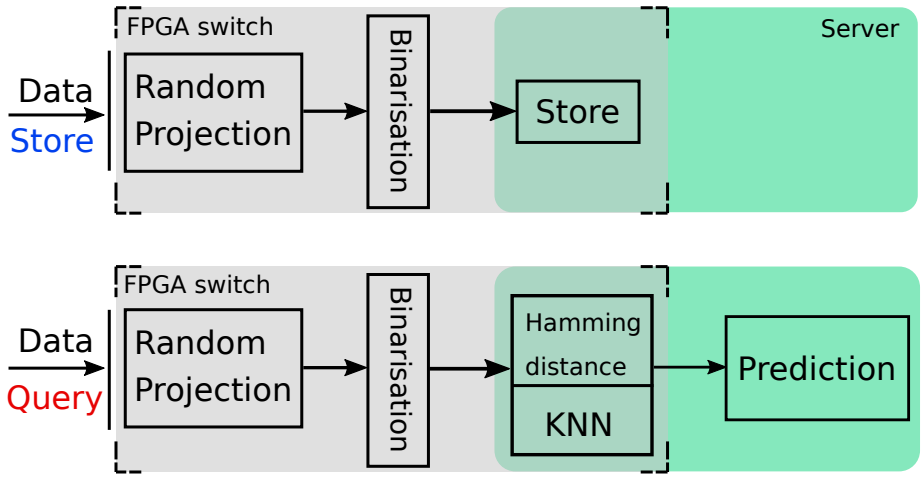


Figure 5.5: Detailed kNN flow for a single copy of the data in the ensemble, with Store phase (building the database) on the top and Query phase (computing kNN) on the bottom.

match the state-of-the-art FPGA, we assume it is implemented on a U280. Based on [108], T_{trans} contributes little to the total time due to the domination of the RP process. Another reason to ignore this parameter is that both the base case and our proposed design contain packet forwarding, reducing their impact on the speed-up which is calculated by the ratio of the two cases. Therefore, T_{single} is reduced to $T_{cos} + T_{sort}$. This gives us $T_{single} = 0.29270$s. Note that T_{single} includes the time taken for the fitting process, this is to ensure a fair comparison with the $T_{ensemble}$.

To calculate $T_{ensemble}$: As stated in Eq. 5.2 and 5.3, there are two conditions for $T_{ensemble}$, which can be further divided into two phases, T_{store} and T_{query}, both sharing the random projection and binarisation processes as illustrated in Fig. 5.5.

Table 5.2: Experiment results

Name	# entry	M	m	sp	Lib Time CPU Cosine/s	Estimated time/s FPGA	FPGA speedup	Total memory reduction	Trigger size Terabytes
epsilon [101]	7000	2000	1000	0.9	0.2927	0.095917	3.0	3.9	1.138
catsndogs	10000	512	100	0.6	0.3065	0.103337	2.9	21.0	4.369
AD [110]	3279	1558	200	0.6	0.1111	0.012718	8.7	27.7	6.647
stock	10074	2000	1000	0.8	0.4916	0.151872	3.2	26.6	1.280
realsim [102]	3614	20958	8000	0.8	1.5200	0.652816	2.3	6.9	1.677
gisette [111]	6000	5000	1000	0.8	0.5027	0.078037	6.4	22.2	3.200
REJAFDA [112]	1996	6824	700	0.6	0.2153	0.012023	17.9	519.9	8.318
qsar_oral [113]	8992	1024	800	0.8	0.3953	0.111949	3.5	12.8	0.819

Condition 1 — when the on-board memory can hold the dataset. In this case, there won't be

packet transmissions between the FPGA and the servers, removing one level of data transmission. The Store and Query in Fig. 5.5 will both run on the FPGA-based switch. However, we do need to consider the DDR RAM latency for storing and reading. For worst-case analysis, we assume the read and write latency takes 200 cycles = 1 us. Since it is smaller than T_{rp}, it can be hidden within the 2.5us window. Therefore T_{DDR}^{fpga} can be replaced by the initial latency of 1us, assuming read and write have the same latency.

For the Store phase, with the random projection being a time-consuming part of the design and to support different RP matrices, we design our system to choose between two types of matrix multiplication modules. A DSP-based multiplier for the RP matrices with large fixed-point values, or an adder tree-based multiplier for RP matrices contains only 0, -1 and 1. To avoid overutilising the resources, we assume that each switch will implement only two copies of the RP module. For the DSP-based RP module, we allocate 2000 DSPs for each module. Then two copies of the RP module give 4000 DSPs, which is less than half of the total DSPs (9024) on the U280 board.

To align with the DSP-based RP module, we also implement an adder tree-based RP module for GaussianInt type RP matrix as shown in Table 5.1, with 2000 input of type 16-bit fixed-point using Vivado HLS. Because all input data are normalised between 0 to 1, a 16-bit fixed-point data type is enough to avoid overflow. In terms of resource utilisation, the adder tree-based RP module only uses 5% LUT on the U280 board. Using FPGA to binarise a vector can be simply done by extracting the sign bit of each element in the vector, this is included in our adder tree design. The overhead of this operation is trivial, thus adding this functionality to the DSP-based multiplier will not affect the estimation outcome. Therefore, T_{rp} = 0.5*1000*5ns = 2.5us, meaning the projected data will be generated every 2.5us. The total time to generate the entire projected dataset in the store phase is T_{store} = 0.9 * 0.8 * 7000 * 2.5us = 12.60 ms.

