Algorithm and Architecture Co-design for High Performance Digital Signal Processing

by

Jung Kuk Kim

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in the University of Michigan
2015

Doctoral Committee:

Assistant Professor Zhengya Zhang, Co-chair
Professor Jeffrey A. Fessler, Co-chair
Associate Professor Wei Lu
Associate Professor S. Sandeep Pradhan

ACKNOWLEDGEMENTS

First of all, I sincerely thank my doctoral research co-advisor, Professor Zhengya Zhang, for his advice and guidance. His insights into algorithm design and hardware design influenced my research deeply. He always led me the way to higher standards whenever I was complacent about my research. Because of his consideration and constant encouragement, I have been able to focus on my Ph.D. studies, and now move forward after I graduate. I would also like to thank my research co-advisor, Professor Jeffrey Fessler for his generous advice and comments on my Ph.D. work. I have benefited his insights into optimization algorithms to keep a balance between algorithm research and hardware research. I would like to thank Professor Wei Lu for his constructive comments on my research. His high standards have influenced my research on neuromorphic hardware. I would like to thank Prof. Sandeep Pradhan for reviewing my thesis proposal and dissertation. Furthermore, I would like to give my sincere thanks to Phil Knag and Thomas Chen for hard work, ceaseless discussions, and their contribution to the collaborative work on sparse coding.

My research was supported in part by DARPA. I enjoyed constructive discussions in bi-weekly meetings with Dr. Wei Lu, Garret Kenyon, Michael Flynn and Christof Teuscher, and my colleagues, Phil Knag, Patrick Sharidan, Thomas Chen, Zelin Zhang and Steven Mikes. Each meeting has stimulated my progress on neuromorhpic hardware. My research was also supported in part by the Korean Foundation of Advanced Studies scholarship and the Reithmiller fellowship. I also acknowledge BEEcube, Xilinx and Intel Corporation for the generous equipment donations.

I was fortunate to learn from many talented colleagues over the course of my Ph.D. career. Firstly, I would like to thank members of MICL. I sincerely thank Phil Knag and Thomas Chen for their contributions to the collaborative work in a sparse coding chip design, Youn Sung Park for sharing his experience in chip designs, Chia-Hsiang Chen, Shuanghong Sun, Shimming Song, Wei Tang, Chester Liu and Dike Zhou for being patient to answer my questions about hardware design and sharing many great memories. I would also like to thank Yejoong Kim and Seokhyun Jeong for sparing their time to review tool setup for my chip design. I would like to thank Taekwang Jang and Sunmin Jang for sharing their ideas and answering my questions on VLSI circuits and building blocks. I would like to thank members of Prof. Fessler's optimization algorithm group, Yong Long, Janghwan Cho, Donghwan Kim and Heung Nien for answering my questions about optimization algorithms for X-ray CT, and Hao Sun, Madison McGaffin, Mai Le and Jean Young Kwon for constructive discussions about various medical imaging applications.

I cannot imagine my graduate life without my friends. I especially thank Jaehun Jeong for listening to any concern, and correcting me if I am wrong. I also thank Suyoung Bang, Jeffrey Fredenburg, Inhee Lee, Hyoguem Rhew, and Yonghyun Shim for cheering me up when I had a difficult time. Thanks to Daeyon Jung, Jeongeun Kim, Ju-hyun Song, Jihye Kim, Dongmin Yoon, Dongsuk Jeon, Taehoon Kang, Inyong Kwon, Sinhyun Choi, Kihyuk Sohn, Jaeyoung Kim, Xuejing He, Yue Liu, Myungjoon Choi, Yong Lim, Wanyoung Jung, Sechang Oh and Kyojin Choo for sharing great memories in Ann Arbor. Special thanks to Seungha Baik, Chang-gyu Kim, Taewoo Han and Changwook Chun for being great friends from youth and cheering me up anytime wherever we are.

Finally, I would like to give many thanks to my parents, Kwangsung Kim and Bunja Gu. Their unlimited support, kind considerations, dedication to everything related to my life have helped me to study, and have motivated me to move forward.

I would like to give special thanks to my brother, Jungmyoung Kim for caring about my life and studies and being my best friend ever since I was born. I would also like to thank Anwida Prompijit for cheering me up over the course of my Ph.D. studies and helping me to be a better person. Again, I thank great individuals in University of Michigan, sister institutions, government labs, colleagues, friends and family. They all influenced me to write this doctoral thesis.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Algorithm and Architecture Co-design for High Performance Digital Signal Processing

by

Jung Kuk Kim

Co-chairs: Zhengya Zhang and Jeffrey A. Fessler

CMOS scaling has been the driving force behind the revolution of digital signal processing (DSP) systems, but scaling is slowing down and the CMOS device is approaching its fundamental scaling limit. At the same time, advanced DSP algorithms are continuing to evolve, so there is a growing gap between the increasing complexities of the algorithms and what is practically implementable. The growing gap can be bridged by exploring the synergy between algorithm design and hardware design, using the so-called co-design techniques.

In this thesis, algorithm and architecture co-design techniques are applied to X-ray computed tomography (CT) image reconstruction. Analysis of fixed-point quantization and CT geometry identifies an optimal word length, intrinsic parallelism, and a mismatch between the object and projection grids. A water-filling buffer is designed to resolve the grid mismatch, and is combined with parallel fixed-point arithmetic units to improve the throughput. The effects of the fixed-point arithmetic on the image quality are analyzed, and an analytical upper bound on the quantization error variance of the reconstructed image is derived. This investigation eventually leads

to an out-of-order sectored processing architecture that reduces the off-chip memory access by three orders of magnitude, allowing for a throughput of 925M voxel projections/s at 200MHz on a Xilinx Virtex-5 FPGA.

The co-design techniques are further applied to the design of spiking neural networks for sparse coding. Analysis of the neuron spiking dynamics leads to the optimal tuning of network size, spiking rate, and neuron update step size to keep the neuron spiking sparse and random. The resulting sparsity enables a bus-ring architecture to address routing complexity. The bus and ring sizes are optimized to achieve both high throughput and scalability. The architecture is demonstrated in a 65nm CMOS chip. The test chip demonstrates sparse feature extraction at a high throughput of 1.24G pixel/s at 1.0V and 310MHz. The error tolerance of sparse coding can be exploited to further enhance the energy efficiency.

As a natural next step after the sparse coding chip, a neural-inspired inference module (IM) is designed for object recognition. The object recognition chip consists of an IM based on the spiking locally competitive algorithm and a sparse event-driven classifier. A light-weight learning co-processor is integrated on chip to enable on-chip learning. The throughput and energy efficiency are further improved using a number of architectural techniques including sub-dividing the IM network and classifier into modules and optimal pipelining. The result is a 65nm CMOS chip that performs sparse coding at 10.16G pixel/s at 1.0V and 635MHz. The integrated IM and classifier provide extra error tolerance for voltage scaling, allowing the power to be lowered to 3.65mW at a throughput of 640M pixel/s.

Algorithm and architecture co-design techniques have been demonstrated in this work to advance the hardware design for CT image reconstruction, neuromorphic sparse coding, and object recognition. The co-design techniques can be applied to the design of other advanced DSP algorithms for emerging applications.

CHAPTER I

Introduction

High performance signal processing algorithms have shown a number of applications in medical imaging and machine learning. Model-based image reconstruction algorithms provide high quality X-ray computed tomography (CT) [1, 2, 3, 4, 5] and magnetic resonance imaging (MRI) images [6, 7, 8]. The emerging neural network algorithms have demonstrated applications in image processing, including feature extraction [9, 10, 11, 12, 13], object recognition [14, 15], as well as speech recognition [16, 17]. Conventional microprocessor based solutions are widely used to implement high-performance signal processing algorithms. However, the architecture is not tailored to the algorithms, limiting the achievable performance for practical applications.

X-ray CT systems using model-based iterative image reconstruction algorithms process a massive amount of high-resolution image data. Computations involved in practical image data require 100 billion floating-point multiply-accumulate operations per iteration and 100Gb/s memory bandwidth, so the CT image reconstruction algorithms are fundamentally memory-bound, presenting a severe challenge for von Neumann based hardware. The memory challenge is also seen in the implementation of neural network algorithms for machine learning applications. A 1000-neuron network requires 1 million neuron-to-neuron connections, and needs to store 1 million

connection weights in distributed memory for high performance signal processing.

Customizing the hardware to target applications in a so-called application specific integrated circuit (ASIC) is a promising approach towards energy-efficient, high-performance hardware implementation. However, most commercial ASICs have not been designed in a systematic way. Mapping DSP algorithms to ASIC architecture often results in some performance loss, and the impact of hardware architecture on the DSP algorithm is not always known. This thesis explores efficient ASIC hardware for high-performance signal processing using a co-design technique by following four systematic steps: 1) analysis of intrinsic characteristics of an algorithm, 2) implementation of custom architectures, 3) analysis of the impacts of architecture optimization on the algorithm performance, and 4) transformation of the algorithm to enhance architecture performance.

As a proof of the concept, this work discusses the co-design of two state-of-the-art signal processing systems: 1) iterative image reconstruction for X-ray CT and 2) sparse coding for feature extraction and object recognition. Three custom hardware accelerators are implemented through co-design techniques: 1) an FPGA-based forward projection accelerator, 2) a 65nm CMOS neural network processor, and 3) a 65nm CMOS end-to-end neuromorphic object recognition processor.

1.1 CT image reconstruction

X-ray computed tomography (CT) is a widely used medical imaging method that produces three-dimensional (3D) images of the inside of a body from many two-dimensional (2D) X-ray images. A 2D X-ray image captures X-ray photons that pass through a body. As different materials attenuate X-ray differently, they can be effectively differentiated by their attenuation coefficients. Using many X-ray images taken around an axis of rotation, the attenuation coefficient of each volume element (voxel) can be reconstructed, providing high-resolution imaging for medical diagnosis.

Among various image reconstruction algorithms for X-ray CT, this thesis primarily focuses on state-of-the art statistical iterative image reconstruction algorithms [1, 2]. The statistical methods incorporate measurement statistics in the image reconstruction problem and improve the image quality over iterations [3, 4, 5]. Current commercial CT systems using the iterative methods take on the order of an hour to reconstruct one patient CT scan, so it is crucial to accelerate image reconstruction for X-ray CT. An iterative method evaluates forward- and back- projection and regularization (if necessary) in each iteration, and finds a solution to minimize reconstruction errors through iterations. Since large data set and intensive computation involved in each iteration require a long computation time, it is necessary to accelerate the iterative methods for clinical use [18] including real-time image-guided surgery.

This work demonstrates algorithm and architecture co-design for fast iterative image reconstruction. General-purpose CPUs and GPUs are widely used hardware accelerators for high-performance CT image reconstruction, but it still takes a few minutes to obtain high quality 3D images, so there is a gap between their practical use and the performance of image reconstruction. Fixed-point datapath customized to the CT geometry enables highly parallel processing for a high effective throughput. The effects of fixed-point arithmetic are analyzed to quantify errors in the reconstructed images.

1.1.1 Scope of work

This work implements highly parallel forward-projection architectures for iterative CT image reconstruction through the co-design techniques [18, 19] in a sequence of four steps. 1) A study of CT geometry uncovers the relationship between geometry parameters and the range of a voxel projection, and the mismatch between the 3D object grid and the 2D projection grid. 2) Based on the study of CT geometry, a water-filling buffer is designed by resolving data dependency caused by the grid mis-

match, and parallel multiply-and-accumulate units are implemented to increase the throughput. 3) The effects of fixed-point quantization on image reconstruction are simulated using an iterative method to confirm the feasibility of fixed-point quantization. 4) An out-of-order sectored processing algorithm is developed to reduce off-chip memory bandwidth by up to three orders of magnitude, enabling highly parallel forward-projection processing.

This work analyzes the effects of fixed-point quantization to the CT image reconstruction [20]. This work applies a floating-point to fixed-point conversion to the iterative image reconstruction algorithm in order to reduce hardware costs. The effect of fixed-point quantization is modeled as a perturbation of floating-point arithmetic by injecting uniform white noise after the arithmetic. This work derives an analytical upper bound on the quantization error variance of the reconstructed image and shows that the quantization step size can be chosen to meet a given upper bound. The analytical results are confirmed by numerical simulations.

1.1.2 Related work

High-performance computing platforms have been proposed to accelerate image reconstruction. For example, graphics processing unit (GPU) has recently been demonstrated to achieve 10 to 100 times speedup over a microprocessor for image reconstruction [21, 22]. As a vector processor, GPU can be programmed for efficient parallel processing [23]. Provided with sufficient memory bandwidth, GPU accomplished a 30 times speedup of cone-beam Feldkamp (FDK) back-projection over a system based on 12 2.6-GHz dual-core Xeon processors [21], and a 12 times speedup of algebraic reconstruction [22]. Field-programmable gate array (FPGA) is another family of hardware platforms that enable more flexibility in mapping parallel computation with an improved efficiency. It was shown to accomplish a 6 times speedup of the cone-beam Feldkamp (FDK) back-projection [21], [24]. However, existing GPU

and FPGA implementations are tailored to analytical reconstruction algorithms or algebraic reconstruction methods [21, 22, 24, 25], and challenges still remain in mapping statistical iterative algorithms.

In comparison with prior work, this work performs algorithm and architecture co-design for fast iterative image reconstruction. CT geometry is investigated to identify intrinsic parallelism and data access sequence. A water-filling buffer is proposed to resolve pipeline stalls and support the average voxel consumption rate. This work adopts the fixed-point quantization in the hardware implementation, and the quantization effects are analyzed to quantify the image fidelity. An out-of-order sectored processing is proposed to improve the performance of the hardware architecture.

1.2 Sparse coding

Better understanding of the mammalian primary visual cortex has led to advances in computer vision [26, 27]. The visual cortical neurons respond to visual stimuli with spikes. The visual feature or region that stimulates a cortical neuron in visual cortex is known as the receptive field of the neuron [28, 29, 30]. The receptive fields of the visual cortical neurons can be compared to the basis functions that form the natural images we see, and are closely related to the intrinsic structures of natural images. Learning the receptive fields and neuron activities allows us to carry out many complex vision processing tasks, including efficient encoding of images and detecting features and objects [9, 10].

Sparse coding algorithms have successfully reproduced key features of the biological receptive fields. Sparse coding is inspired by biology as implementing a network of computational neurons and regularizing the activities of the neurons in the network. One unique feature of the network is that the activities of the neurons are sparse. Sparsity is appealing, but sparse coding faces challenges in hardware implementation. The number of interconnects between neurons grows quadratically with the number

of neurons in the network. Massively connected neurons require all-to-all communication for sparse coding and enormous on-chip memory bandwidth. Because of massive interconnects and high memory bandwidth, the conventional von Neumann architecture is ill suited for the sparse coding problem.

This work presents efficient hardware architectures for sparse coding. Hardware architecture determines the neuron spiking dynamics, which in turn determine the reconstruction error. The relationship between neuron spiking dynamics and hardware performance and complexity are not well understood. There are existing architectures for artificial neural networks. However, a direct mapping of sparse coding onto these architectures degrades hardware efficiency, because they are not tailored to sparse coding. Co-design techniques contribute to a better understanding of the spiking dynamics, and show how the spiking dynamics are affected by hardware architectures. These insights lead to custom hardware designs to achieve higher throughput and better energy efficiency.

Object recognition is one primary application of sparse coding. Recognizing objects in an image or video can be accomplished by first extracting features from the image using an inference module (IM), and then classifying the object based on the extracted features using a classifier. Both feature extraction and object classification can be computationally intensive, and their power consumption is high. Sparse feature extraction is advantageous as it performs sparse feature extraction and it reduces the workload of the classifier.

This work presents an end-to-end neuromorphic object recognition processor. An efficient sparse coding IM is realized by co-optimizing hardware and the accuracy of classification. The sparse outputs of the sparse coding IM are exploited in the design of an efficient classifier to accomplish object recognition.

1.2.1 Scope of work

This work develops custom digital hardware architecture for sparse coding by co-optimizing neuron spiking dynamics and high-performance network architectures [31]. Similar to the CT image reconstruction, this work was conducted in four steps. 1) A study of neuron spiking dynamics uncovers design considerations involving the network size, the neuron update step size and firing rate. An optimal tuning of these parameters keeps the neuron spikes sparse and random over the inference window, and reduces reconstruction errors. 2) Two practical hardware architectures are investigated to address routing complexity and scalability. A bus architecture provides efficient communications, but results in spike collisions. A ring architecture is more scalable than the bus, but causes neuron misfires. 3) Spike collision is reduced by leveraging sparsity, so an arbitration-free bus is designed to tolerate collisions without the need of arbitration. To reduce neuron misfires, a latent ring architecture is designed to damp the neuron responses. 4) The bus and the ring are combined in a hierarchical architecture to achieve both high throughput and scalability, and to keep the reconstruction error low.