In the Query Phase, the query data is first projected to the same latent space using the database, which takes 2.5us. Then this projected query will be compared with the entire database, computing the Hamming distance along the way; Burst reading from the DRAM can be initiated within this 2.5us window, and by the time the projection is ready, the entire database would

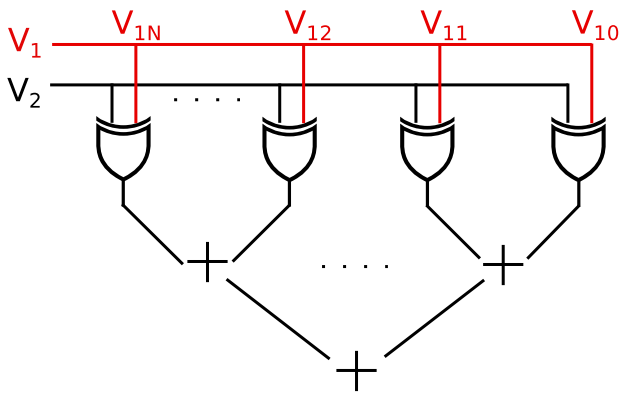
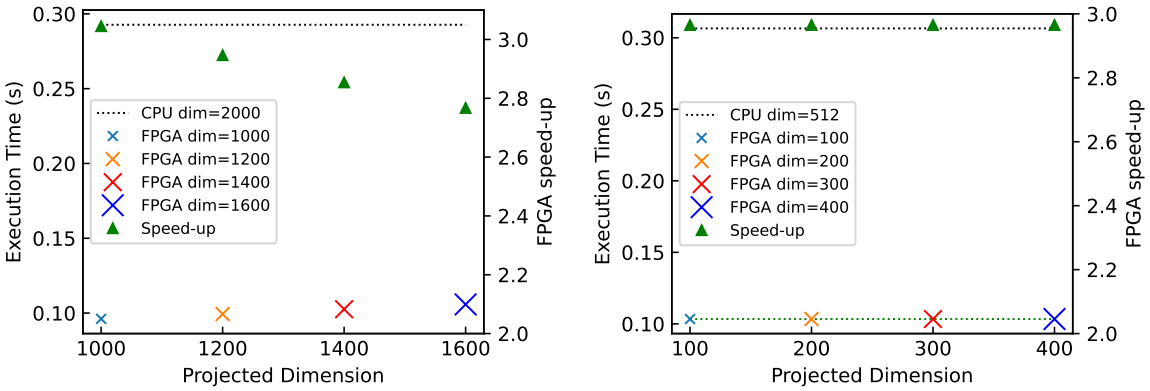


Figure 5.6: A simple circuit design for computing the Hamming distance between two binary vectors V1 and V2, which are extracted for the sign of the previous stage.

be ready to use. To compute the Hamming distance, another adder tree is implemented. This adder tree only takes as input a single bit, therefore significantly reducing resource usage. This circuit is much simpler than the adder tree for random projection, and our implementation result shows less than one per cent of LUT is used on the target platform.

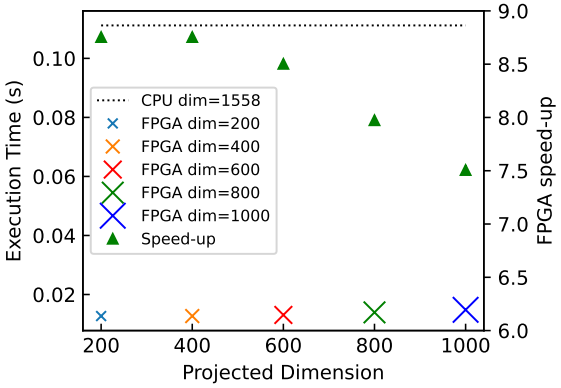
If pipelined, the initial latency can be calculated as $log_2 N$, where N is the length of the input bit vector. For an input of 1000 bits, the initial latency is around 11 cycles. The initial latency can be amortised with a large input dataset. For the sake of simplicity, we will ignore this latency in our estimation as the datasets used in our experiments are relatively large. When reading the data from DRAM, with a reading port data width of 1024, it is possible to read one data from the DRAM every cycle. Therefore, computing the Hamming distance of a single vector can be achieved every cycle, and the total time is taken n_{data} cycles. For the epsilon dataset, this is $T_{ham}^{fpga} = 0.9 * 0.8 * 7000 * 5\text{ns} = 25.2\text{us}$.

Finally, for the sorting time, we need to sort an integer vector of length n_{data}. We implement an iterative merge sorter using Vivado HLS with 6300 inputs using a short data type (signed 16-bit), the latency result is shown in Table 5.3. Since the size of each input is bounded by the number of elements in the input vector, this allows further optimisation of the input data width of the design. We use this simple design as a proof of concept, since designing a high-performance sorter is not the main object of this chapter. We are aware that better designs for retrieving top-k exist, this design serves as the upper bound of taking the top-k element. The implementation result shows little resource usage. In fact, it is designed to return the result in