Based on the co-design, this work implements a sparse coding ASIC for feature extraction [32, 33]. Routing complexity and scalability are addressed by building a 2-layer bus-ring hierarchy. Neurons in a cluster are connected in a 2D bus to improve the communication delay over a 1D bus, and the roots of neuron clusters are linked in a short systolic ring to reduce latency. To save power in inference, memory is divided into a core section to support inference, and an auxiliary section that is only powered on for learning. An approximate learning scheme tracks only significant neuron activities to complete on-chip learning in seconds. Error tolerance of a soft sparse coding algorithm is exploited to reduce supply voltages, improving energy efficiency.

Continuing with the design of a sparse coding ASIC, this work advances a ca-

pability of sparse coding by demonstrating a state-of-the-art neuromorphic object recognition processor through a number of co-design techniques [34]. In particular, 1) a convolutional sparse coding based inference module (IM) is developed to infer large input images with feature dictionary that does not depend on the input size. 2) An event-driven classifier is designed, and it is activated by sparse neuron spikes, reducing its power and simplifying its implementation by removing all multiplications. 3) A 2-stage pipelined neuron is designed in a 2-layer bus-ring architecture to achieve a very high throughput. 4) To further enhance the throughput, neurons are organized in multiple IM networks, and the classifier is divided into multiple sub-modules to process multiple image patches in parallel. Integrated IM and classifier provides extra error tolerance for voltage scaling to improve the energy efficiency.

A part of this work was done by my co-workers Phil Knag and Thomas Chen. They have contributed to the design and implementation of the sparse coding ASIC chip.

1.2.2 Related work

To implement neural networks, many hardware architectures have been developed. The direct mapping of neural networks onto VLSI hardware [35] is not scalable due to the overwhelming interconnect and memory bandwidth necessary to support the full connectivity between neurons and the access to synaptic weights. More scalable architectures including bus [36], ring [37] and array [38] solved the interconnect bottleneck, and the invention of address-event representation (AER) [39, 40] enables the efficient time-multiplexing of sparse neuron spikes on a shared bus. Since then, much progress has been made in the key challenging areas of compact synaptic weight storage [41, 42, 43], efficient neuron and synapse circuits [44, 41, 42, 43, 45], and scalable synaptic connections [42, 43, 46, 45, 47, 48].

The latest wave of hardware designs for neural networks has demonstrated in-

creasing capabilities, from simulating real-time spike-timing-dependent plasticity [44] and cortical circuits [41] to digit recognition [42] and pattern recognition [43], from multilayer vision sensing and actuation [46] to performing arbitrary mathematical computations [45] and simulating neuroscience experiments [47]. At the same time, the integration scale has gone from tens of neurons [44] to over 10K neurons [46] and over 10M synaptic connections [45], and the power consumption has been lowered to mW level [43] and the energy reduced to tens of pJ/spike [42]. However, some of the latest works are not directly applicable to sparse coding due to the mismatch of learning rules and our requirement of entirely on-chip learning capability. The general-purpose solutions are applicable, but they are not tailored to sparse coding algorithm, thus the energy and area efficiency will be sacrificed. In comparison, this work proposes a custom hardware accelerator for sparse coding, and we focus on a synchronous digital implementation which exhibits robust deterministic logical behavior at nominal operating conditions [43]. Alternative designs including analog and asynchronous approaches offer unique advantages and are also expected to affect algorithm dynamics. They remain our future work and will not be discussed in this thesis.

High performance and energy efficient object recognition processors are designed by implementing cellular neural network based IM [49], SIFT based IM [50], and neural-fuzzy logic or algorithm based IM [51, 52]. The processors are used for visual attention and real-time object database matching. The object recognition processors achieve a high throughput of 100 to 200 GOPS [49, 51], and the power consumption can be enhanced to 50mW for mobile object recognition [52]. In comparison, this work is motivated by sparse coding, and implements spiking neurons in the IM for sparse feature extraction and a sparse spike-event driven classifier that are dedicated to enhance energy efficiency to 5.7pJ/pixel at 40MHz, dissipating 3.65mW.

Past work on sparse coding IM design has produced an 18-neuron spiking LCA

based analog IM [53], but the small scale is not suitable for practical problems. A 256-neuron digital IM using SAILnet [32] is scalable and achieved a much higher throughput, but the design was dominated by memory and it is not capable of object classification. Digital 256-neuron networks using crossbar architecture [43, 42] and a scalable deep learning processor [54] have implemented neuromorphic IMs such as spike-timing-dependent plasticity (STDP) [55] and variations of Restricted Boltzmann Machine (RBM) [56, 57], but the designs ignore sparsity, missing the opportunity to reduce power consumption and hardware cost [33, 58]. Furthermore, compared to deep learning approaches, the neuromorphic networks considered in this work use not only feed-forward but also inhibitory feedback connections to reducing firing rate, resulting in sparse spiking. This work demonstrates on-chip object recognition by implementing a spiking LCA based digital IM. The IM uses the ideas of convolutional neural networks to reduce the size of on-chip memory for area- and power-efficient implementation.

1.3 Outline

Chapter II discusses the background of statistical image reconstruction for X-ray CT. Model-based penalized weighted least square formulation is presented for CT image reconstruction, and particularly the separable footprint (SF) forward- and back-projection method is provided for fast image reconstruction. To reduce hardware cost, fixed-point quantization is applied to the iterative method. The cone-beam CT geometry is analyzed for efficient hardware implementation of the SF projector.

Chapter III describes a parallel forward-projection architecture for fast CT image reconstruction. A study of the CT geometry allows for loop-level parallelism by taking advantage of SIMD architecture and stall-free pipelined architecture by making use of a water-filling buffer. The effects of fixed-point arithmetic used in the projector are investigated in a diagonally preconditioned gradient descent method. An analytical

upper bound on the quantization error variance of the reconstructed image is derived. Finally, this chapter proposes an out-of-order voxel scheduling to reduce the off-chip memory bandwidth by 3 orders of magnitue. As a proof of the concept, this chapter summarizes the implementation results of a 5-stage pipelined, 55-way parallel projector on a Xilinx Virtex-5 FPGA.

Chapter IV provides the background of sparse coding that models the activities of cortical neurons in the human brain and encodes input signals into sparse signals using learned receptive fields (or feature dictionary). SAILnet, LCA, spiking LCA and hardware implementation challenges are discussed. Also, this chapter analyzes the tradeoff between the fidelity of sparse coding and the hardware performance. Particularly, the focus is on three tuning parameters, 1) network size, 2) target firing rate and 3) neuron update step size, and their effects on image fidelity and hardware performance.

Chapter V explores practical hardware architectures for sparse coding. Two scalable architectures (bus and ring) are discussed, and the impacts of the architectures on the performance of algorithm and hardware are analyzed. Particularly, spike collision rate on the bus is derived to identify the tradeoff between the image fidelity and collision rate. In a ring architecture, the image fidelity is improved by damping neuron responses and implementing a holding policy. The proposed 2-layer architecture combines the bus and the ring to achieve both high throughput and scalability. As a proof of the concept, the three architectures are synthesized and compared.

Chapter VI presents the design of a 3.06mm^2 65nm sparse coding ASIC that implements a 2-layer grid-ring architecture. In this ASIC chip, 256 leaky integrate-and-fire neurons are connected in 2D local grids, which are linked in a 4-stage systolic ring to reduce the communication latency. A snooping core is attached to the ring to record a subset of spikes for a fast and approximate learning. Fixed-point arithmetic with adjustable precision allows for highly parallel inference and on-chip learning.

Error tolerance of the SAILnet algorithm enables energy-efficient inference, allowing the memory supply voltage to be lowered to 440mW. The sparse coding ASIC demonstrates sparse feature extraction and learning of features in natural images.

Chapter VII presents the design of a 1.82mm^2 65nm neuromorphic object recognition processor. The 256-neuron IM is organized in four parallel neural networks to process four image patches to achieve a high throughput. The sparse neuron spikes allow the classifier implementation to be simplified by removing all multiplications. A light-weight co-processor performs efficient on-chip learning by taking advantage of the sparse neuron activity. The result is a 10.16G pixel/s test chip that dissipates 268mW. Integrated IM and classifier provides extra error tolerance for voltage scaling, allowing the power to be lowered to 3.65mW at a throughput of 640M pixel/s.

CHAPTER II

Background and simulation of CT image reconstruction

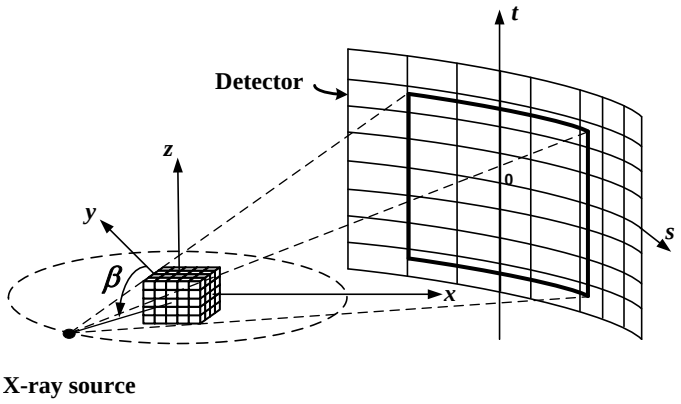


Figure 2.1: Axial cone-beam arc-detector geometry for X-ray CT.

Current generation CT systems have a cone-beam projection geometry, illustrated in Fig. 2.1 [59, 2, 60, 61]. The X-ray source rotates on a circle centered at $(x, y) = (0, 0)$ on the $z = 0$ plane. The angle β indexes the projection view measured from positive y-axis to X-ray source. For each angle β, the source emits X-rays that project the volume onto the detector. The transaxial direction s is perpendicular to z and the axial direction t is parallel to z.

This chapter introduces iterative methods for CT image reconstruction. The floating-point to fixed-point conversion is applied to the iterative methods to con-

This work is based in part on [18].

firm the feasibility of use, and the projection geometry is analyzed for an efficient mapping onto hardware.

2.1 Background of CT image reconstruction

In current clinical practice, a single CT scan using a state-of-the-art helical CT scanner records up to several thousand X-ray images taken in multiple rotations as the patient's body is moved slowly through the scanner. The projections are captured on an array of detector cells and a dedicated computer is used for image construction. Efficient algorithms, such as filtered backprojection (FBP) [62] and its variants, are in common commercial use to handle large projection data sets and reconstruct images at sufficient throughput. However, being an analytical algorithm, FBP disregards the effects of noise. To improve the image quality and/or reduce X-ray dose, statistical image reconstruction methods have been proposed [1, 2]. These methods are based on accurate projection models and measurement statistics, and formulated as a maximum likelihood (ML) estimation. Iterative algorithms such as conjugate gradient (CG) [3], coordinate descent (CD) [4] and ordered subsets (OS) [5], have been proposed. These algorithms find the minimizer of a cost function by iterative forward- and back-projection. Iterations increase the compute load substantially over FBP and impede routine clinical use.

2.1.1 Statistical iterative image reconstruction

A CT system captures a large series of projections at different view angles, recorded as sinogram. Mathematically, sinogram y can be modeled as $y = Af + \varepsilon$, where f represents the volume being imaged, A is the system matrix, or the forward-projection model, and ε denotes measurement noise. The goal of image reconstruction is to estimate the 3D image f from the measured sinogram y. A statistical image reconstruction method estimates f based on detector measurement statistics. The estimator $\hat{f}$

can be formulated as a solution to a weighted least square (WLS) problem [2],[63]

$$\hat{f} = \arg\min_f \frac{1}{2}\|y - Af\|_W^2, \tag{2.1}$$

where W is a diagonal matrix with entries based on photon measurement statistics [2]. Please see [64] for the detailed derivation of (2.1). A solution to (2.1) satisfies $A'WA\hat{f} = A'Wy$ [63]. If $A'WA$ is invertible, the unique solution to (2.1) is given by $\hat{f} = (A'WA)^{-1} A'Wy$, where A', the adjoint of the system matrix, represents the back-projection model. This solution can be interpreted as the weighted back-projection of y, followed by a deconvolution filter $(A'WA)^{-1}$. As the deconvolution filter has a high pass characteristic, the deconvolved image is affected by high frequency noise [63]. One approach to control this noise is to add a penalty term to form a penalized weighted least square (PWLS) [2],[63] cost function:

$$\hat{f} = \arg\min_f \Psi(f) = \arg\min_f \frac{1}{2}\|y - Af\|_W^2 + \beta R(f), \tag{2.2}$$

where $R(f)$ is known as the regularizer and β is a regularization parameter. One example of $R(f)$ is an edge-preserving regularizer [65].

Minimizing (2.2) requires iterative methods [3, 4, 5]. In this chapter, we consider a diagonally preconditioned gradient descent method to solve (2.2) [5], [63]:

$$\hat{f}^{(i+1)} = \hat{f}^{(i)} - D\nabla\Psi(\hat{f}^{(i)}) = \hat{f}^{(i)} + D\left[A'W(y - A\hat{f}^{(i)}) - \beta\nabla R(\hat{f}^{(i)})\right]. \tag{2.3}$$

The solution is obtained iteratively. In each iteration, a new 3D image estimate $\hat{f}^{(i+1)}$ is obtained by updating the previous image $\hat{f}^{(i)}$ with a chosen step, the negative gradient of the cost function $\Psi(\hat{f})$ scaled by D. Fig. 2.2 shows a block diagram of this iterative approach. To start, the CT scanner produces the measured sinogram, y and the FBP algorithm is used to estimate the initial image $\hat{f}^{(0)}$, followed by computed

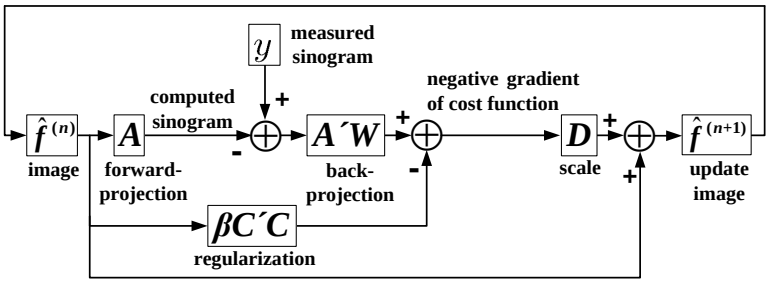


Figure 2.2: Block diagram of iterative image reconstruction.

forward-projection to obtain the computed sinogram $A\hat{f}^{(0)}$. The error between the computed and measured sinogram $y - A\hat{f}^{(0)}$ is back-projected $A'W(y - A\hat{f}^{(0)})$, then offset by a regularization term. The result is scaled by D, and used to improve the initial image to produce $\hat{f}^{(1)}$. The image $\hat{f}$ is iteratively updated to minimize the cost function.

2.1.2 Forward- and back- projection

Recently, a separable footprint (SF) projection algorithm was designed to simplify the forward-projection by approximating the voxel footprints as separable functions [66]. The SF projector has high accuracy and favorable speed, but it is still very computationally intensive: each forward- and back-projection requires on the order of 100 billion floating-point multiply-accumulate (MAC) operations, requiring minutes or longer for each forward- and back-projection on a state-of-the-art multicore microprocessor [19].

Forward and back-projection are the most computationally intense operations in iterative image reconstruction due to the large size of the system matrix A. It is infeasible to store A, thus the forward-projection $Af^{(i)}$, and back-projection $A'W(y - f^{(i)})$ in (2.3) are computed on the fly.

The forward-projection is mathematically based on the Radon transform. The Radon transform of a 3D volume $f(x, y, z)$ at view angle β is described by the line integrals[66]:

$$g(s,t;\beta) = \int\limits_{L(s,t,\beta)} f(x,y,z)dl, \tag{2.4}$$

where $L(s,t,\beta)$ is the line that connects the X-ray source and the detector cell at (s,t). In a practical implementation, a 3D continuous volume $f(x,y,z)$ is discretized to a collection of volume elements, or voxels $f[n_1,n_2,n_3]$, where $[n_1,n_2,n_3]$ is the voxel coordinate. The grid spacings are Δ_{x}, Δ_{y}, Δ_{z} and dimensions are N_{x}, N_{y}, N_{z} along the x, y, z directions. Let β_0 be the common voxel basis function, defined as a cubic function, $\beta_0(x,y,z)= \mathrm{rect}(x)\mathrm{rect}(y)\mathrm{rect}(z)$, and $(x_{\mathrm{c}}[n_1], y_{\mathrm{c}}[n_2], z_{\mathrm{c}}[n_3])$ be the location of voxel $[n_1,n_2,n_3]$. We have

$$f(x,y,z) = \sum_{n_1=0}^{N_{\mathrm{x}}-1} \sum_{n_2=0}^{N_{\mathrm{y}}-1} \sum_{n_3=0}^{N_{\mathrm{z}}-1} f[n_1,n_2,n_3] \beta_0 \left(\frac{x - x_{\mathrm{c}}[n_1]}{\Delta_{\mathrm{x}}}, \frac{y - y_{\mathrm{c}}[n_2]}{\Delta_{\mathrm{y}}}, \frac{z - z_{\mathrm{c}}[n_3]}{\Delta_{\mathrm{z}}} \right). \tag{2.5}$$

To account for the finite detector cell size, the projection is convolved with the detector blur $h(s,t)$. Following a common assumption that the detector blur is shift invariant, independent of the view angle β, and acts only along s and t coordinates, then the ideal noiseless forward-projection on the detector cell $[k,l]$ centered at $(s_{\mathrm{k}}, t_{\mathrm{l}})$ is given by

$$y_\beta[k,l] = \sum_{n_1=0}^{N_{\mathrm{x}}-1} \sum_{n_2=0}^{N_{\mathrm{y}}-1} \sum_{n_3=0}^{N_{\mathrm{z}}-1} a_{\mathrm{b}}(s_{\mathrm{k}}, t_{\mathrm{l}}; \beta; n_1, n_2, n_3) f[n_1,n_2,n_3], \tag{2.6}$$

where

$$a_{\mathrm{b}}(s_{\mathrm{k}}, t_{\mathrm{l}}; \beta; n_1, n_2, n_3) \triangleq \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} h(s_{\mathrm{k}} - s, t_{\mathrm{l}} - t) a(s,t;\beta;n_1,n_2,n_3) ds dt, \tag{2.7}$$

and

$$a(s,t;\beta;n_1,n_2,n_3) \triangleq \int\limits_{L(s,t,\beta)} \beta_0 \left(\frac{x - x_{\mathrm{c}}[n_1]}{\Delta_{\mathrm{x}}}, \frac{y - y_{\mathrm{c}}[n_2]}{\Delta_{\mathrm{y}}}, \frac{z - z_{\mathrm{c}}[n_3]}{\Delta_{\mathrm{z}}} \right) dl, \tag{2.8}$$

where $a(s,t;\beta;n_1,n_2,n_3)$ is the footprint of voxel $[n_1,n_2,n_3]$ and $a_{\mathrm{b}}(s_{\mathrm{k}},t_{\mathrm{l}};\beta;n_1,n_2,n_3)$ is the blurred footprint. For a detailed description of this derivation, see [63]. The separable footprint (SF) method [66] approximates the blurred footprint function as the product of $a_{\mathrm{b1}}(s_{\mathrm{k}},\beta;n_1,n_2)$ and $a_{\mathrm{b2}}(t_{\mathrm{l}},\beta;n_1,n_2,n_3)$, thus (2.6) is approximated as

$$y_\beta[k,l] \approx \sum_{n_1=0}^{N_{\mathrm{x}}-1} \sum_{n_2=0}^{N_{\mathrm{y}}-1} \sum_{n_3=0}^{N_{\mathrm{z}}-1} a_{\mathrm{b1}}[k,\beta;n_1,n_2] a_{\mathrm{b2}}[l,\beta;n_1,n_2,n_3] f[n_1,n_2,n_3].$$

Based on (2.9), one complete forward-projection involves multiplication and summation over six nested loops: n_1, n_2, n_3, β, k, and l. For a practical object made up of more than 10 million voxels, a SF forward-projection that comprises more than 900 view angles, as in a commercial axial CT scanner [2], requires on the order of 100 billion multiply-accumulate (MAC) operations. In the following sections, we explore architecture and algorithm co-optimization to accelerate the SF forward-projection.

For the sake of completeness, we briefly summarize back-projection. Back-projection is the operation that smears the projection in detector space back into the object space to reconstruct the 3D volume [63]. Back-projection is mathematically described as

$$f_{\mathrm{b}}[n_1,n_2,n_3] = \sum_{n_\beta=0}^{N_\beta-1} \sum_{l=0}^{N_{\mathrm{t}}-1} \sum_{k=0}^{N_{\mathrm{s}}-1} a_{\mathrm{b}}(s_{\mathrm{k}},t_{\mathrm{l}};n_\beta;n_1,n_2,n_3) g[k,l,n_\beta], \tag{2.9}$$

where $g[k,l,n_\beta]$ is the weighted difference between measured sinogram and the computed sinogram $y_{\beta(n_\beta)}[k,l]$. Similarly, the SF method approximates back-projection as

$$f_{\mathrm{b}}[n_1, n_2, n_3] \approx \sum_{n_\beta=0}^{N_\beta-1} \sum_{l=0}^{N_{\mathrm{t}}-1} \sum_{k=0}^{N_{\mathrm{s}}-1} a_{\mathrm{b1}}[k, n_\beta; n_1, n_2] a_{\mathrm{b2}}[l, n_\beta; n_1, n_2, n_3] g[k, l, n_\beta]. \quad (2.10)$$

Note that the equations governing forward- and back-projection are similar and they also share a common architecture. In this chapter, we will focus the discussions on forward-projection, but the results can also be applied to back-projection.

2.2 Fixed-point quantization and CT geometry

This section explores fixed-point quantization error effects and CT projection geometry. Numerical simulation verifies that with much reduced wordlengths, the fixed-point forward-projection provides a comparable image quality in an iterative algorithm. The CT projection geometry is analyzed to identify the intrinsic parallelism and data access sequence for an highly parallel hardware accelerator.

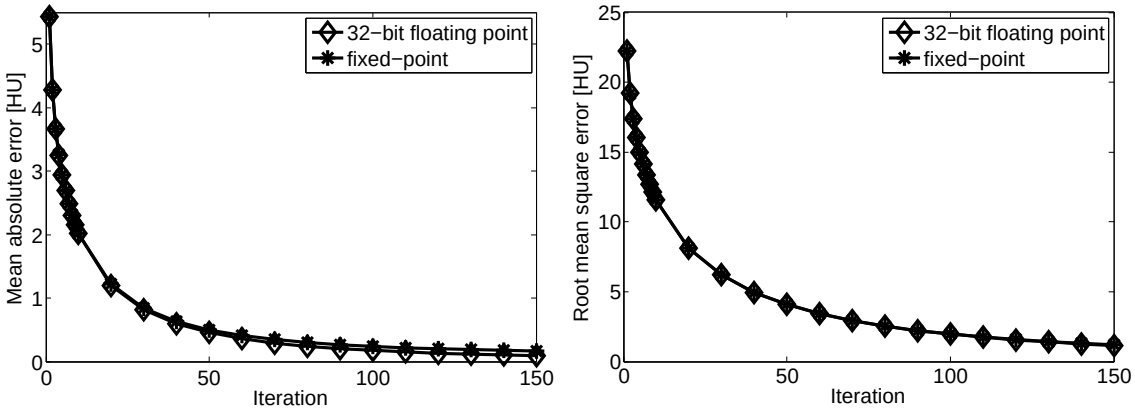


Figure 2.3: (a) Mean absolute error and (b) root mean square error of iterative image reconstruction using floating-point and fixed-point quantization.

2.2.1 Quantization errors investigation

Iterative CT image reconstruction algorithms are usually implemented in 32-bit single-precision floating-point quantization. Floating-point arithmetic costs more hardware resources and longer latency than integer (or fixed-point) operations. The substantially smaller area and higher speed provide strong incentives for using fixed-point operations. However, fixed-point quantization introduces errors that may degrade image quality. We show in the following that good image quality can be achieved with appropriate quantization choice and sufficient number of iterations.

Our experiment was done using a 61-slice test volume, with each slice made up of 320×320 voxels. Errors are defined in reference to a baseline that is the image reconstructed using 32-bit floating-point quantization after 1,000 iterations. We converted floating-point to fixed-point and varied the word length and quantization of each parameter and operand. Mean absolute error (MAE) and root mean square error (RMSE) of the image update in every iteration were measured compared to the baseline. The errors are expressed in Hounsfield unit (HU), which is a linear transformation of the linear attenuation coefficient (the attenuation coefficient of water at standard pressure and temperature is defined as 0 HU and that of the air is -1,000 HU).

Table 2.1: Fixed-Point Quantization of Iterative Image Reconstruction

Forward-projection		Back-projection	
Parameter	Quant.	Parameter	Quant.
f	Q13.0	g	Q5.15
$a_{\mathrm{b}2}$	Q1.15	$a_{\mathrm{b}1}$	Q3.17
$a_{\mathrm{b}2}f$	Q13.3	$a_{\mathrm{b}1}g$	Q7.15
$\sum_{n_3} a_{\mathrm{b}2}f$	Q13.3	$\sum_{k} a_{\mathrm{b}1}g$	Q8.15
$a_{\mathrm{b}1}$	Q3.13	$a_{\mathrm{b}2}$	Q1.15
$\sum_{n_3} a_{\mathrm{b}1}a_{\mathrm{b}2}f$	Q15.8	$\sum_{k} a_{\mathrm{b}1}a_{\mathrm{b}2}g$	Q9.15
$\sum_{n_1}\sum_{n_2}\sum_{n_3} a_{\mathrm{b}1}a_{\mathrm{b}2}f$	Q20.8	$\sum_{n_\beta}\sum_{l}\sum_{k} a_{\mathrm{b}1}a_{\mathrm{b}2}g$	Q9.15

We used an OS algorithm [5] with 82 subsets which is a variation of (2.3) that uses a subset of the projection views for each update. Fig. 2.3 comparises the 32-bit floating-point quantization and the fixed-point quantization described in Table 2.1. We use the notation $Qn_{\text{int}}.n_{\text{frac}}$ to denote a fixed-point format with n_{int} before the radix point and n_{frac} after the radix point. The experiment confirms that the fixed-point quantization errors introduced can be limited to fairly low levels. More iterations can help suppress the errors, and the word length can be increased to reduce the errors further if necessary.

Fig. 2.4 shows the images obtained by iterative image reconstruction as well as the absolute pixel-by-pixel differences between the reconstructed image using 32-bit floating-point quantization and the reconstructed image using fixed-point quantization. Three representative slices in the region of interest are shown from left to right. The vast majority of the pixel errors remain relatively small. We observe no perceptual difference between floating-point and fixed-point reconstructed images. These initial results suggest that the iterative image reconstruction algorithm can be robust to quantization error. The property allows us to simplify the hardware with much more efficient integer arithmetic and smaller memory.

2.2.2 Projection geometry

The projection geometry is central to the proposed algorithms. Fig. 2.5 illustrates the X-ray projection of a single voxel of dimension $\Delta_{\text{x}} \times \Delta_{\text{y}} \times \Delta_{\text{z}}$ centered at (x, y, z). We define the magnification factor $M_\beta(x, y)$ as the ratio of the source-to-detector distance D_{sd} (which is a constant in cone-beam geometry) over the distance between the source and $(x, y, 0)$. (The magnification factors of all voxels in an axial column are equal.) $M_\beta(x, y)$ is maximized when the voxel is closest to the X-ray source and minimized when the voxel is furthest to the X-ray source, i.e.,

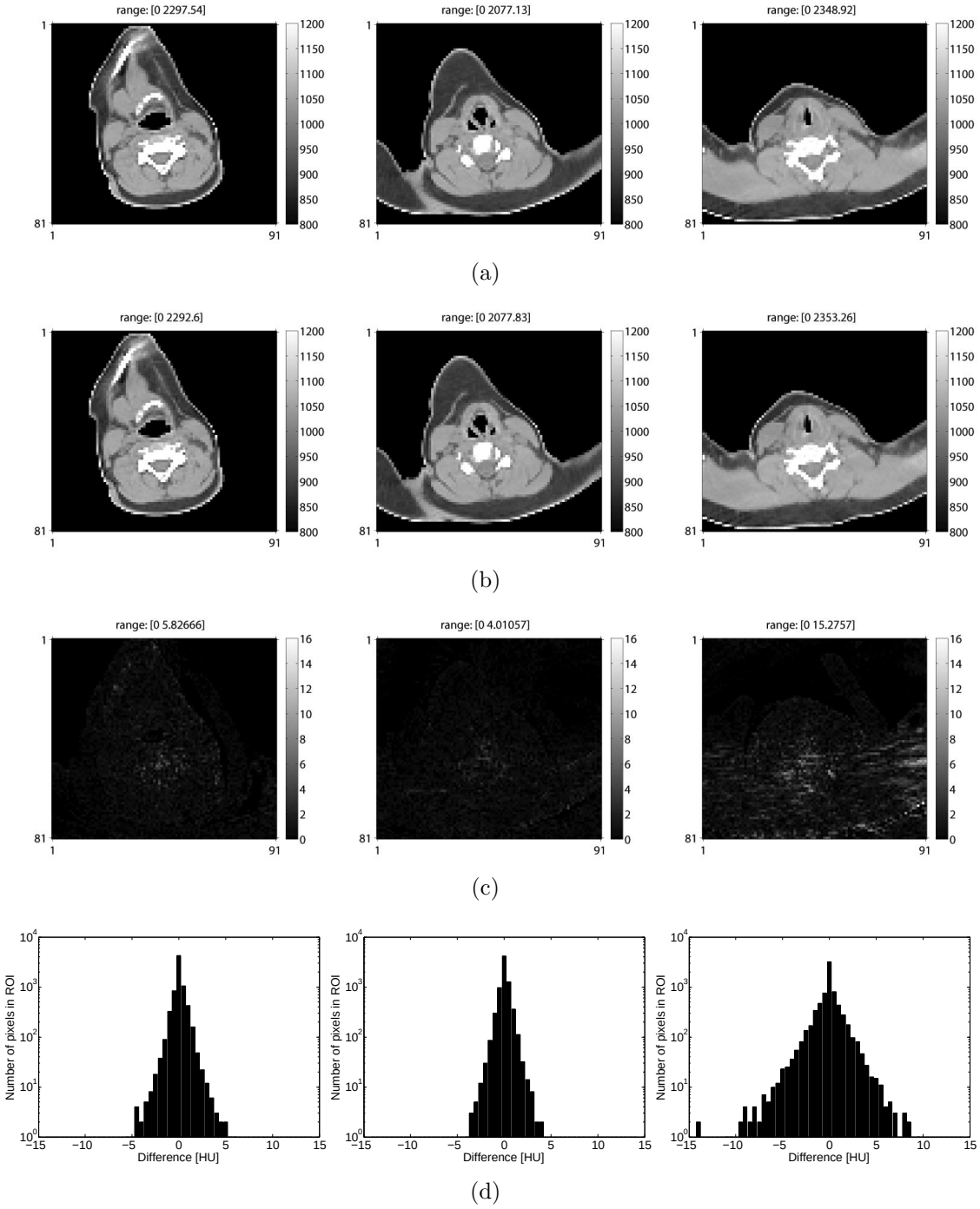


Figure 2.4: Reconstructed images using (a) 32-bit floating-point quantization, (b) fixed-point quantization, (c) absolute pixel-by-pixel differences between the floating-point and the fixed-point quantization, and (d) histograms of the differences in logarithm scale. Three slices in the region of interest are shown: slice 17, 31 and 45 from left to right.

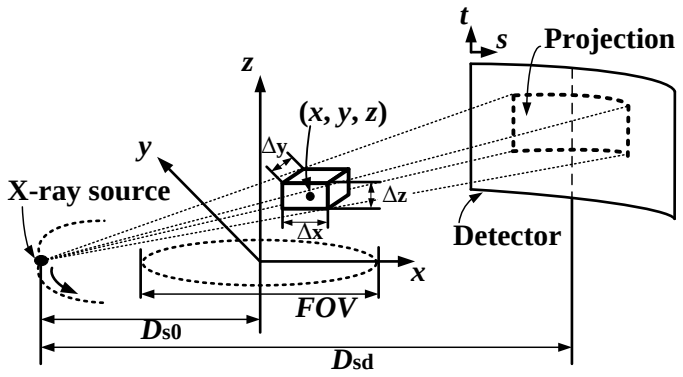


Figure 2.5: Forward-projection of a single voxel.

$$\frac{D_{\mathrm{sd}}}{D_{\mathrm{s0}} + FOV/2} \leq M_\beta(x, y) \leq \frac{D_{\mathrm{sd}}}{D_{\mathrm{s0}} - FOV/2}, \tag{2.11}$$

where FOV, or field of view, is the diameter of the volume that is reconstructed from all view angles, and D_{s0} is the source-to-rotation-center distance.

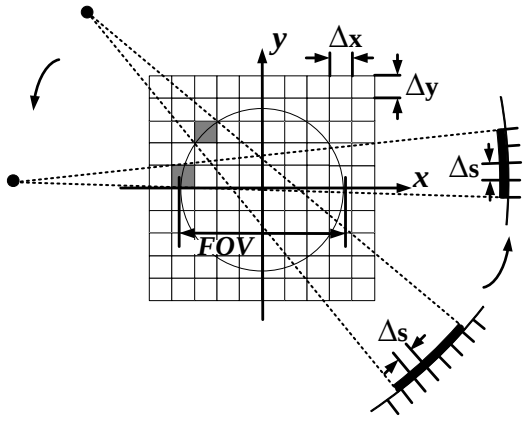


Figure 2.6: Top view of the transaxial span of the forward-projection of one voxel.