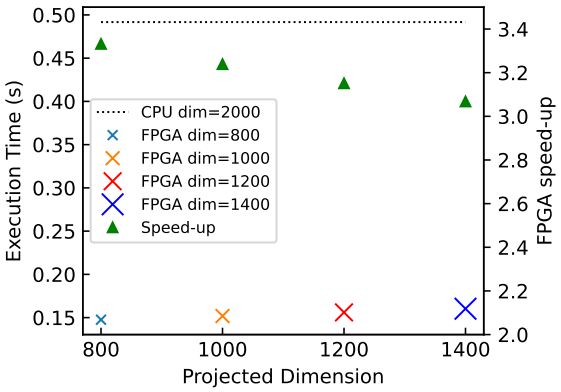


(a) epsilon

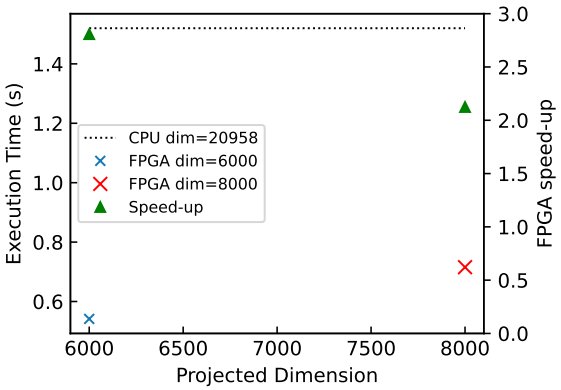
(b) catsndog



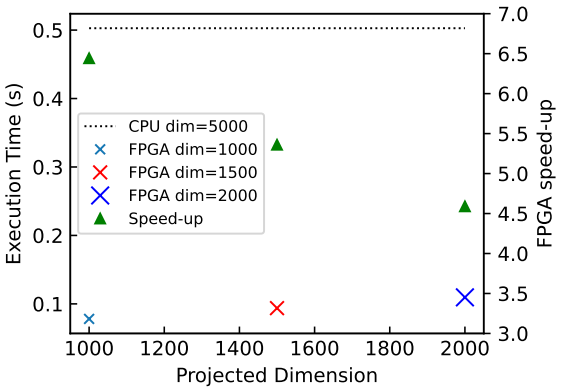
(c) AD



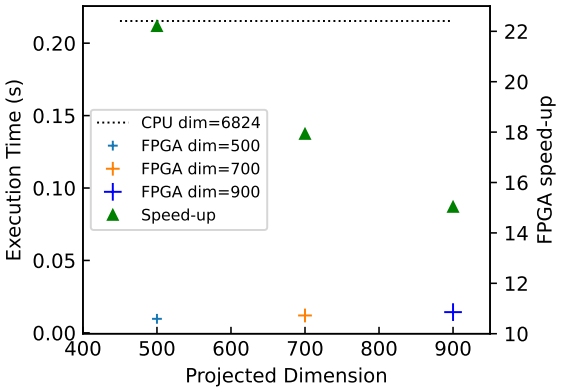
(d) stocks



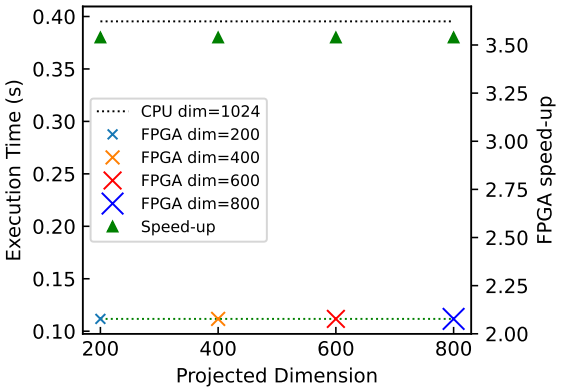
(e) realsim



(f) gisette



(g) REJAFDA



(h) qsar oral toxicity

Figure 5.7: FPGA speed-up versus projected dimensionality.

Table 5.3: Sorter HLS implementation latency result.

latency (cycles)	Latency (absolute)	Interval (cycles)	Type
81951	0.410ms	6304	dataflow

Table 5.4: Resource usage for Xilinx U280 Alveo.

Resource	LUT	BRAM	FF	DSP
Total	1303680	4032	2607360	9024
Used	7421	192	3755	0
Used%	0.57%	4.76%	0.14%	0.00%

$log_2(N)$ stages, so the output latency is deterministic given the running frequency.

In practice, our pipelined implementation with minimum optimisation (array partition) using Vivado HLS has an initial latency of 0.410ms and an interval latency of 6304 cycles = 31.52 us, as shown in Table 5.3. In theory, the interval latency (N) is the latency of a single stage and the initial latency ($Nlog_2(N)$) is the total length of the pipeline and the latency. The implementation summary shows the actual latency corresponds to the theoretical value. The resource usage is shown in Table 5.4. Putting everything together, the total time taken by test data is T_{query} = 7000 * (1 - 0.8) * (2.5us + 25.2us + 31.52us) + 0.410ms = 83.318ms Adding the time for the two phases gives the $T_{ensemble}$ = 12.6 ms + 83.318 = 95.917 ms

Condition 2 — when all data are stored in the server. Under this condition, the speed-up will be dominated by the CPU computation time. KNeighborClassifier uses SciPy's hamming distance which returns a normalised distance with type double. This does not fit our design purpose which requires only a small non-normalised integer distance as the output type. As a result, we implement our own version of kNN that can take our own hamming distance function as input. Re-running the experiments on our implementation and the speed-up under this condition is 7.5.

Although this speed-up appears to be higher than that in Condition 1, it does not reflect the real behaviour of our design. This is because our own implementation does not leverage parallel computing on the CPU. The problem then becomes memory bound, meaning the time difference

is dominated by the memory retrieval time of a full-precision dataset and a hamming encoded one, which outweighs the benefit of computing RP on the FPGA. Getting the same comparison of Condition 1 requires the exact same software implementation of the Scikit learning library, which is beyond the scope of this study. Fortunately, it requires the dataset to be massive to trigger Condition 2; this is presented in the last column of Table 5.2. The computation detail is shown in section 5.6.2.

With respect to classification accuracy, our simulation results show that by configuring the experiment with [projection dimension=1000, sample portion=0.9, number of split=18], we could recover the accuracy loss caused by the RP and binarisation process. In terms of memory usage, depending on the data type of the original dataset, there is a fixed compression ratio W using the hamming space i.e., the bit width of the data type to 1. For example, W=32 for the epsilon dataset that uses floating-point numbers. The total memory reduction (the ratio of the original memory size to the compressed memory size) can be calculated as:

$$R = \frac{M}{m} \times \frac{W}{n_split \times sp} \tag{5.6}$$

where $\frac{M}{m}$ is the ratio between the original dimension and the projected dimension, sp is the sample portion, and n_{split} is the size of the ensemble to recover the accuracy.