Now, consider the position of a voxel relative to the X-ray source – the transaxial width of a voxel's projection is maximized if the transaxial diagonal of the voxel is perpendicular to the line joining the X-ray source and the center of the voxel, illustrated in Fig. 2.6. Considering both the magnification and the transaxial diagonal of the voxel, the transaxial span of the projection of a voxel, quantized to the axial spacing Δ_{s} of the detector grid, is

$$s_{\text{transaxial}} \leq \left\lceil \frac{\sqrt{\Delta_{\text{x}}^2 + \Delta_{\text{y}}^2}}{\Delta_{\text{s}}} M_\beta(x,y) \right\rceil + 1 \leq \left\lceil \frac{\sqrt{\Delta_{\text{x}}^2 + \Delta_{\text{y}}^2}}{\Delta_{\text{s}}} \frac{D_{\text{sd}}}{D_{\text{s0}} - FOV/2} \right\rceil + 1 = s_{\text{bin}}, \tag{2.12}$$

where $\lceil\ \rceil$ denotes ceiling.

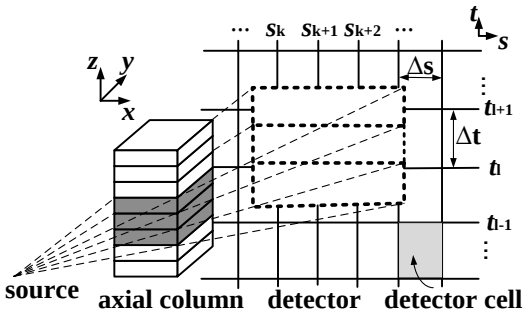


Figure 2.7: Forward-projection of one axial column of voxels.

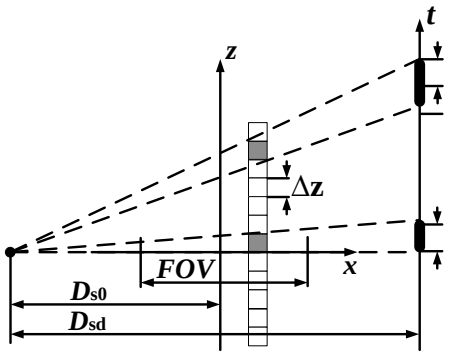


Figure 2.8: Side view of the axial span of the forward-projection of one voxel.

The magnification factor in (2.11) can also be used to derive the axial span. Typically the axial spacing Δ_{t} of the detector grid is designed to match the voxel grid Δ_{z} by having $\Delta_{\text{t}}/\Delta_{\text{z}} = D_{\text{sd}}/D_{\text{s0}}$. Therefore, on average one voxel maps to one detector cell along the axial direction. However, grid misalignment and geometry cause multiple consecutive voxels in an axial column to project to a single detector cell, as shown in Fig. 2.7. The axial height of a voxel's projection is minimized if the voxel is located on the $z = 0$ plane, illustrated in Fig. 2.8. It follows that the number of voxels in an axial column that project to a single detector cell is

$$z_{\text{axial}} \leq \left\lceil \frac{\Delta_{\text{t}}}{\Delta_{\text{z}}} \frac{1}{M_\beta(x,y)} \right\rceil + 1 \leq \left\lceil \frac{FOV/2}{D_{\text{s0}}} \right\rceil + 2 = z_{\text{vx}}. \tag{2.13}$$

Table 2.2: Sample Helical Cone-beam CT Geometry Parameters

Parameter	Value	Parameter	Value
N_1	320	Δ_{x}	2.1911 $[mm]$
N_2	320	Δ_{y}	2.1911 $[mm]$
N_3	61	Δ_{z}	0.625 $[mm]$
N_s	888	Δ_{s}	1.023 $[mm]$
N_t	32	Δ_{t}	1.096 $[mm]$
N_{views}	3,625	D_{s0}	541.0 $[mm]$
Views per rotation	984	D_{sd}	949.075 $[mm]$
Pitch	0.513	FOV	500 $[mm]$

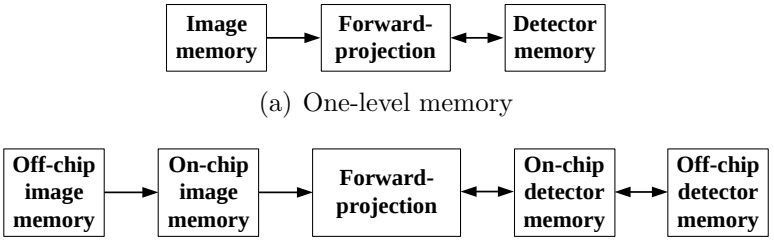
For a numerical example, substituting sample helical cone-beam geometry parameters given in Table 2.2, we get $s_{\text{bin}} = 11$ and $z_{\text{vx}} = 3$, i.e., one voxel's projection spans at most 11 detector cells along the transaxial direction, and at most 3 consecutive voxels in an axial column project to one detector cell.

2.3 Summary

Iterative algorithms for image reconstruction in X-ray CT are introduced. This chapter discusses the fixed-point quantization of forward-projection, which reduces hardware costs over the floating-point quantization and enables fast computations. Numerical simulation shows that the induced fixed-point quantization errors are tolerable in an iterative algorithm. The analysis of the projection geometry leads to the investigation of tranaxial projection and axial projection. Limited transaxial span and axial span allow for highly parallel hardware architecture. A design of a custom hardware accelerator will be discussed in the next chapter.

CHAPTER III

Parallel forward-projection architecture



(a) One-level memory

(b) Two-level memory hierarchy

Figure 3.1: High-level forward-projection architecture.

This chapter describes the design of a hardware accelerator for the forward-projection. Impacts of the fixed-point quantization used in the design are modeled and analyzed, and a quantization error bound in a diagonally preconditioned gradient descent algorithm is derived. The out-of-order sectored processing is proposed to enhance the performance of the accelerator. As a proof of concept, a 5-stage pipelined 55-way parallel forward projector supported by the sectored processing is prototyped on a Xilinx Virtex-5 FPGA.

This chapter is based in part on [18, 19, 20].

3.1 Custom architecture for acceleration

Forward- and back-projection are the core and most computationally intense building blocks of iterative image reconstruction. A simplistic forward-projection architecture includes image memory on the input and detector memory on the output as in Fig. 3.1; back-projection exchanges the positions of image and detector memory but its processing architecture is similar. In a state-of-the-art commercial CT scanner, the image and detector datasets are up to 1 GB in size. Such enormous datasets can only be accommodated in off-chip memory, and input and output data are selectively brought to on-chip memory (cache) for processing. The on-chip memory is smaller but much faster and sometimes immediately accessible by the processor, while the larger off-chip memory interface is much slower and costs a longer latency to access. Iterative image reconstruction algorithm in its original form requires moving of large datasets on and off chip constantly, resulting in a low throughput due to limited off-chip memory interface.

Parallelism can be used to improve the throughput, but it further increases memory bandwidth. The architecture can be pipelined, though its throughput is far from ideal due to loop-carried dependencies from geometry processing. In the following we investigate the projection geometry and design algorithms and architectures to reduce the memory bottleneck and improve the efficiency of parallel and pipelined architectures.

3.1.1 Loop-level parallelism

The SF forward-projection algorithm contains six layers of nested loops (2.9): β (view angle), n_1 (x index), n_2 (y index), n_3 (z index), l (t index) and k (s index) for each forward-projection. The innermost k loop computes the transaxial projection of a voxel. As discussed in the previous section, one voxel projects to a row of up to s_{bin} detector cells, each of which can be evaluated independently. Thus we exploit loop-

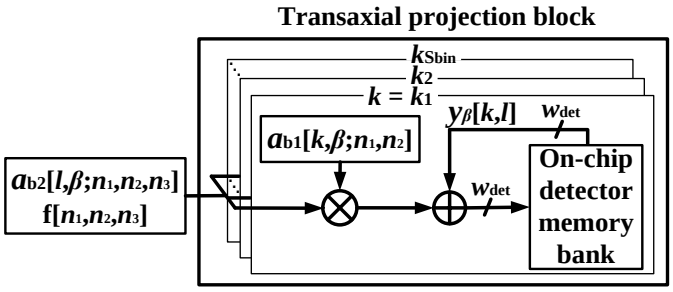


Figure 3.2: Parallel transaxial projection.

level parallelism by allocating s_{bin} multiply-accumulate (MAC) units and detector memory banks for the transaxial projection, as shown in Fig. 3.2.

The quantization study showed that the transaxial projection can be carried out in a 16-bit $\times$ 16-bit fixed-point multiply followed by a 28-bit accumulate. To operate at a high clock frequency, e.g., 200 MHz on a Xilinx Virtex-5 FPGA, we pipeline the MAC unit to 3 stages: multiply (MU), add (AD), and write back (WB). Let w_{det} be the wordlength of $y_\beta[k,l]$ that is stored in the detector memory and f_{clk} be the clock frequency, the required read and write bandwidth to the on-chip detector memory is $2w_{\text{det}}f_{\text{clk}}$ b/s. Since one complete transaxial projection block uses s_{bin} MAC units, the total on-chip detector memory bandwidth is $2s_{\text{bin}}w_{\text{det}}f_{\text{clk}}$ b/s.

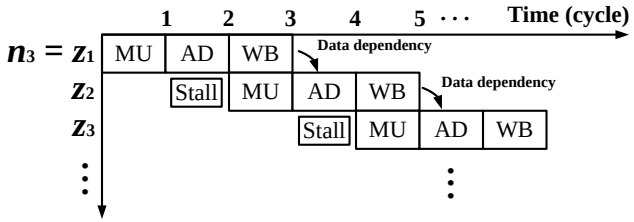


Figure 3.3: Pipeline bubbles inserted to resolve data dependencies in axial projections.

The outer l loop can be easily pipelined, but it is complicated by loop-carried dependencies: multiple voxels in an axial column can project to a single detector cell, as illustrated in Fig. 2.7, so the pipeline would have to be stalled, waiting for write back to complete before next add. The 3-stage pipeline chart in Fig. 3.3 shows that one pipeline bubble is necessary to resolve data dependency. A deeper pipeline will

result in more stalls.

3.1.2 Water-filling

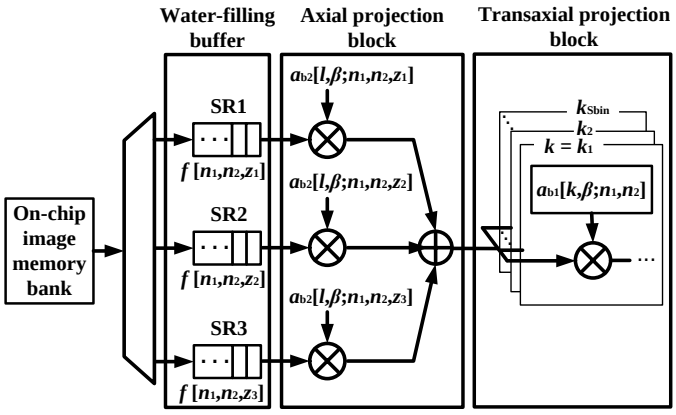


Figure 3.4: Water-filling buffer and partially-unrolled axial projection.

The mismatch between the voxel grid and detector grid requires the joint consideration between the n_3 loop and the l loop. To eliminate loop-carried dependencies, we propose an algorithm transformation to merge the two loops. In the transformed algorithm, for each l-th detector cell, we identify the group of contiguous voxels along the axial column that project to the cell and sum up the contributions. In particular, we allocate z_{vx} shift registers, each providing one candidate voxel (because up to z_{vx} voxels in an axial column project to a single detector cell), as in Fig. 3.4. Each candidate voxel is multiplied by its axial footprint and the contributions are summed, which is equivalent to a partial unrolling of the n_3 loop.

Table 3.1: Pipeline Stall Rate versus Shift Register Length of the Water-Filling Buffer

Shift register length	Stall rate (%)
1	9.70
2	7.42
3	5.65
4	4.36
5	3.48

Note that in the above example, one new voxel is brought in the water-filling buffer

every cycle to support the average input consumption rate. The average consumption rate is one input per clock cycle because Δ_z and Δ_t are designed to be matched as previously described. However, the actual input consumption varies every cycle and prefetching is needed to avoid stalling the pipeline. A longer shift register and prefetching guarantee a lower stall rate, but increase latency and resource usage. We experimentally verified the stall rate versus shift register length, and the results are listed in Table 3.1. We choose a 2-stage shift registers in our prototype design for a stall rate $P_{\text{stall}} = 7.42\%$. A lower stall rate is possible with longer shift registers.

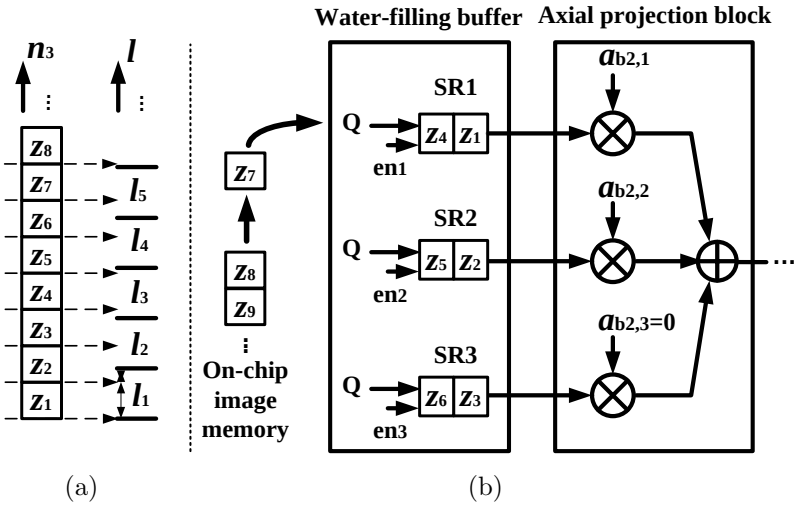


Figure 3.5: Example showing (a) n_3 and l grid mismatch, and (b) the corresponding water-filling buffering scheme.

An example is shown in Fig. 3.5 using 2-stage shift registers and input prefetching. Initially, $l = l_1$, voxels z_1 and z_2 project to detector cell l_1. A controller sets $a_{\text{b2},1} = a_{\text{b2}}[l_1, \beta; n_1, n_2, z_1]$, $a_{\text{b2},2} = a_{\text{b2}}[l_1, \beta; n_1, n_2, z_2]$ and $a_{\text{b2},3} = 0$, respectively. The contributions by voxels z_1 and z_2 to the axial projection are summed, followed by transaxial projection. Next, $l = l_2$, voxels z_2 and z_3 project to detector cell l_2. The controller sets $en_1 = 1$, $en_2 = 0$, $en_3 = 0$ to pop z_1 and keep z_2 and z_3. Now the water level in $SR1$ has dropped and the input multiplexer will direct the new voxel input z_7 to $SR1$.

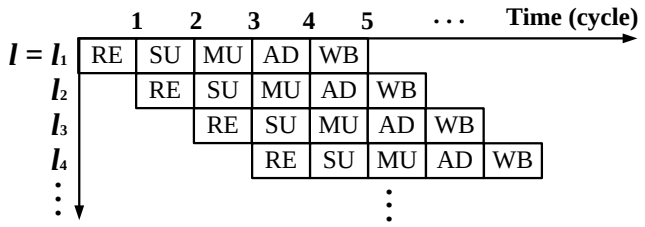


Figure 3.6: Pipeline chart for the complete forward-projection module.

The new water-filling architecture can be implemented using 3 MAC units that are pipelined in two stages: read (RE) and sum (SU), which augment the 3-stage pipeline in Fig. 3.3 to 5 stages as in Fig. 3.6. Pipeline bubbles due to loop-carried dependencies have been eliminated to achieve an average throughput of $f_{\text{clk}}(1-P_{\text{stall}})$ voxel projections/s. The required on-chip image memory bandwidth is $w_{\text{img}}f_{\text{clk}}$ b/s with w_{img} as the voxel wordlength. Substituting parameters from Table 2.2, $P_{\text{stall}} = 7.42\%$, and $f_{\text{clk}} = 200$ MHz that is typical of an FPGA platform, the proposed projection module completes 185.2 million voxel projections/s and requires an on-chip image memory bandwidth of 2.6 Gb/s and detector memory bandwidth of 123.2 Gb/s. In the following section, we propose out-of-order scheduling to reduce the detector memory bandwidth.

Table 3.2: FPGA Resource Utilization of a Forward-Projection Module based on XILINX Virtex-5 XC5VLX155T Device

	Usage	Utilization ratio
FPGA slice register	10,419	10%
FPGA slice LUT	9,124	9%
Occupied FPGA slice	5,119	21%
BRAM	37	17%
DSP48E	17	13%

A complete forward-projection module consisting of the water-filling axial projection and parallel transaxial projection has been synthesized on a Xilinx Virtex-5 XC5VLX155T FPGA and the device usage is listed in Table 3.2.

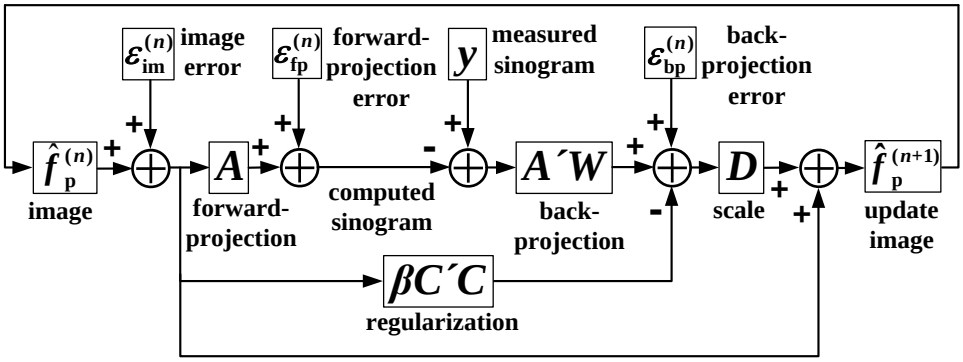


Figure 3.7: Iterative image reconstruction with perturbed forward-projection, back-projection and image update.

3.2 Impact of fixed-point quantization

Fixed-point calculations can substantially reduce the computational load, but also increase quantization error. To investigate the effect of fixed-point quantization, we analyze the error propagation after introducing perturbation in a diagonally preconditioned gradient descent algorithm for X-ray computed tomography. The effects of the quantization error in forward-projection, back-projection, and image update are calculated using the open loop and loop gain of the iterative algorithm. We derive an analytical upper bound on the quantization error variance of the reconstructed image and show that the quantization step size can be chosen to meet a given upper bound. The analytical results are confirmed by numerical simulations.

3.2.1 Perturbation-based analysis

This section analyzes the effect of perturbation in iterative image reconstruction and show that both the maximum and the mean error variance in an image update are bounded for a given level of uniform white noise. Hereafter, we define $\hat{f}_{\mathrm{p}}^{(n)}$ as the n^{th} image update of the perturbed iterative algorithm, and $e^{(n)}$ as the corresponding image error relative to the unperturbed version $\hat{f}^{(n)}$, i.e., $e^{(n)} = \hat{f}^{(n)} - \hat{f}_{\mathrm{p}}^{(n)}$.

3.2.1.1 Perturbation of forward-projection

We proceed by first perturbing the forward-projection to model the effect of fixed-point quantization [67]. We add a random error vector, $\varepsilon_{\text{fp}}^{(n)}$, to the ideal forward-projection, as illustrated in Fig. 3.7. We further assume that the error samples are uncorrelated. Specifically, we assume

$$\varepsilon_{\text{fp}}^{(n)} \sim U\left[-\frac{\Delta_{\text{fp}}}{2}, \frac{\Delta_{\text{fp}}}{2}\right], \;\; \text{cov}(\varepsilon_{\text{fp}}^{(i)}, \varepsilon_{\text{fp}}^{(i)}) = \frac{\Delta_{\text{fp}}^2}{12} I, \;\; \text{cov}\left(\varepsilon_{\text{fp}}^{(i)}, \varepsilon_{\text{fp}}^{(j)}\right) = 0 \qquad \forall i, \; \forall j, \; i \neq j, \tag{3.1}$$

where Δ_{fp} denotes the quantization step size.

From (2.3), the first image update of the perturbed algorithm can be written as

$$\begin{aligned} \hat{f}_{\text{p}}^{(1)} &= \hat{f}^{(0)} + D\left[A'W\left(y - \left(A\hat{f}^{(0)} + \varepsilon_{\text{fp}}^{(0)}\right)\right) - \beta C'C\hat{f}^{(0)}\right] \\ &= \hat{f}^{(1)} + K_{\text{fp}}\varepsilon_{\text{fp}}^{(0)}, \end{aligned} \tag{3.2}$$

where $K_{\text{fp}} \triangleq -DA'W$ is the open loop gain of the error due to perturbation in forward-projection. Similarly, we have the second image update as

$$\hat{f}_{\text{p}}^{(2)} = \hat{f}_{\text{p}}^{(1)} + D\left[A'W\left(y - \left(A\hat{f}_{\text{p}}^{(1)} + \varepsilon_{\text{fp}}^{(1)}\right)\right) - \beta C'C\hat{f}_{\text{p}}^{(1)}\right]. \tag{3.3}$$

Substituting (3.2) into (3.3) and simplification yields

$$\hat{f}_{\text{p}}^{(2)} = \hat{f}^{(2)} + MK_{\text{fp}}\varepsilon_{\text{fp}}^{(0)} + K_{\text{fp}}\varepsilon_{\text{fp}}^{(1)},$$

where $M \triangleq I - D(A'WA + \beta C'C)$ is the loop gain of the error in this iterative method. (Note that M is related to the Hessian of the cost function [63], which is given by $H = A'WA + \beta C'C$). By induction, the image update of the n^{th} iteration and the

image update error are given by

$$\hat{f}_{\mathrm{p}}^{(n)} = \hat{f}^{(n)} + e^{(n)}$$
$$e^{(n)} = \sum_{k=0}^{n-1} M^k K_{\mathrm{fp}} \varepsilon_{\mathrm{fp}}^{(n-1-k)}.$$

Using (3.1), the mean of $e^{(n)}$ is zero, and the covariance is

$$\mathrm{cov}\left(e^{(n)}, e^{(n)}\right) = \frac{\Delta_{\mathrm{fp}}^2}{12} \sum_{k=0}^{n-1} M^k K_{\mathrm{fp}} K_{\mathrm{fp}}' \left(M^k\right)'. \tag{3.4}$$

Note that a covariance matrix is positive semidefinite[68], and its eigenvalues are nonnegative[69]. Thus, an upper bound on the error variance is the maximum eigenvalue, i.e., spectral radius, of the covariance matrix of $e^{(n)}$. Evaluating the spectral radius is nontrivial due to the term M^k. Since matrix D is a real diagonal matrix with positive diagonal entries, we can decompose M as

$$M = I - DH = D^{\frac{1}{2}} \left(I - D^{\frac{1}{2}} H D^{\frac{1}{2}}\right) D^{-\frac{1}{2}}.$$

The Hessian matrix H is a nonnegative definite and so is $I - D^{\frac{1}{2}} H D^{\frac{1}{2}}$, by the design of D. Thus by the spectral theorem [70], there exists a unitary matrix U and a diagonal matrix Σ such that $I - D^{\frac{1}{2}} H D^{\frac{1}{2}} = U \Sigma U'$. Then M becomes

$$M = D^{\frac{1}{2}} U \Sigma U' D^{-\frac{1}{2}}. \tag{3.5}$$

Similarly, $(A'W)(A'W)'$ is also nonnegative definite and can be decomposed using a unitary matrix V and a nonnegative diagonal matrix F[70]. It follows that

$$K_{\mathrm{fp}} K_{\mathrm{fp}}' = (-DA'W)(-DA'W)' = D(VFV')D. \tag{3.6}$$

Substituting (3.5) and (3.6) into (3.4), we have

$$\mathrm{cov}\left(e^{(n)}, e^{(n)}\right) = \frac{\Delta_{\mathrm{fp}}^2}{12} \sum_{k=0}^{n-1} D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} \left(VFV'\right) D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}}.$$

Thus the spectral radius equals the 2-norm and its upper bound can be derived using the matrix norm property that $\parallel AB \parallel \leq \parallel A \parallel \parallel B \parallel$ [71]. After considerable simplification, we have

$$\begin{aligned}
\rho\left(\mathrm{cov}\left(e^{(n)}, e^{(n)}\right)\right) &= \max_{x:\|x\|=1} \left(x' \mathrm{cov}\left(e^{(n)}, e^{(n)}\right) x\right) \\
&= \frac{\Delta_{\mathrm{fp}}^2}{12} \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} \parallel F^{\frac{1}{2}} V' D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} x \parallel^2 \right) \\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12} \rho(F) \rho(D^2) \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma^2)}. \qquad (3.7)
\end{aligned}$$

For the detailed derivation of (3.7), please see (A.1).

The spectral radius of the covariance matrix measures the maximum error variance in the n^{th} iteration. i.e., $\sigma^2_{(n)\max} \triangleq \rho\left(\mathrm{cov}\left(e^{(n)}, e^{(n)}\right)\right)$. Next, we analyze the mean error variance in an image update, which is related to the trace, or sum of diagonal entries, of the covariance matrix, i.e., $\sigma^2_{(n)\mathrm{mean}} \triangleq \mathrm{tr}\left(\mathrm{cov}\left(e^{(n)}, e^{(n)}\right)\right)/n_{\mathrm{v}}$, where n_{v} is the number of diagonal entries in the covariance matrix. Using the property that $\mathrm{tr}(AB) \leq \rho(B)\mathrm{tr}(A)$ [70], we have

$$\begin{aligned}
\mathrm{tr}\left(\mathrm{cov}\left(e^{(n)}, e^{(n)}\right)\right) &= \frac{\Delta_{\mathrm{fp}}^2}{12} \mathrm{tr}\left(\sum_{k=0}^{n-1} D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} \left(VFV'\right) D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} \right) \\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12} \sum_{k=0}^{n-1} \rho\left(\Sigma^k\right) \mathrm{tr}\left(D^{\frac{1}{2}} \left(VFV'\right) D^{\frac{1}{2}} U \Sigma^k U' D \right) \\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12} \mathrm{tr}\left(D \left(VFV'\right) D\right) \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma^2)}. \qquad (3.8)
\end{aligned}$$

For the detailed derivation of (3.8), please see (A.2). To guarantee the convergence of

iterative reconstruction algorithm, the matrix D is always selected such that $D^{-1} \succ H$, i.e., $D^{-1} - H$ is positive definite, which implies $\rho(I - DH) < 1$, where H is the Hessian of the cost function [63]. It follows that

$$\begin{aligned}\rho(\Sigma) &= \rho(U'(I - D^{\frac{1}{2}} H D^{\frac{1}{2}})U) = \rho(I - D^{\frac{1}{2}} H D^{\frac{1}{2}}) \\ &= \rho(D^{-\frac{1}{2}}(I - DH)D^{\frac{1}{2}}) = \rho(I - DH) < 1.\end{aligned}$$

In steady state as $n \to \infty$, the upper bounds of (3.7) and (3.8) become

$$\begin{aligned}\sigma^2_{(n)\max} &\leq \rho(\operatorname{cov}(e^{(\infty)}, e^{(\infty)})) \leq \frac{\Delta^2_{\text{fp}}}{12} \frac{\rho(D^2)\rho(F)}{1 - \rho(\Sigma^2)}, \\ \sigma^2_{(n)\text{mean}} n_{\text{v}} &\leq \operatorname{tr}(\operatorname{cov}(e^{(\infty)}, e^{(\infty)})) \leq \frac{\Delta^2_{\text{fp}}}{12} \frac{\operatorname{tr}(D(VFV')D)}{1 - \rho(\Sigma^2)}.\end{aligned}$$

Therefore, both the maximum and the mean error variance of an image update are bounded. For example, given $\epsilon > 0$, if we choose Δ_{fp} such that

$$\Delta_{\text{fp}} < \sqrt{\frac{12\,(1 - \rho(\Sigma)^2)}{\rho(D^2)}} \sqrt{\frac{1}{\rho(F)}} \sqrt{\epsilon},$$

then

$$\sigma^2_{(\infty)\max} < \epsilon, \qquad \sigma^2_{(\infty)\text{mean}} < \epsilon.$$

The result implies that we can make the error due to perturbation in forward-projection arbitrarily small for this algorithm by choosing an appropriate quantization step size, provided quantization noise can be modeled as in (3.1).

3.2.1.2 Perturbation of forward- and back- projection, and image update

Following the derivation from the previous section, we can also model the effect of fixed-point quantization in the back-projection and image update by injecting uniform white noises $\varepsilon^{(n)}_{\text{bp}}$ and $\varepsilon^{(n)}_{\text{im}}$, as indicated in Fig. 3.7. Similar to (3.1), we make

the following assumptions:

$$\varepsilon_{\text{bp}}^{(n)} \sim U\left[-\frac{\Delta_{\text{bp}}}{2}, \frac{\Delta_{\text{bp}}}{2}\right],\ \varepsilon_{\text{im}}^{(n)} \sim U\left[-\frac{\Delta_{\text{im}}}{2}, \frac{\Delta_{\text{im}}}{2}\right], \tag{3.9}$$

where Δ_{bp} and Δ_{im} denote the quantization step sizes of back-projection and image update respectively.

Similar to (3.2), we can express the perturbed image update of the first iteration as

$$\begin{aligned}\hat{f}_{\text{p}}^{(1)} =&(\hat{f}^{(0)} + \varepsilon_{\text{im}}^{(0)}) + D[A'W(y - (A(\hat{f}^{(0)} + \varepsilon_{\text{im}}^{(0)}) + \varepsilon_{\text{fp}}^{(0)})) + \varepsilon_{\text{bp}}^{(0)} - \beta C'C(\hat{f}^{(0)} + \varepsilon_{\text{im}}^{(0)})] \\ =&\hat{f}^{(1)} + K_{\text{fp}}\varepsilon_{\text{fp}}^{(0)} + K_{\text{bp}}\varepsilon_{\text{bp}}^{(0)} + M\varepsilon_{\text{im}}^{(0)},\end{aligned}$$

where $K_{\text{bp}} \triangleq D$ is the open loop gain of the error due to perturbation in back-projection. It follows that the image update error in the n^{th} iteration is

$$e^{(n)} = \sum_{k=0}^{n-1}(M^k(K_{\text{fp}}\varepsilon_{\text{fp}}^{(n-1-k)} + K_{\text{bp}}\varepsilon_{\text{bp}}^{(n-1-k)} + M\varepsilon_{\text{im}}^{(n-1-k)})).$$

We assume independence of the three noise vectors. Using (3.1), (3.5), (3.6), and (3.9), the mean of $e^{(n)}$ is zero, and the covariance can be written as

$$\begin{aligned}&\text{cov}\left(e^{(n)}, e^{(n)}\right) \\ &= \frac{\Delta_{\text{fp}}^2}{12}\sum_{k=0}^{n-1}(M^k K_{\text{fp}} K_{\text{fp}}'(M^k)') + \frac{\Delta_{\text{bp}}^2}{12}\sum_{k=0}^{n-1}(M^k K_{\text{bp}} K_{\text{bp}}'(M^k)') + \frac{\Delta_{\text{fp}}^2}{12}\sum_{k=0}^{n-1}(M^{k+1}(M^{k+1})').\end{aligned} \tag{3.10}$$

Following a similar approach as in the previous section, we can derive the upper

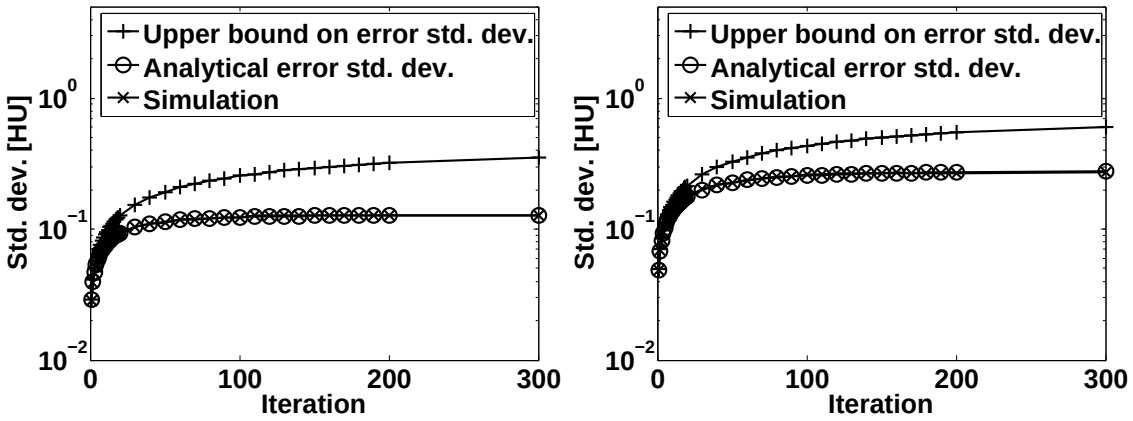


Figure 3.8: Theoretical bound and numerical simulation of standard deviation of the image updates : (a) forward-projection with the quantization step size of $\Delta_{\text{fp}} = 2^7[\text{HU}\times\text{mm}]$ (b) forward-projection, back-projection, and image update with $\Delta_{\text{fp}} = 2^7[\text{HU}\times\text{mm}]$, $\Delta_{\text{bp}} = 2^{15}[\text{mm}]$, $\Delta_{\text{im}} = 2^{-3}[\text{HU}]$.

bounds on the spectral radius and the trace of the covariance matrix.

$$\rho(\text{cov}(e^{(\infty)}, e^{(\infty)})) \leq \frac{\Delta_{\text{fp}}^2}{12}\frac{\rho(D^2)\rho(F)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{bp}}^2}{12}\frac{\rho(D^2)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{im}}^2}{12}\frac{\rho(\Sigma^2)\rho(D)\rho(D^{-1})}{1-\rho(\Sigma^2)}, \tag{3.11}$$

and

$$\text{tr}(\text{cov}(e^{(\infty)}, e^{(\infty)})) \leq \frac{\Delta_{\text{fp}}^2}{12}\frac{\text{tr}(VFV'D^2)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{bp}}^2}{12}\frac{\text{tr}(D^2)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{im}}^2}{12}\frac{\rho(\Sigma^2)\text{tr}(I)}{1-\rho(\Sigma^2)}. \tag{3.12}$$

For the detailed derivation of (3.11) and (3.12), please see (A.6) and (A.10), respectively.