Quantifying the gap between forwarding latency and computation latency

Based on the official NetFPGA-SUME wiki [114], the average port-to-port latency of the reference switch is 823ns. This corresponds to T_{trans}^{fpga} in Eq. 5.2 and Eq. 5.3. For the epsilon dataset, T_{rp}^{fpga} takes more than 3 times longer than T_{trans}^{fpga}. As long as the total transmission latency is shorter than T_{rp}^{fpga}, it can be hidden under the random projection latency, therefore, we could keep increasing the levels of the hierarchy until T_{trans}^{fpga} saturates T_{rp}^{fpga}. In addition, if a certain level of buffering is enabled in the hierarchy, then it is able to control the packet arriving interval by sending packets back-to-back, with a controlled burst interval. As long as the burst interval is smaller than T_{rp}^{fpga}, then it can also be hidden under T_{rp}^{fpga}. Therefore we

only need to add the initial latency to the whole model, which is 823ns times the number of levels in the hierarchy. Compared to $T_{ensemble}$ which is in millisecond scale, this is still small.

5.6 Result and Discussion

Table 5.2 summarises the experiment results for all the datasets; it also contains all the information required to compute the speed-up and the memory reduction. The # entry column represents the total number of data instances used to compute kNN. Column *sp* is the minimum sample portion to successfully recover the accuracy given the ensemble size. The data shows that our FPGA-based design offers speed-up for datasets of different sizes and dimensionality. Notice that for the REJAFDA dataset, the speed-up is around 17 times, this is due to the high sparsity of the dataset, which results in a very high M/m ratio. The trigger size is the maximum amount of data that can be stored on the FPGA, this is detailed in section 5.6.2. To generalise our performance model, we have created a function that takes as input the first four columns of the table and returns the estimated FPGA execution time.

5.6.1 The relationship between speed-up and projected dimension

Fig. 5.7 presents the speed-up under different projected dimensions. The dotted line at the top of each figure represents the based line execution time for the original dataset. The markers at the bottom of each figure represent the execution time of our system with different projected dimensionalities (the lower the better). The triangle markers in the middle of each figure show the speed-up for each projected dimension. The speed-up various due the difference in data size ($\#entry$), original dimension (M) and the projected dimension (m) as shown in Table 5.2. In general, the speed-up decreases linearly with the projected dimension. There are a few exceptions such as Fig. 5.7b and 5.7c, where a constant speed-up for the FPGA can be observed. This is because the projected dimension is smaller than the input size of the matrix multiplier. With the current design, all dimensions smaller than the input threshold will experience a constant speed-up. In terms of prediction accuracy, all the presented dimensions are able to

achieve the same accuracy as the baseline. If speed-up is vital, it is possible to further reduce the project dimension to gain higher speed-up, with the sacrifice of the prediction accuracy.

5.6.2 The effect of memory reduction on system scalability

Eq. 5.6 presents the memory reduction ratio after summing all copies of data stored in all members of the ensemble. Multiply Eq.5.6 by n_split gives the memory reduction ratio (R_s) for a single copy of the dataset. Take the epsilon dataset which utilises 18 splits as an example. $R_s = 18 * 3.95 = 71.11$, which means for a single FPGA board, we will be able to store 71.11 times more data. If we only consider off-chip memory for storage, for an FPGA board with 32GB available DDR RAM, based on the assumption that each FPGA board holds two copies of the dataset, which means 16GB for each copy. With the memory reduction, we are able to store data with sizes up to 1.138TB on a single FPGA node, which is enough for most available datasets online. We define this maximum data size to be stored on the FPGA as the trigger size, shown in the last column of Table 5.2.

5.7 Quantitative performance comparison

Compared with previous designs, our system supports a much higher input dimensionality. The comparison is summarised in Table 5.5. In particular, [115] proposed a parameterised kNN IP core that supports either a large number of training instances or a large feature dimension. However, achieving high in one aspect inevitably brings down the other. For example, the IP core theoretically supports dimensions up to 16384, but for only 15 data entries.

Table 5.5: Comparison with other designs.

Designs	Platform	Focus	Metrics	Input Data dimension
[116]	AP	memory bandwidth, throughput	Euclidean	unspecified
[71]	AP	optimised circuit for Hamming distance	Hamming	64 - 256
CHIP-kNN [63]	Cloud FPGA	memory bandwidth optimisation	Euclidean	2 - 128
PRINS [117]	in-storage	in-situ processing architecture design	Euclidean	1 - 384
kNN-MSDF [118]	FPGA SoC	parallel MSDF	Euclidean	4 - 17
[119]	FPGA SoC	multi-index hashing circuit design	Hamming	64
Proposed	FPGA network switch	in network computing with random projection and ensemble	Hamming	<20958
[115]	FPGA	intensive block pipelining	Manhattan	16384
Rosetta [120]	FPGA	benchmark	Hamming	unspecified
[121]	FPGA	ANN with distance-based hashing	Hamming	<24
kNN-STUFF [68]	FPGA	parallel accelerator design	Euclidean	<50
[65]	FPGA	modified bubble sort for optimisation	Euclidean	64
[66]	FPGA	multi-core ensemble KNN	Euclidean	1 - 16
[67]	FPGA	PCA with low precision feature	Euclidean	16
[122]	FPGA	LSH-based kNN for traffic classification	Hamming	<50

5.8 Summary

By combining random projection and ensemble techniques for a binarised Hamming embedding database. Our system is able to host the kNN classification entirely within an FPGA-based platform equipped with customised network switches and gain promising speedup through software hardware co-analysis. This indicates our system is able to completely offload kNN applications from downstream servers, which preserves the server resource for other applications.