Therefore, both the maximum and the mean error variance of the reconstructed image are bounded after considering perturbation in forward-projection, back-projection, and image update. The error can be made arbitrarily small for this algorithm by choosing an appropriate quantization step size.

3.2.2 Simulation results

To verify the quantization error analysis, we performed numerical simulations of an iterative reconstruction of a 40×40×4 test object in an axial cone-beam arc-detector X-ray CT system with a detector size of 170×10 over 90 projection views. The PWLS diagonally preconditioned gradient descent algorithm (2.3) was simulated with a quadratic roughness regularizer. We evaluated analytical quantization error variance and its upper bound in each iteration, which are compared to measured quantization error variance from simulations by injecting uniformly distributed error vectors that correspond to quantization step sizes of $\Delta_{\text{fp}} = 2^7$[HU×mm], $\Delta_{\text{bp}} = 2^{15}$[mm], and $\Delta_{\text{im}} = 2^{-3}$[HU]. Fig. 3.8 shows the standard deviation of the image update error due to (a) perturbation in forward-projection alone and (b) perturbation in forward-projection, back-projection, and image update. The measured standard deviation matches the analytical standard deviation and stays below the upper bound. Due to limited space, we only show one set of quantization step size, but alternative choices could be equally used. Both the analytical and simulation results in Fig. 3.8 point to the conclusion that the error variance of image updates converges to a fixed level after a sufficient number of iterations. Note that evaluating the analytical error variance is not feasible for large object sizes.

3.3 Algorithm rescheduling

The architecture can further parallelize the n_1 and n_2 loops, but it would increase the memory bandwidth. Absent of any temporal locality of reference, the off-chip memory bandwidth will be easily saturated as we continue to parallelize. To circumvent the difficulty, we compress the off-chip memory bandwidth by an out-of-order access schedule that maximizes the temporal locality of reference.

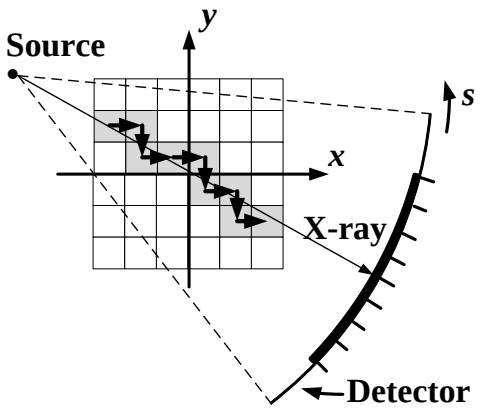


Figure 3.9: Top view of the forward-projection following an X-ray.

3.3.1 Out-of-order scheduling

To explain the sectored processing, note that the voxels along a line cast projections onto the same block of detector cells, thus the on-chip memory can be reused without resorting to off-chip access, as shown in Fig. 3.9. Based on this observation, we design an out-of-order scheduling algorithm as follows: (1) divide the detector into sectors as in Fig. 3.10(a); (2) draw the upper and lower edge of each sector by connecting the X-ray source and the upper and lower end of the sector; (3) determine the set of voxels whose projections lie entirely in each sector. Assign the set of voxels to a projection module for processing to maximize the detector memory's locality of reference.

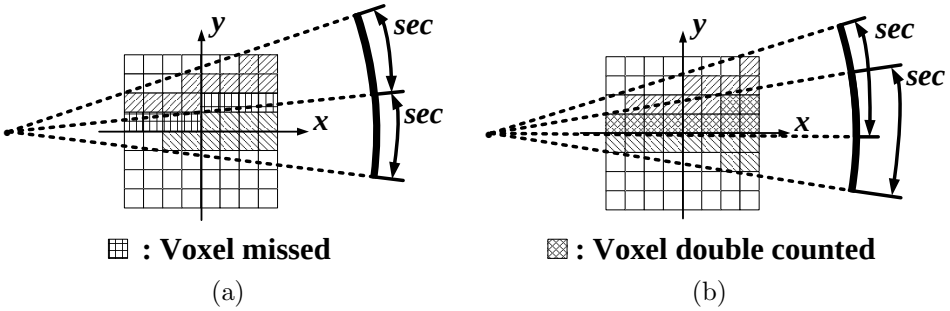
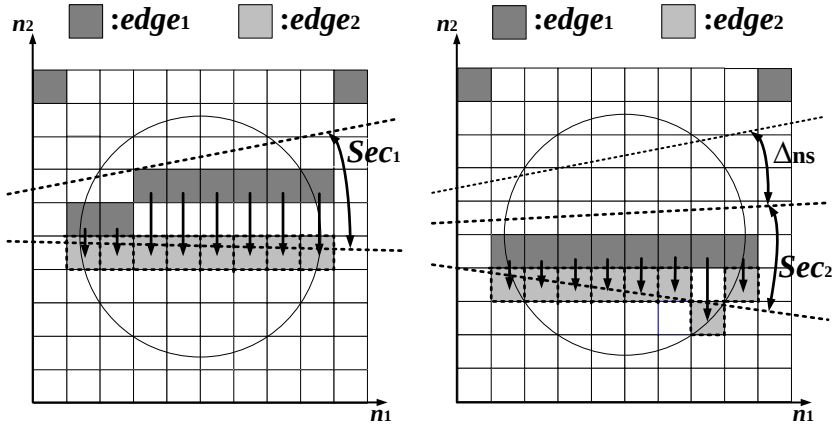


Figure 3.10: Illustrations showing (a) non-overlapping sectors, and (b) overlapping sectors.

If we choose the sectors to be non-overlapping as in Fig. 3.10(a), some voxels will

be missed as their projections do not completely lie in any sector. Adjacent sectors will have to overlap by an amount $(s_{\text{bin}} - 1)\Delta_{\text{s}}$ to ensure all voxels are accounted for. (Recall that s_{bin} is the maximum transaxial span of a voxel's projection. An overlap of $s_{\text{bin}}\Delta_{\text{s}}$ or more is not necessary.) For simplicity of implementation, we choose a fixed overlap of $(s_{\text{bin}} - 1)\Delta_{\text{s}}$ in making sectors. Now another problem arises with the choice of a fixed overlap, as some voxels will be counted twice in adjacent sectors, as shown in Fig. 3.10(b). To avoid double counting, we keep track of the upper and lower edge of each sector.

The out-of-order schedule can be computed in design time and stored in memory. The required memory is $w_{\text{coord}} N_{\text{x}} N_{\text{y}} N_{\text{views}}$, where w_{coord} is the wordlength to store the (x, y) coordinate pair. Using the sample geometry in Table 2.2, the out-of-order schedule memory takes 796.5 MB. If we take into account the multiple rotations in a CT scan that repeat view angles and only voxels inside the FOV, the out-of-order schedule memory size is reduced to 86.3 MB, which is still significant.



(a) 1^{st} scan (b) 2^{nd} scan

Figure 3.11: Illustration of run-length encoding of access schedule.

To further shrink the out-of-order schedule memory, we design a run-length encoding to compress the schedule. The encoding scheme is illustrated in Fig. 3.11: we store the voxel coordinates along $edge_1$ of Sec_1, and encode and store $edge_2$ of Sec_1

Table 3.3: Moving Directions for Run-Length Encoding

View : β (rad)	Direction
$\frac{\pi}{4} \leq \beta \leq \frac{3\pi}{4}$	-n_2
$\frac{3\pi}{4} \leq \beta \leq \frac{5\pi}{4}$	+n_1
$\frac{5\pi}{4} \leq \beta \leq \frac{7\pi}{4}$	+n_2
$0 \leq \beta \leq \frac{\pi}{4}$ or $\frac{7\pi}{4} \leq \beta \leq 2\pi$	-n_1

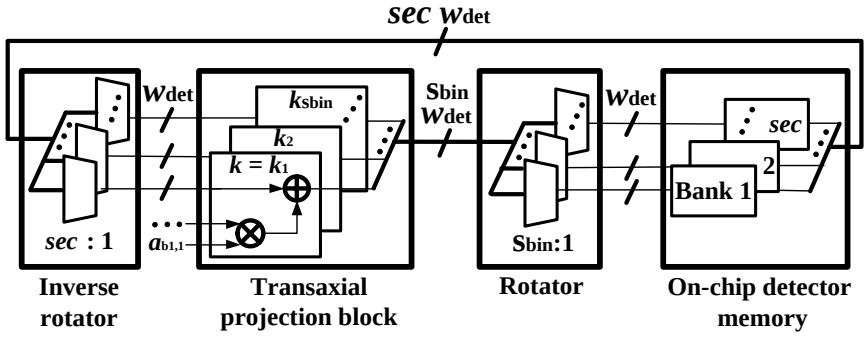
as the run length from $edge_1$. $edge_2$ of Sec_1 becomes the $edge_1$ of Sec_2 and the $edge_2$ encoding follows a similar fashion. The direction to count run length depends on the view angle β, as described in Table 3.3. For a numerical example, if we choose a sector size of $sec = 20$, the out-of-order schedule memory can be compressed by an order of magnitude to 8 MB.

Table 3.4: Sector Choice for Out-of-Order Scheduling

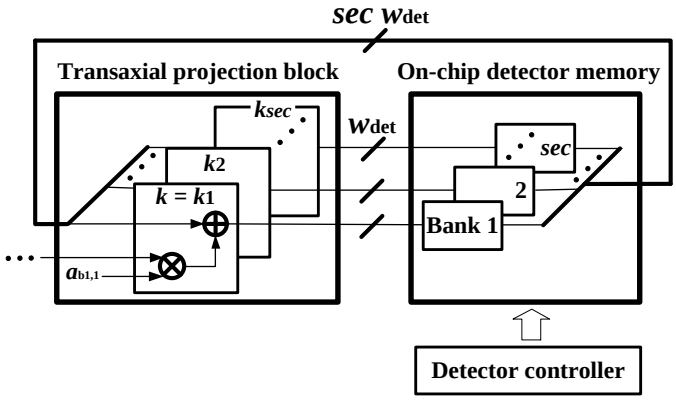
sec	N_{sec}	Δ_{ns}	$N_{\text{vx,min}}$	$N_{\text{vx,max}}$	N_{vx}	On-chip memory (Kb)	$\Delta_{\text{ns}}/s_{\text{bin}}$ $/N_{\text{vx}}$	Off-chip BW (Mb/s)	Schedule memory (Mb)
14	222	4	41	341	183	15.75	0.00199	245.17	98.85
16	148	6	61	457	274	19.25	0.00199	245.17	67.05
18	111	8	96	572	365	22.75	0.00199	245.17	75.00
20	89	10	112	681	456	26.25	0.00199	245.17	60.82
30	45	20	189	1245	903	43.75	0.00201	247.63	42.12
40	30	30	330	1855	1355	61.25	0.00201	247.63	29.23
50	23	40	170	2401	1767	78.75	0.00206	253.79	28.15

Table 3.4 lists a few more example sector sizes based on the geometry in Table 2.2. If we choose sector size $sec = 20$, with a fixed sector-sector overlap of $s_{\text{bin}} - 1$ =10, the detector is divided into 89 sectors. A sector covers an average of $N_{\text{vx}} = 456$ voxel columns. Sectors are processed sequentially. After finishing one sector, we move forward by a stride of $\Delta_{\text{ns}} = sec - (s_{\text{bin}} - 1) = 10$ to the new sector. The external memory access is reduced to only Δ_{ns} banks every sector. When $sec = 20$, the off-chip detector memory bandwidth of the proposed projection module described in the previous section is reduced to 245.2 Mb/s. As we increase the sector size, both the stride Δ_{ns} and sector coverage N_{vx} increase, resulting in an almost constant off-chip memory bandwidth. A larger sector size requires a larger on-chip memory but a

smaller out-of-order schedule memory.



(a) Rotator-based architecture



(b) Selector-based architecture

Figure 3.12: Architectures supporting sectored processing.

The out-of-order scheduling requires sectored processing. The number of on-chip detector memory banks has to be increased from s_{bin} to *sec*. Since a projection covers only an s_{bin} segment of the sector, a rotator and an inverse rotator are needed to select the detector memory banks. The rotator-based architecture can be implemented using multiplexers and it incurs a high routing overhead. An alternative selector-based architecture allocates *sec* transaxial projection blocks, and each block can be enabled or disabled by the write enable to the corresponding memory bank. The comparison between the rotator-based and the selector-based architecture is illustrated in Fig. 3.12 with FPGA synthesis results listed in Table 3.5. A selector-based architecture uses fewer logic units or FPGA slices, but more MAC units or DSP48E slices. In both

Table 3.5: FPGA Resource Utilization of a Forward-Projection Module Supporting Sectored Processing based on XC5VLX155T Device

	$sec = 14$		$sec = 20$	
	Rotator	Selector	Rotator	Selector
FPGA slice register	11,120	11,487	11,494	12,250
FPGA slice LUT	12,335	11,060	15,028	11,809
Occupied FPGA slice	6,347	6,166	7,132	6,522
BRAM	39	39	45	45
DSP48E	17	20	17	26

architectures, a small sector size results in more efficient use of hardware.

The detector memory is dual-port to support one read and one write per cycle for the read-accumulate-write operation. To enable loading and unloading from off-chip memory without stalling the computation, we increase the number of detector memory banks from sec to $sec + \Delta_{\text{ns}}$. While sec memory banks are accessed for the projection of the current sector, the remaining Δ_{ns} banks are being unloaded/loaded to/from off-chip memory. To avoid stalling the pipeline, the loading and unloading time by the Δ_{ns} memory banks should be no greater than the time spent on the projection computation. This condition can be easily met in the proposed sectored processing.

3.3.2 FPGA implementation

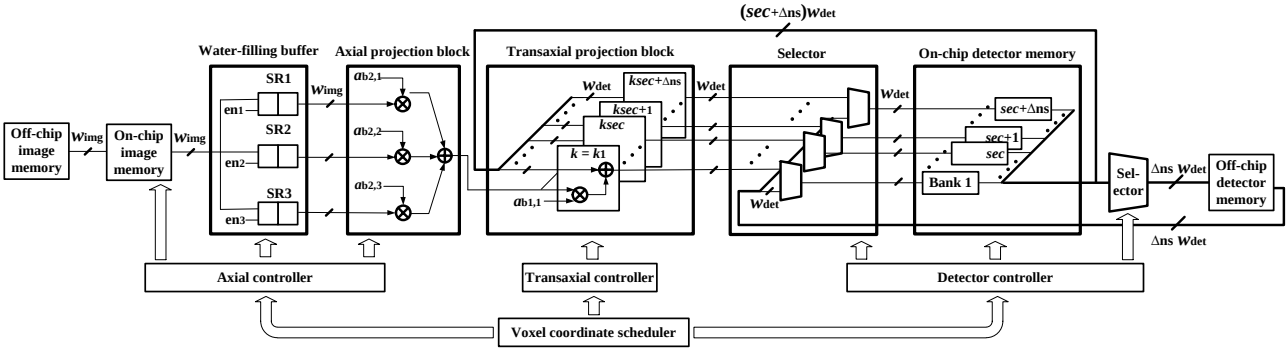


Figure 3.13: Complete selector-based forward-projection module supporting sectored processing.

Table 3.6: Architecture Metrics of a Forward-Projection Module Supporting Sectored Processing

On-chip image memory bandwidth	$w_{\text{img}} f_{\text{clk}}$ [b/s]
Off-chip image memory bandwidth	$w_{\text{img}} f_{\text{clk}}$ [b/s]
On-chip detector memory bandwidth	$2 s_{\text{bin}} w_{\text{det}} f_{\text{clk}}$ [b/s]
Off-chip detector memory bandwidth	$2 \Delta_{\text{ns}} w_{\text{det}} f_{\text{clk}} / N_{\text{vx}}$ [b/s]
On-chip image memory banks	1
On-chip detector memory banks	$sec + \Delta_{\text{ns}}$
MAC units	$sec + \Delta_{\text{ns}} + z_{\text{vx}}$
Throughput	$f_{\text{clk}}(1 - P_{\text{stall}})$ [voxel projs/s]

Table 3.7: FPGA Resource Utilization of Complete Forward-Projection Modules based on XILINX Virtex-5 XC5VLX155T Device

	single module		5× parallel modules	
	Usage	Utilization ratio	Usage	Utilization ratio
FPGA slice register	12,077	12%	30,323	31%
FPGA slice LUT	11,939	12%	31,874	32%
Occupied FPGA slice	6,328	26%	14,243	58%
BRAM	43	20%	117	55%
DSP48E	24	18%	108	84%

A complete forward-projection module is shown in Fig. 3.13. Inputs are read from the image memory, held by the water-filling buffer before being processed by the partially-unrolled axial projection block. Transaxial projections are performed in parallel and the results are accumulated in the detector memory. A selector-based architecture orchestrates sectored processing following an out-of-order schedule. A summary of the architecture metrics is listed in Table 3.6.