Chapter 6

Conclusion

This PhD thesis starts with a study on the feasibility of implementing network switches on the FPGA, followed by the applications that can benefit from the FPGA-based switch design. At the beginning, we study three FPGA-based switch designs namely, SMiSLIP (a memory-efficient iSLIP-based switch design), GCQ (a hierarchical switch design) and Hipernetch (a high throughput switch design). We quantify the gap between the commercial ASIC-based network switches and show that modern FPGAs are capable of providing competitive performance in terms of link capacity and latency; this is supported by showing that the internal bandwidth of our FPGA-based design is higher than the combined transceiver (same as ASIC-based switches) bandwidth on the FPGA, and the achieved port-to-port latency of the FPGA designs. However, there is still a significant gap in the aggregated throughput, leading to a smaller number of ports with the same link capacity. As shown in section 2.2.1, the high-end switches support a bandwidth of 51.2Tbps, and the low-end switches support a bandwidth of 1.04Tbps. With the highest bandwidth of the FPGA-based switch design being 3.4Tbps (1.7Tbs $\times$ 2 as reported in Hipernetch [1]).

The main cause of the gap is the available resources on the current FPGA chips. The routing and buffering resources limit the scalability of the switch fabric. The applications that make use of the remaining resources on the FPGA are also affected by those limitations. For example, in Chapter 4 and 5, the limited computational resources prevent us from implementing a fully

pipelined version of the random projection module. Despite the speed gained from the current design, better performance could be obtained.

Overall, the current FPGA-based network switch can be a good bump-in-the-wire solution that replaces low-end ASIC-based switches. In terms of packet switching, FPGA-based switches due to their customisability, can match the performance of high-end ASIC-based switches in either number of ports, latency or link capacity. When combining these factors, current FPGAs are more likely to match low-end data centre switches as mentioned in Chapter 2, which makes them suitable for edge data centres.

In addition to packet switching, we show that FPGA-based switches offer customised applications which is not supported by current ASIC-based switches. We identify the traits for applications that suit the FPGA-based switch and propose two novel architectures that facilitate machine learning applications. This gives the FPGA the advantage of off-loading tasks from the downstream servers without sacrificing the performance. Although the gap between the FPGA-based switch and the ASIC-based switch exists, the benefits of a programmable switch are still notable.

With the advancement of the FPGA, for example, hardening the switching fabric or 3D stacking, we foresee FPGA-based switches achieving their full potential and closing the gap between the ASIC-based switch in terms of packet switching.

6.1 Summary of Thesis Contributions

This section summarises the contributions of this thesis and the challenges addressed by them. The challenges are: **(C1)** Matching performance with ASIC-based switches, **(C2)** Resource limitation and application design and **(C3)** Problem size and load distribution.

This thesis first conducts a feasibility study that reveals the challenges in FPGA-based switch design and quantifies the gap against ASIC-based switches. To tackle **C1**, A memory-efficient version of iSLIP design named SMiSLIP targeting the reduction of on-chip memory resource

usage is proposed. Compared with the naive input buffered iSLIP design, the proposed design increases the number of ports from 32 to 64 at 10Gbps link capacity, improving the scalability requirement. We then proposed a novel FPGA-switch based architecture for ensemble learning that utilises spare computational resources on the FPGA, this architecture achieves speed-ups in training ensemble models whilst maintaining high accuracy, aiming to address **C2**. Finally, we propose another novel architecture that enables fast kNN classification using an FPGA-based network switch. The design achieves high memory reduction on big datasets, leading to a more efficient onboard memory utilisation. As a result, we show that kNN classification can be completely offloaded to the switch, saving computing power for downstream servers. This addresses **C3**.

6.2 Future Work

To further overcome challenge **C1**, future work could focus on the exploration of more multi-stage fabric designs such as [29,94] on FPGA since only a few switch architectures and scheduling algorithms were implemented. Provided that a high-speed low-latency chip-to-chip communication method is discovered in the future, it is also worth exploring multi-chip based FPGA switches. In addition, improving the current design could also help tackle the challenge. For example, the current SMiSLIP is implemented without pipelining and therefore runs at a low frequency; it is worthwhile developing a pipelined version of the iSLIP scheduler to gain a higher clock rate. With a higher running frequency, it is possible to achieve higher throughput.

Due to the limited amount of resources on the FPGA, implementing applications on the FPGA-based network switch will inevitably compete for resources. One possible future work is to exploit the reconfigurability of FPGA technology to adapt the switch design to meet the resource requirements for different applications. This would help tackle challenge **C2**. Another angle to look at is to place ASIC-based hardware modules for accelerating applications on the FPGA and leave all the programmable logic for the FPGA-based switch design. Also, future work could involve automating the search for optimal performance, given the application performance and

the switches' performance.

As for challenge **C3**, a question might arise: could applications be implemented in the SmartNIC instead of the FPGA-based switch, as there could be more resources left in the SmartNIC? To answer this, we need to analyse the application requirements. If the implemented acceleration task takes the same time for the switch and the smartNIC to finish, then it is worth looking at the data size and the response time. If the result data eventually enters the server and needs to be further processed, then a method such as compression could reduce the data transmission time after the switch (FPGA $\rightarrow$ NIC $\rightarrow$ server). If the application terminates at the FPGA switch or the NIC, then operating at the switch will give a better response time since it is closer to the request.