The projection module has been mapped to a Xilinx Virtex-5 XC5VLX155T FPGA [72] and the device utilization is listed in Table 3.7. We followed the sample geometry in Table 2.2 and chose a small sector size $sec = 14$ with $\Delta_{\text{ns}} = 4$. The projection module uses 24 DSP48E slices as MAC units, 43 block RAMs as on-chip memory banks, and occupies 6,328 FPGA slices. Note that the resource usage includes a fixed overhead created to handle interfaces to the FPGA and controls. At a 200 MHz clock frequency, the off-chip input image memory bandwidth is 2.6 Gb/s and the off-chip output detector memory bandwidth is compressed to 245.2 Mb/s.

Additional memory access is needed to load the out-of-order schedule, but the bandwidth is very low as only one pair of coordinates is read per column of voxels and the coordinates have been compressed using run-length encoding. The projection module is fully pipelined and capable of completing up to $s_{\text{bin}} = 11$ projections per clock cycle for an average throughput of 185.2 million voxel projections/s at $f_{\text{clk}} = 200$ MHz.

The substantially reduced off-chip memory bandwidth allows us to parallelize the design further by multiple projection modules. The Xilinx Virtex-5 XC5VLX155T FPGA can accommodate 5 parallel projection modules, and the device utilization is shown in Table 3.7. The parallel projection modules will be assigned to non-adjacent sectors, so they will be able to operate independently for a 55-way parallel computation towards a combined average throughput of 925.8 million voxel projections/s at $f_{\text{clk}} = 200$ MHz. The 55-way parallel forward-projector is integrated with two DDR400 64-bit DRAM channels that each provides up to 25.6 Gb/s off-chip memory interface. One DRAM channel is used as the off-chip image memory and the other as the off-chip detector memory. This 55-way parallel design completes one forward-projection of a 320×320×61 test object over 3,625 views in 6.31 seconds. The same task implemented in C requires 31.1 seconds of execution time on an 8-core 2.8-GHz Intel processor for a throughput of 203.0 million voxel projections/s. The C program uses 16 threads, and is optimized based on the projection geometry.

3.4 Summary

This chapter implements a forward-projector for fast iterative image reconstruction. A custom designed forward-projection is enabled by studying the projection geometry and the proposed water-filling buffer. An 11-way parallel multiply-and-accumulate (MAC) is used to match the maximum transaxial span of voxel projections to compute the transaxial projection efficiently. The water-filling buffer resolves pipeline stalls caused by geometric mismatch between voxel grid and detector grid.

The water-filling buffer is implemented using 3 shift registers and they can provide 3 voxels concurrently to compute voxel contributions to the axial projection. The axial and transaxial projection modules are concatenated to enable a 5-stage pipelined 11-way parallel forward-projection processing.

To analyze the impacts of the fixed-point quantization on algorithm performance, we evaluated quantization error variance and its upper bound in each iteration of the PWLS diagonally preconditioned gradient descent algorithm (2.3), which are compared to measured quantization error variance from simulations by injecting uniformly distributed error vectors. Both the analytical and simulation results point to the conclusion that the error variance of image updates converges to a fixed level after a sufficient number of iterations.

We propose an out-of-order sectored processing to enhance the performance of the forward-projector. The sectored processing exploits spatial locality of reference in detector cell updates, and the off-chip detector memory bandwidth is reduced by grouping voxels whose projections are overlapped in a limited detector segment. The cost of implementing the sectored processing is kept low by judicious choices of on-chip detector memory and scheduling memory.

A 5-stage pipelined, 55-way parallel forward-projector implemented on a Xilinx Virtex-5 XC5VLX155T FPGA demonstrates an average throughput of 925.8 million voxel projections/s at a clock frequency of 200 MHz. Note that the throughput is limited by the number of MAC units available on this device, as this FPGA contains only 128 DSP48E slices. The latest Xilinx Virtex-7 devices offer up to 3,600 DSP slices [73], allowing for a much higher throughput potential. The proposed architecture can be easily adopted for back-projection for a complete iterative image reconstruction system. The proposed algorithm and architecture techniques also apply to designs that are built on alternative hardware platforms, such as GPU and DSP to achieve significant accelerations.

CHAPTER IV

Background and simulation of sparse coding

This chapter discusses the background of sparse coding which mimics human vision processing. Input stimuli first pass through the retina, and are encoded to recognizable features in the primary visual cortex. The features are called receptive fields. The receptive fields can be considered as vectors that represent signals in the input space. Each cortical neuron in the primary visual cortex has its own receptive field. The receptive fields are selectively activated depending on the features in the input stimuli. The activation of receptive fields is manifested as neuron spiking (1 or 0). Sparse coding mimics the mechanisms of the primary visual cortex. It is an important step in complex vision processing such as object recognition [9, 10].

4.1 Background of sparse coding

Much progress has been made in training unsupervised machine learning algorithms using natural images to build the receptive fields that resemble the receptive fields of the primary visual cortex. Among the most promising candidates are the sparse coding algorithms [11, 12, 13, 74, 75, 76] that learn to represent natural images using a small number of receptive fields.

The Sparsenet algorithm by Olshausen and Field [11] attempts to minimize the

This work is based in part on [31].

mean neuron activity in learning the representation of natural images, and it is shown to reproduce the receptive fields that match the key qualitative features of the receptive fields of the primary visual cortex. The sparse-set coding (SSC) network by Rehn and Sommer [74] tries to minimize the number of active neurons, and it successfully predicted the distribution of receptive field shapes found in the primary visual cortex of cat and monkey.

Sparse coding algorithms can be mapped to a biologically inspired network of computing nodes, or "neurons". Foldiak proposed a network of model neurons with feed-forward connections between neurons and stimulus, and inhibitory feedback connections between neurons [12]. The feed-forward connection weights are updated by the Hebbian rule [77] to strengthen a feed-forward connection when an input pattern matches the receptive field; and the feedback connection weights are updated by the anti-Hebbian rule to suppress correlated neuron activities and enforce sparse activation. The locally competitive algorithm (LCA) by Rozell et al. [75] is naturally mapped to a network similar to what Foldiak proposed. In Rozell's network, a model neuron's membrane potential charges up in response to input stimulus at a rate depending how well the input pattern matches the neuron's receptive field, and when the potential exceeds a threshold, the neuron emits an action potential to inhibit neighboring neurons. LCA was shown to perform the optimal sparse approximation that minimizes the mean squared error (MSE) of image reconstruction and a sparsity cost function. The parallel network of neurons implementation is appealing, but the model neurons in Foldiak's network was designed to communicate with analog signal. Also, the weight updates in Rozell's network are performed offline by costly global computations.

Recently, Zylberberg et al. proposed sparse and independent local network (SAILnet) algorithm to perform sparse coding using spiking neurons and local update rules [76]. The SAILnet algorithm was demonstrated to learn the full diversity of

the primary visual cortex simple cell receptive field shapes when trained on natural images. The SAILnet algorithm enables a fundamentally more efficient mapping to a network of spiking neurons that uses only local computation in weight updates. The spiking neural network consists of a set of interconnected simple computing neurons that communicate using binary spikes. In contrast to modern multi-core von Neumann processors that are optimized for sequential tasks and often limited by interconnect and memory bandwidth, the biologically inspired spiking neural network is an inherently parallel array of self-adapting computational units that are ideally suited for computer vision processing.

Sparse coding is operated in two phases: 1) learning and 2) inference. In the learning phase, a spiking neural network is trained using images to extract a library of features. During learning, the neural network updates the connection weights to optimally adapt to the operating environment. Training image pixels excite the neurons, which then generate binary (1 or 0) spikes [76, 12, 13] that are propagated through the neural network. When a neuron spikes, it updates its feed-forward connection weight. When a neuron sees a spike from a neighboring neuron, it updates the feedback connection weight. The updates are based on a learning rule. Upon convergence, the neural network will have finalized a library of feed-forward connection weights, resembling the receptive fields, as well as the feedback connection weights that regulate the interactions between neurons.

Learning is very compute-intensive as it involves weight updates, but learning is done infrequently – in the beginning for setting up the weights and occasionally to update the weights to accommodate changes in the operating environment. The decision to switch learning on and off is made by a controller that is external to the sparse coding processor, e.g., if the sparse coding processor is integrated as part of a vision system, the system decides whether to switch learning on or off. More specifically, when the sparse coding processor is moved to a new operating environment, e.g., a

different terrain, it needs to update the receptive fields to reflect the new operating environment. If it is not properly trained, the image fidelity will be poor. There is no real-time constraint and the learning power budget is not the most critical due to its infrequent activation.

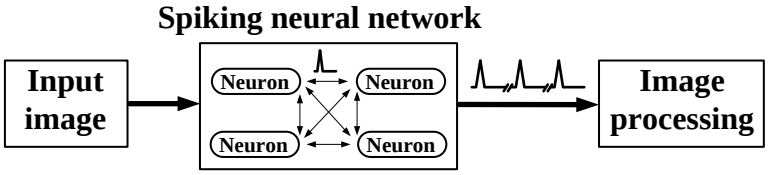


Figure 4.1: A spiking neural network for inference.

In the inference phase, the neural network receives an input image and responds by neuron spikes that correspond to the receptive fields that are activated, as illustrated in Fig. 4.1. Using a sparse coding algorithm, such as SAILnet [76], spikes will be kept very sparse, thus the neural network will be capable of encoding an image using a sparse set of receptive fields. Tasks including image reconstruction, target extraction and tracking can be performed based on the neuron firing and the receptive fields.

Inference is also compute-intensive, but to a lesser extent compared to learning, because it does not perform weight updates. However, inference needs to be done in real time. Furthermore, inference is always on and its power consumption needs to be minimized. In this paper, we develop optimized algorithm and hardware architecture to reduce the implementation cost and power consumption of a spiking neural network for inference. As inference shares the same hardware architecture as learning, the efficiency of learning is also improved.

4.1.1 Sparse and Independent Local Network (SAILnet)

In this section, we discuss the sparse and independent local network (SAILnet) algorithm. SAILnet uses spiking neurons as the fundamental computing units, and the neurons process input signals to perform inference and learning. We first introduce

a spiking neuron model, followed by an introduction of the SAILnet learning rule.

4.1.1.1 Spiking neuron model

A biological neuron in the visual cortex receives stimuli from visual inputs and other neurons in the network in the form of electrical signals. The received stimuli will increase or decrease the neuron's membrane potential. The neuron fires an action potential, or spike, when its membrane potential reaches a threshold value [78]. After firing, the neuron resets its membrane potential for the next firing.

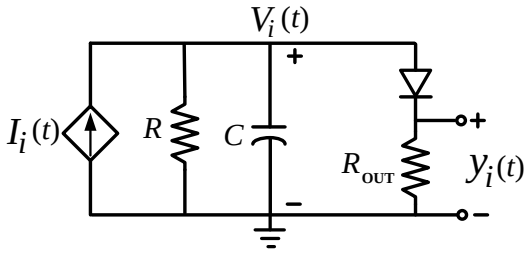


Figure 4.2: Integrate-and-fire neuron model.

Fig. 4.2 describes a simple passive or leaky integrate-and-fire (IF) neuron model, including a current source $I_i(t)$ and a parallel RC circuit [78, 79, 80]. The current source $I_i(t)$ in a neuron is determined by the inputs and the activities of other neurons in the network along with feed-forward and feedback connection weights. The current $I_i(t)$ is mathematically formulated as

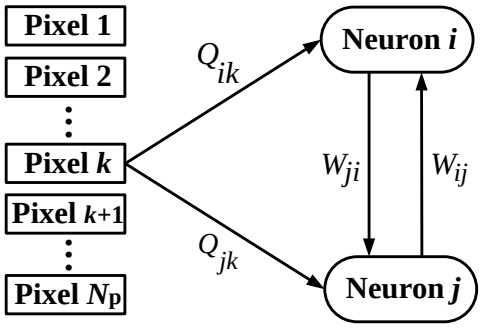


Figure 4.3: Feed-forward connection between neuron and pixel, and feedback connection between neurons.

$$I_i(t) = \frac{1}{R}\left(\sum_{k=1}^{N_p} Q_{ik}x_k - \sum_{j\neq i} W_{ij}s_j(t)\right), \tag{4.1}$$

where x_k denotes an input pixel value, $s_j(t)$ represents the spike train generated by neuron j ($s_j(t) = 1$ if neuron j fires at time t, and $s_j(t) = 0$ otherwise). Q_{ik} is the weight of the feed-forward connection between input pixel k and neuron i, and W_{ij} is the weight of the feedback connection from neuron j to neuron i, as labeled in Fig. 4.3. N_p is the number of pixels. An interpretation of (4.1) is that the input stimuli increase the current (an excitatory effect) and the neighboring neuron spikes decrease the current (an inhibitory effect).

The voltage $V(i)$ across the capacitor C represents a neuron's membrane potential. The resistor R in parallel with the capacitor models the membrane resistance. While the current source $I_i(t)$ charges up the capacitor and increases the membrane potential, some current leaks through R. The following equation describes the leaky integration of the membrane potential.

$$C\frac{dV_i(t)}{dt} = I_i(t) - \frac{V_i(t)}{R}. \tag{4.2}$$

When $V_i(t)$ exceeds a threshold voltage θ_i, set by the diode, the neuron emits a spike as its output $Y_i(t)$, or a spike train $s_i(t)$ over time. After firing, the capacitor is discharged through a small R_{out}, i.e., $R_{\text{out}} \ll R$, to reset $V_i(t)$. Note that the spiking neural network described above uses binary spikes to communicate between neurons, different from a non-spiking neural network [75, 74] or a spiking neural network that relies on analog voltage or current as the way to communicate between neurons [12, 13].

For simulation and implementation of the neuron, it is customary to discretize the

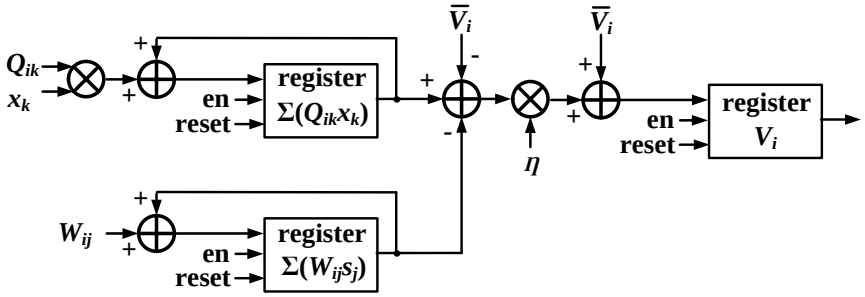


Figure 4.4: Digital SAILnet neuron model.

continuous-time voltage and current equations to [76, 31]

$$V_i^{(n+1)} = \overline{V}_i^{(n)} + \eta \left(\sum_{k=1}^{N_p} Q_{ik} x_k - \sum_{j \neq i} W_{ij} s_j^{(n)} - \overline{V}_i^{(n)} \right), \tag{4.3}$$

where

$$\overline{V}_i^{(n)} = V_i^{(n)} I_{(-\infty,\theta)} \left(V_i^{(n)} \right), \tag{4.4}$$

and

$$s_j^{(n)} = I_{[\theta,\infty)} \left(V_j^{(n)} \right), \tag{4.5}$$

where (n) and $(n+1)$ are the current and the next neuron update iteration. $\eta = \Delta t/\tau$ is the neuron update step size, and $\tau = RC$ is the neuron time constant. η is Δt normalized by the time constant τ. We define $I_{(a,b)}(x)$ as an indicator function where it is 1 if $x \in (a, b)$, and 0 otherwise. Fig. 4.4 describes a digital SAILnet neuron model.

Given a neuron update step size of η, the number of update steps n_s in response to each image patch is $w/(\eta\tau)$. With these, the spike count of neuron i for the steps

of n_s is

$$c_i = \sum_{n=1}^{n_s} s_j^{(n)}. \tag{4.6}$$

Spike count c_i is the output of the inference phase, and is used for learning.

4.1.1.2 Local learning rule

In the learning phase, the firing threshold θ_i and the weights W_{ij} and Q_{ik} are constantly updated according to a learning rule. Computations involved in the SAILnet learning rule [76] are local, and the results are also shown to successfully reproduce key features of biological receptive fields in mammalian visual cortex. The SAILnet learning rule enforces sparse and independent neuron spiking, and it involves only local computations, both of which are implementation-friendly features. In the following, we briefly introduce the SAILnet learning rule.