However, the situation changes if the resources on the FPGA switch are not enough, resulting in a longer running time. Then depending on how much it is slower, careful planning should be made to decide the amount of workload to be offloaded to the NIC or even the server. This planning is tightly related to the application and may require new algorithms to enable collaborative work between different machines. Two possible future directions might help alleviate this challenge:

1. Harden the switch fabric on the FPGA and keep the programmable logic separated from it. This would provide a higher operating frequency of the switch, and the speed-up in the switching fabric gives opportunities for many stable-scheduling algorithms, which could not be efficiently implemented with current FPGAs. Moreover, there will be less resource contention between the applications and the switch.

2. Use machine learning to learn the offloading settings of different types of traffic and enable the change of this setting through run-time reconfiguration, provided that reconfiguration time is acceptable.

For the proposed system architecture for kNN classification, applying optimisation to the hardware design could result in a higher speed-up. In addition, the kNN architecture can also be

used in machine learning model compression using top-k sparsification [106], i.e. only sending the most significant entries of the stochastic gradient.

6.3 Final Thoughts

One scenario for an FPGA-based switch to completely replace an ASIC-based switch is when the growth rate of data is slower than Moore's law, leading to excessive hardware power that surpasses the data traffic size. However, in practice, we are currently experiencing an explosion in data, especially in the era of machine learning. Therefore, it is unlikely that the data growth rate will be slower than Moore's law, which means ASIC-based designs would always perform better than their FPGA counterparts for the same functionality due to the inherent tradeoff between performance and flexibility in FPGAs. As a result, it is unlikely that an FPGA-based switch to replace a top-tier ASIC-based switch in the near future.

To narrow the gap and maintain programmability at the same time, a viable solution is to produce FPGAs with hardened (ASIC-based) switch capability in the FPGA fabric. These devices would become examples of domain-specific FPGAs: they combine high-performance switching capability and acceleration for demanding applications.

Bibliography

[1] P. Papaphilippou, J. Meng, N. Gebara, and W. Luk, "Hipernetch: High-performance FPGA network switch," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 1, pp. 1–31, 2021.

[2] Z. Dai and J. Zhu, "Saturating the transceiver bandwidth: Switch fabric design on FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2012, pp. 67–76.

[3] A. P. Engbersen, "Prizma Switch Technology," *IBM Journal of Research and Development*, vol. 47, no. 2.3, pp. 195–209, 2003.

[4] Mellanox, "SB7800 InfiniBand EDR 100Gb/s Switch System," 2020. [Online]. Available: https://www.mellanox.com/sites/default/files/doc-2020/pb-sb7800.pdf

[5] ——, "Qm8700 mellanox quantum™ hdr edge switch," 2020. [Online]. Available: https://www.mellanox.com/sites/default/files/doc-2020/pb-qm8700.pdf

[6] Cisco, "Cisco Nexus 3000 Series Switches: Compare Models," 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/switches/nexus-3000-series-switches/models-comparison.html#~tab-nexus3500

[7] ——, "Cisco," 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/switches/nexus-3000-series-switches/models-comparison.html#~tab-nexus3200

[8] F. Ltd., "N8500-48b6c 48-port 25gb sfp28 l3 trident 3 data centre managed ethernet switch," 2020. [Online]. Available: https://www.fs.com/uk/products/75807.html

[9] J. networks, "Qfx5100 ethernet switch," 2019. [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/datasheets/1000480-en.pdf

[10] Cisco, "Cisco nexus 3264q switch data sheet," 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-3264q-switch/datasheet-c78-734905.html

[11] ——, "Cisco," 2023. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-744052.html

[12] P. Jain, "Recent trends in next generation terabit ethernet and gigabit wireless local area network," in *2016 International Conference on Signal Processing and Communication (ICSC)*. IEEE, 2016, pp. 106–110.

[13] D. Firestone *et al.*, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *USENIX NSDI*, 2018.

[14] Xilinx, "Alveo U25 SmartNIC." [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo/u25.html

[15] Intel, "Intel® FPGA SmartNIC." [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/platforms/smartnic.html

[16] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[17] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Netagg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 249–262.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[19] Broadcom, "Broadcom Ships Tomahawk 5, Industry's Highest Bandwidth Switch Chip to Accelerate AI/MLWorkloads." [Online]. Available: https://investors.broadcom.com/node/60456/pdf

[20] Amazon, "Amazon EC2 F1 Instances." [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[21] Microsoft, "Microsoft Azure." [Online]. Available: https://azure.microsoft.com/en-gb

[22] Google, "Google Cloud: Cloud Computing Services." [Online]. Available: https://cloud.google.com/

[23] D. Sowry *et al.*, "Edge Data Centers." [Online]. Available: https://www.tiaonline.org/wp-content/uploads/2018/10/TIA_Position_Paper_Edge_Data_Centers-18Oct18.pdf

[24] J. West, *CompTIA Network+ guide to networks.* Cengage Learning, 2021.

[25] Cisco, "Cisco Nexus 9500 Series Switches Data Sheet." [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-729404.html

[26] ——, "Cisco Nexus 9300-GX2 Series Fixed Switches Data Sheet." [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-743854.html

[27] ——, "Cisco Business 350 Series Managed Switches Data Sheet." [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/business-350-series-managed-switches/datasheet-c78-744156.html

[28] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proceedings of the 1st annual symposium on Computer architecture*, 1973, pp. 21–28.

[29] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[30] M. J. Narasimha, "The batcher-banyan self-routing network: universality and simplification," *IEEE Transactions on Communications*, vol. 36, no. 10, pp. 1175–1178, 1988.

[31] Y.-S. Yeh, M. Hluchyj, and A. Acampora, "The knockout switch: A simple, modular architecture for high-performance packet switching," *IEEE Journal on selected Areas in communications*, vol. 5, no. 8, pp. 1274–1283, 1987.

[32] M. Karol *et al.*, "Input Versus Output Queueing on a Space Division Switch," *IEEE Trans. Commun*, vol. 35, no. 12, pp. 1347–1356, 1987.

[33] N. Gebara *et al.*, "Scheduling Algorithms for High Performance Network Switching on FPGAs: A Survey," *Intl. Conf. on Field Programmable Technologies*, 2018.

[34] N. McKeown, "The iSLIP Scheduling Algorithm for Input-queued Switches," *IEEE/ACM transactions on networking*, no. 2, pp. 188–201, 1999.

[35] W. E. Denzel *et al.*, "A Flexible Shared-buffer Switch for ATM at Gbs Rates," *Computer Networks and ISDN systems*, vol. 27, no. 4, pp. 611–624, 1995.

[36] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[37] J. Kim *et al.*, "Microarchitecture of a High-radix Router," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 420–431.

[38] Z. Dai and J. Zhu, "Saturating the Transceiver Bandwidth: Switch Fabric Design on FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 67–76.

[39] N. Gebara, J. Meng, W. Luk, and P. Costa, "Scheduling algorithms for high performance network switching on FPGAs: A survey," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 166–173.

[40] K. Yoshigoe *et al.*, "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed," in *Conference Proceedings of the IEEE International. Performance, Computing, and Communications Conference.* IEEE, 2003, pp. 481–485.

[41] NetFPGA, https://netfpga.org/site/.

[42] A. Bitar *et al.*, "Efficient and Programmable Ethernet Switching with a NoC-enhanced FPGA," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems.* ACM, 2014, pp. 89–100.

[43] M. S. Abdelfattah *et al.*, "Take the Highway: Design for Embedded NoCs on FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 2015, pp. 98–107.

[44] I. Swarbrick *et al.*, "Network-on-Chip Programmable Platform in Versal TM ACAP Architecture," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 2019, pp. 212–221.

[45] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO).* IEEE, 2016, pp. 1–13.

[46] Firestone *et al.*, "Azure accelerated networking:{SmartNICs} in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.

[47] P. Papaphilippou, J. Meng, and W. Luk, "High-performance FPGA network switch architecture," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 76–85.

[48] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA).* IEEE, 2019, pp. 279–291.

[49] R. A. Cooke and S. A. Fahmy, "Quantifying the latency benefits of near-edge and in-network FPGA acceleration," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 7–12.

[50] W. B. Johnson, "Extensions of lipschitz mappings into a hilbert space," *Contemp. Math.*, vol. 26, pp. 189–206, 1984.

[51] D. Achlioptas, "Database-friendly random projections: Johnson-lindenstrauss with binary coins," *Journal of computer and System Sciences*, vol. 66, no. 4, pp. 671–687, 2003.

[52] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 287–296.

[53] T. I. Cannings and R. J. Samworth, "Random-projection ensemble classification," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 79, no. 4, pp. 959–1035, 2017.

[54] S. Fox, S. Tridgell, C. Jin, and P. H. Leong, "Random projections for scaling machine learning on FPGAs," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 85–92.

[55] M. Nazemi, A. E. Eshratifar, and M. Pedram, "A hardware-friendly algorithm for scalable training and deployment of dimensionality reduction models on fpga," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2018, pp. 395–400.

[56] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[57] R. E. Schapire, "The boosting approach to machine learning: An overview," *Nonlinear estimation and classification*, pp. 149–171, 2003.

[58] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[59] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[60] B. Lou, D. Boland, and P. H. Leong, "fsead: a composable fpga-based streaming ensemble anomaly detection library," *ACM Transactions on Reconfigurable Technology and Systems*, 2022.

[61] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002, pp. 380–388.

[62] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

[63] A. Lu, Z. Fang, N. Farahpour, and L. Shannon, "CHIP-KNN: A configurable and high-performance k-nearest neighbors accelerator on cloud FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 139–147.

[64] F. An, T. Koide, and H. J. Mattausch, "A k-means-based multi-prototype high-speed learning system with FPGA-implemented coprocessor for 1-NN searching," *IEICE TRANSACTIONS on Information and Systems*, vol. 95, no. 9, pp. 2327–2338, 2012.

[65] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient knn algorithm implemented on FPGA based heterogeneous computing system using opencl," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 167–170.

[66] H. Hussain, K. Benkrid, C. Hong, and H. Seker, "An adaptive FPGA implementation of multi-core k-nearest neighbour ensemble classifier using dynamic partial reconfiguration," in *22nd international conference on field programmable logic and applications (FPL)*. IEEE, 2012, pp. 627–630.

[67] X. Song, T. Xie, and S. Fischer, "A memory-access-efficient adaptive implementation of knn on FPGA through HLS," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 177–180.

[68] J. Vieira, R. P. Duarte, and H. C. Neto, "kNN-STUFF: kNN streaming unit for FPGAs," *IEEE Access*, vol. 7, pp. 170 864–170 877, 2019.

[69] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar, "Compacting privacy-preserving k-nearest neighbor search using logic synthesis," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[70] F. An, H. J. Mattausch, and T. Koide, "An FPGA-implemented associative-memory based online learning method," *International Journal of Machine Learning and Computing*, vol. 1, no. 1, p. 36, 2011.

[71] V. T. Lee, J. Kotalik, C. C. Del Mundo, A. Alaghi, L. Ceze, and M. Oskin, "Similarity search on automata processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 523–534.

[72] Cisco, "Cisco Nexus 3000 Series Switches." [Online]. Available: https://www.cisco.com/c/en/us/products/switches/nexus-3000-series-switches/models-comparison.html

[73] A. M. Caulfield *et al.*, "A Cloud-scale Acceleration Architecture," *MICRO*, 2016.

[74] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," *Proc. of 45th Intl. Symp. on Computer Architecture*, 2018.

[75] "P4 Language." [Online]. Available: https://p4.org/

[76] Intel, "Intel Stratix10 TX Advance Information Brief." [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/\en/pdfs/literature/hb/stratix-10/s10_tx_fpga_aib.pdf

[77] Broadcom, "Tomahawk3 / BCM56980 Series." [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series

[78] J. Sheng *et al.*, "Towards Low-latency Communication on FPGA Clusters with 3D FFT Case Study," *Proc. Highly Efficient and Reconfigurable Technologies*, 2015.

[79] Intel, "Interlaken (2nd Generation) Intel FPGA IP User Guide." [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683396.pdf

[80] Xilinx, "UltraScale+ FPGAs Product Tables and Selection Guide." [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

[81] ——, "Block Memory Generator v8.3." [Online]. Available: https://www.xilinx.com/support/\documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf

[82] P. Gupta and N. McKeown, "Designing and Implementing a Fast Crossbar Scheduler," *IEEE Micro*, no. 1, pp. 20–28, 1999.

[83] Xilinx, "7 Series GTX Transceivers - TX and RX Latency Values." [Online]. Available: https://www.xilinx.com/support/answers/42662.html

[84] "Juniper." [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/datasheets/1000480-en.pdf

[85] "Cisco." [Online]. Available: https://www.cisco.com/c/en/us/products/\switches/nexus-3000-series-switches/models-comparison.html

[86] "FS." [Online]. Available: https://www.fs.com/uk/c/network-switches-3079

[87] L. Mhamdi and M. Hamdi, "Output queued switch emulation by a one-cell-internally buffered crossbar switch," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, vol. 7. IEEE, 2003, pp. 3688–3693.

[88] R. B. Magill, C. E. Rohrs, and R. L. Stevenson, "Output-queued switch emulation by fabrics with limited memory," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 606–615, 2003.

[89] N. McKeown, B. Prabhakar, and M. Zhu, "Matching output queueing with combined input and output queueing," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 35. Citeseer, 1997, pp. 595–603.

[90] M. Zukerman, "Introduction to queueing theory and stochastic teletraffic models," *arXiv preprint arXiv:1307.2968*, 2013.

[91] H. Ahmadi, W. E. Denzel, C. A. Murphy, and E. Port, "A high-performance switch fabric for integrated circuit and packet switching," *International journal of digital & analog cabled systems*, vol. 2, no. 4, pp. 277–287, 1989.

[92] Xilinx, "10g/25g high speed ethernet subsystem v3.0," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xxv_ethernet/v3_0/pg210-25g-ethernet.pdf

[93] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[94] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[95] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die stacking (3d) microarchitecture," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 469–479.

[96] J. Meng, N. Gebara, H.-C. Ng, P. Costa, and W. Luk, "Investigating the feasibility of FPGA-based network switches," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 218–226.

[97] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[98] D. Dua and C. Graff, "Uci machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[99] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[100] N. Kushmerick, "Learning to remove internet advertisements," in *Agents*, 1999, p. 175.

[101] "LIBSVM data: Classification (binary class)," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon.

[102] "LIBSVM data: Classification (binary class)," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#real-sim.

[103] J. Chen and J. Shao, "Nearest neighbor imputation for survey data," *Journal of official statistics*, vol. 16, no. 2, p. 113, 2000.

[104] S. Zhang, "Nearest neighbor selection for iteratively knn imputation," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2541–2552, 2012.

[105] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[106] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[107] D. He, M. Daum, W. Cai, and M. Balazinska, "Deepeverest: accelerating declarative top-k queries for deep neural network interpretation," *arXiv preprint arXiv:2104.02234*, 2021.

[108] J. Meng, C. Guo, N. Gebara, and W. Luk, "Fast and accurate training of ensemble models with FPGA-based switch," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 81–84.

[109] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language

supervision," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8748–8763.

[110] D. Dua and C. Graff, "Uci machine learning repository," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/internet+advertisements

[111] ——, "Uci machine learning repository," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Gisette

[112] ——, "Uci machine learning repository," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/REJAFADA+

[113] ——, "Uci machine learning repository," 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/QSAR+oral+toxicitye

[114] NetFPGA. (2021) NetFPGA SUME reference learning switch lite. [Online]. Available: https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/NetFPGA-SUME-Reference-Learning-Switch-Lite

[115] I. Stamoulias and E. S. Manolakos, "Parallel architectures for the knn classifier–design of soft ip cores and fpga implementations," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2, pp. 1–21, 2013.

[116] V. T. Lee, A. Mazumdar, C. C. del Mundo, A. Alaghi, L. Ceze, and M. Oskin, "Application codesign of near-data processing for similarity search," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 896–907.

[117] R. Kaplan, L. Yavits, and R. Ginosar, "Prins: Processing-in-storage acceleration of machine learning," *IEEE Transactions on Nanotechnology*, vol. 17, no. 5, pp. 889–896, 2018.

[118] S. Gorgin, M. Gholamrezaei, D. Javaheri, and J.-A. Lee, "knn-msdf: A hardware accelerator for k-nearest neighbors using most significant digit first computation," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*. IEEE, 2022, pp. 1–6.

[119] de Araújo *et al.*, "Hardware-accelerated similarity search with multi-index hashing," in *2019 IEEE (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2019, pp. 733–740.

[120] Y. Z. et al., "Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs," in *FPGA*, 2018.

[121] T. Ito, Y. Itotani, S. Wakabayashi, S. Nagayama, and M. Inagi, "A nearest neighbor search engine using distance-based hashing," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 150–157.

[122] W. Jiang and M. Gokhale, "Real-time classification of multimedia traffic using fpga," in *2010 International Conference on Field Programmable Logic and Applications*. IEEE, 2010, pp. 56–63.