In response to an input image X, neurons in the network generate spikes, and the spikes can be used to reconstruct the input image based on the learned feature dictionary or learned receptive fields. The dictionary Q is simply the set of feed-forward connection weights: $Q = [Q_1, Q_2, \ldots, Q_N]$ where N is the total number of neurons in the network and $Q_i = [Q_{i1}, Q_{i2}, \ldots, Q_{iN_p}]^{\mathrm{T}}$ are feed-forward connection weights associated with the connections between neuron i and each pixel of the input image patch. For this case, an image patch consists of N_p pixels. The reconstructed image $\hat{X}$ is formulated as the linear combination (or the weighted sum) of the dictionary elements [76], known as the linear generative model.

$$\hat{X} = \sum_{i=1}^{N} Q_i c_i, \tag{4.7}$$

where c_i denotes the number of spikes generated by neuron i in response to an input image, collected over an inference window w. Using the SAILnet learning rule [76], the spikes are kept sparse, and the majority of the terms in the summation of (4.7)

are zero.

The SAILnet algorithm for learning the dictionary Q minimizes the mean squared error (MSE) between M input image patches $X = [X_1, X_2, \ldots, X_M]$ and their corresponding reconstructed patches $\hat{X} = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_M]$, i.e., $\Sigma_{m=1}^{M} \|X_m - \hat{X}_m\|^2$, while satisfying the constraints of sparse and decorrelated neural activities across the network [76]. The two constraints are justified by the experiments in [81, 82]. The constrained optimization problem can be summarized as

$$\begin{aligned} \hat{Q} = \arg\min_{Q} \sum_{m=1}^{M} \|X_m - QC_m(Q)\|^2 \\ \text{s.t. } \frac{1}{M}\sum_{m=1}^{M} c_{mi}(Q_i) = p, \qquad \forall i = 1, 2, \ldots, N \\ \frac{1}{M}\sum_{m=1}^{M} c_{mi}(Q_i)c_{mj}(Q_j) = p^2, \;\; \forall i \neq j \end{aligned} \tag{4.8}$$

where $C_m(Q) = [c_{m1}(Q_1), c_{m2}(Q_2), \ldots, c_{mN}(Q_N)]^{\mathrm{T}}$ and c_{mi} is the number of spikes of neuron i in response to the image patch m. C_m is a function of feature dictionary Q, because the dynamics of neurons depend on Q, as discussed in the previous section. We denote p as the target firing rate, or the average number of spikes per image, which is set to a low value. For example, $p = 0.01$ means a neuron spikes once every 100 image patches.

For simplicity, to find the update equation of Q, we evaluate iterative steps of inference and learning (or updating Q): 1) inference to compute C for a given Q by following (4.3), (4.4), (4.5) and (4.6); 2) learning to update Q using the precomputed C. To find the Q update, we introduce a Lagrange function $L(Q, \lambda, \tau)$ where $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_N]$ and $\tau = [\tau_{11}, \tau_{12}, \ldots, \tau_{NN}]$ are Lagrange multipliers. The Lagrange

function can be given as

$$L(Q,\lambda,\tau)=\sum_{m=1}^{M}\|X_m-QC_m\|^2+\sum_{i=1}^{N}\lambda_i\left(\frac{1}{M}\sum_{m=1}^{M}c_{mi}-p\right)+\sum_{j\neq i}\tau_{ij}\left(\frac{1}{M}\sum_{m=1}^{M}c_{mi}c_{mj}-p^2\right). \tag{4.9}$$

In practical implementation, the network is trained over millions of image patches (M$\sim 10^6$), so it is infeasible to directly calculate derivatives of the Lagrange function. The solution to the constrained optimization problem (4.8) can be found by applying the stochastic gradient descent method to (4.9). In each iteration, we randomly select R patches ($\sim 10^2$) out of M patches ($\sim 10^6$) to compute the gradient.

Reference [76] states that $\Delta\theta_i \propto -\Delta\lambda_i$ (i.e., the neuron threshold needs to be increased to enforce sparsity if a measured firing rate is high) and $\Delta W_{ij} \propto -\Delta\tau_{ij}$ (i.e., the feedback weight between two neurons needs to be increased in order to enforce uncorrelation if the spiking of the two neurons is correlated). The update for Q can be simplified under the condition that the target firing rate is small [76]. The SAILnet learning rule that governs the iterative updates of the firing thresholds (θ), feedback connection weights (W), and feed-forward connection weights (Q) is given as

$$\theta_i^{(t+1)}=\theta_i^{(t)}+\alpha\left(\bar{c}_i^{(t)}-p\right), \tag{4.10}$$

$$W_{ij}^{(t+1)}=W_{ij}^{(t)}+\beta\left(\overline{c_ic_j}^{(t)}-p^2\right), \tag{4.11}$$

$$Q_i^{(t+1)}=Q_i^{(t)}+\gamma\left(\overline{c_iX}^{(t)}-\overline{c_i^2}^{(t)}Q_i^{(t)}\right), \tag{4.12}$$

where

$$\bar{c}_i^{(t)}\triangleq\frac{1}{R}\sum_{m\in S_t}c_{mi}\left(\theta_i^{(t)},W_i^{(t)},Q_i^{(t)}\right), \tag{4.13}$$

$$\overline{c_ic_j}^{(t)}\triangleq\frac{1}{R}\sum_{m\in S_t}c_{mi}\left(\theta_i^{(t)},W_i^{(t)},Q_i^{(t)}\right)c_{mj}\left(\theta_j^{(t)},W_j^{(t)},Q_j^{(t)}\right), \tag{4.14}$$

$$\overline{c_i^2}^{(t)}\triangleq\frac{1}{R}\sum_{m\in S_t}c_{mi}^2\left(\theta_i^{(t)},W_i^{(t)},Q_i^{(t)}\right), \tag{4.15}$$

$$\overline{c_i X}^{(t)} \triangleq \frac{1}{R} \sum_{m \in S_t} c_{mi} \left(\theta_i^{(t)}, W_i^{(t)}, Q_i^{(t)} \right) X_m, \tag{4.16}$$

where α, β, γ are tuning parameters, and (t) and $(t+1)$ indicate the current and the next weight update iteration. Note that (t) differs from the previously defined (n) in that (n) is the current neuron update iteration. S_t denotes a sequence of R integers randomly selected with replacement from $\{1, 2, \ldots, M\}$.

As a result of learning, we obtain $\theta^{(\infty)}$, $W^{(\infty)}$, and $Q^{(\infty)}$, and these thresholds, weights, and receptive fields (dictionary) are used in inference. The pseudo-code for the SAILnet learning rule is listed in Table. 4.1. Note that the above learning rule is local: neuron i updates the firing threshold θ_i and the feed-forward connection weight Q_{ik} based on its own spike count, and it updates the feedback connection weight W_{ij} based solely on the activity of the pair of neurons that are associated with the connection. For a detailed derivation of these learning rules, see [76]. In the learning phase, input stimuli and neuron spikes trigger updates of firing thresholds, feedback connection weights and feed-forward connection weights following (4.10), (4.11) and (4.12). In the inference phase, the thresholds and weights are all fixed, and we obtain spikes using (4.6) with the threshold.

4.1.2 Locally competitive algorithm (LCA)

The locally competitive algorithm (LCA) [75] implements a dynamic system by introducing internal states in the neuron model, and finds sparse coefficients of the feature dictionary that are used to best approximate the input signals like other sparse coding approaches [11, 74]. LCA uses leaky integrate-and-fire neurons for learning and inference, but an LCA network is different from spiking neural networks [76, 53]. Neurons in the network are non-spiking, i.e., analog neuron output, but LCA provides excellent performance in sparse approximation of dynamic inputs such as videos, outperforming conventional greedy matching pursuit algorithms [75].

Table 4.1: Pseudo-code for the SAILnet learning

- Set learning parameters α, β, and γ.
- Set neuron update step size η and neuron update steps n_s.
- Initialize threshold $\theta^{(0)}$ and weights $W^{(0)}$ to zeros.
- Initialize dictionary $Q^{(0)}$ to random values normally distributed with $N(0, 1)$.
- For each batch of images $t = 1, 2, \ldots, T$:
  1. Generate a sequence S_t of R integers randomly selected with replacement from $\{1, 2, \ldots, M\}$
  2. For each image patch $m \in S_t$:
     (a) Initialize neuron potential $V^{(0)}$ to zero.
     (b) For each time step $n = 1, 2, \ldots, n_s$
        i. Evaluate neuron potential using (4.3) and (4.4).
     (c) Compute spike count C_m using (4.6) and neuron potential evaluated in 2-(b)-i.
  3. Compute momentum using (4.13), (4.14), (4.15), (4.16) and spike count computed in 2-(c).
  4. Update threshold, feedback weights, and receptive fields using (4.10), (4.11), (4.12) and the compute momentum.

4.1.2.1 Non-spiking LCA

Non-spiking LCA or the conventional LCA performs sparse approximation by solving an optimization problem given as [75]

$$\hat{C}_m = \arg\min_C \left(\frac{1}{2} \|X_m - QC\|_2^2 + \beta R(C) \right), \tag{4.17}$$

where X_m is an $N_p \times 1$ input signal, and Q is a feature dictionary composed of N feature vectors where N is greater than N_p (i.e., the feature dictionary is overcomplete). We denote C_m as an $M \times 1$ coefficient vector of the feature dictionary, so $Q\hat{C}$ is the reconstructed signal. The sparsity inducing regularizer R is used to find a sparse coefficient vector that minimizes the reconstruction errors.

To find a solution of (4.17), LCA introduces internal state variables V interpreted as the neuron potential. For the details, please see [75]. The dynamics of leaky integrate and fire neurons to update V is formulated as [75]

$$V_m^{(n+1)} = V_m^{(n)} + \eta \left(Q_m^T \left(X_m - Q_m C_m^{(n)} \right) + C_m^{(n)} - V_m^{(n)} \right), \tag{4.18}$$

where

$$C_m^{(n)} = T_\lambda \left(V_m^{(n)} \right), \tag{4.19}$$

where $T_\lambda(\cdot)$ denotes a threshold function. For example, if it is defined as a hard-thresholding function, $C_m^{(n)}$ is 0 if $|V_m^{(n)}| \leq \lambda$ and it is $V_m^{(n)}$ otherwise, so a neuron spikes if its potential is greater than the neuron threshold λ. We denote η as the neuron update step size. The neuron potential converges to a stable state after a sufficient number of iterations [75, 83]. Note that (4.18) is a discretized version of the continuous neuron dynamics of the Rozell's network [75].

Similar to the SAILnet algorithm, we find an optimal dictionary Q in the learning phase by solving the following optimization problem [76]:

$$\hat{Q} = \arg\min_{Q} \sum_{m=1}^{M} \|X_m - Q C_m(Q)\|^2, \tag{4.20}$$

where M is the number image patches used for training. Similar to (4.9), to find the update Q, we introduce a Lagrange function for fixed C_m [76]. To find a solution to (4.20), we apply the stochastic gradient descent method to the Lagrange function. The Q update can be given as

$$Q^{(t+1)} = Q^{(t)} + \alpha \frac{1}{R} \sum_{m \in S_t} \left(\left(X_m - Q^{(t)} C_m \left(Q^{(t)} \right) \right) C_m \left(Q^{(t)} \right)^T \right), \tag{4.21}$$

where α is a tuning parameter, and S_t denotes a sequence of R integers randomly selected with replacement from $\{1, 2, \ldots, M\}$. Neuron output C_m can be evaluated using (4.18) and (4.19).

As a result of learning using (4.21), we obtain a feature dictionary $Q^{(\infty)}$ that is used in the inference phase. The pseudo-code for the learning used for the LCA algorithm is summarized in Table. 4.2. Note that unlike the SAILnet algorithm, the Q update is a global computation, i.e., to update an entry of Q, we need to know all neuron outputs and the other entries of Q. The global update rule is more computationally intensive than the local update rule such as the SAILnet learning rule [76].

Table 4.2: Pseudo-code for the LCA learning

- Set learning parameters α.
- Set neuron update step size η, neuron update steps n_s and the neuron threshold λ.
- Initialize dictionary $Q^{(0)}$ to random values normally distributed with $N(0, 1)$.
- For each batch of images $t = 1, 2, \ldots, T$:
  1. Generate a sequence S_t of R integers randomly selected with replacement from $\{1, 2, \ldots, M\}$
  2. For each image patch $m = 1, 2, \ldots, R$:
     (a) Initialize neuron potential $V^{(0)}$ to zero.
     (b) For each time step $n = 1, 2, \ldots, n_s$:
        i. Evaluate neuron potential using (4.18) and (4.19).
     (c) Evaluate neuron output $C_m^{(n_s)}$ using (4.19).
  3. Update receptive fields $Q^{(t)}$using (4.21) and the neuron outputs evaluated in 2-(c).

4.1.2.2 Spiking LCA

LCA provides benefits in hardware implementation because feedback weights can be precomputed using the feature dictionary, and all neurons in the network share the same neuron threshold. However, neuron output is an analog value, so implementation

cost is high. In comparison, spiking neural networks allow for efficient communication using address-event representation [39, 40, 31], and has a potential to reduce the computations using binary spikes. Therefore, we investigate a rate-based LCA (or spiking LCA [53]).

For simplicity, we consider a soft-thresholding function with non-negativity constraint on spiking rate, i.e., $c_i^{(n)} = \max(0, u_i^{(n)} - \lambda)$ where $u_i^{(n)}$ is an internal state variable of neuron i and λ is the offset of neuron update step [53]. In comparison, note that c_i was defined as analog neuron output and spike count in non-spiking LCA and SAILnet, respectively, and that λ was defined as the neuron threshold in non-spiking LCA. The potential update of rate-based LCA can be formulated as [53]

$$V^{(n+1)} = \overline{V}_i^{(n)} + \eta \left(u^{(n)} - \lambda \right), \tag{4.22}$$

where

$$\overline{V}_i^{(n)} = V_i^{(n)} I_{(-\infty,1)} \left(V_i^{(n)} \right), \tag{4.23}$$

where $I_{(-\infty,1)}(x)$ denotes an indicator function where it is 1 if $x \in (-\infty, 1)$, and 0 otherwise, i.e., the potential is reset to zero if it exceeds 1. Therefore, the steady state inter-spike interval is approximately $1/(u^{(\infty)} - \lambda)$, so the instantaneous firing rate of neuron i at time n becomes $c_i^{(n)} = \max(0, u^{(n)} - \lambda)$.

Given a neuron update step size of η and the number of update steps n_s in response to each image patch, the spiking rate of neuron i can be measured as

$$c_i^{(n_s)} = \frac{1}{n_s \eta} \sum_{n=1}^{n_s} I_{[1,\infty)} \left(V_i^{(n)} \right), \tag{4.24}$$

where $I_{[1,\infty)}(x)$ is an indicator function where the value is 1 if $x \in [1, \infty)$, and 0

otherwise. Spiking rate $c_i^{(\infty)}$ is the output of the inference phase, and is used for learning.

In spiking LCA, the internal variable can be formulated as [53]

$$u_i^{(n)} = Q_i^T X - \sum_{j \neq i} \left(W_{i,j} \sum_{k}^{t_{j,k}<n} \alpha^{(n-t_{j,k})} \right), \tag{4.25}$$

and

$$\alpha^{(n)} = U^{(n)} e^{-n}, \quad W \triangleq Q^T Q - I \tag{4.26}$$

where η is the neuron update step size, and W denotes feedback weights. $t_{j,k}$ is the k^{th} spike time of neuron j, and $U^{(n)}$ is a unit step function. There is a high degree of similarity between the LCA algorithm and the SAILnet algorithm in that the input excitations increase neuron potential, and the spikes of neighboring neurons reduce the neuron potential.

Note that the dynamics of neurons in the spiking LCA algorithm (4.22) and (4.25) are similar to the dynamics of neurons in the SAILnet algorithm. However, compared to SAILnet, there are also three main differences in the neuron dynamics: 1) the feedback weights of spiking LCA are determined by feature dictionary Q; 2) the leaky term in (4.22) is a constant λ. On the other hand, the leaky term of the SAILnet potential update varies and depends on the potential itself, as formulated in (4.3); 3) Finally, the neuron threshold of spiking LCA is same for all neurons, and it is 1 so that the firing rate is approximately equal to the output of soft-thresholding function implemented in non-spiking LCA [75, 53].

In the learning phase, spiking LCA follows (4.21). Pseudo-code for the spiking LCA learning is summarized in Table 4.3. We estimate the spiking rate C by counting spikes in a long inference window, and it is shown that the estimated spiking rate is

Table 4.3: Pseudo-code for the spiking LCA learning

- Set learning parameters α.
- Set neuron update step size η, neuron update steps n_s and the neuron threshold λ.
- Initialize dictionary $Q^{(0)}$ to random values normally distributed with $N(0,1)$.
- For each batch of images $t = 1, 2, \ldots, T$:
  1. Generate a sequence S_t of R integers randomly selected with replacement from $\{1, 2, \ldots, M\}$
  2. For each image patch $m = 1, 2, \ldots, R$:
     (a) Initialize neuron potential $V^{(0)}$ to zero.
     (b) For each time step $n = 1, 2, \ldots, n_s$:
        i. Evaluate neuron potential $V^{(n)}\left(Q_i^{(t-1)}\right)$ using (4.22), (4.23), (4.25), and (4.26).
     (c) Evaluate spiking rate $C_m^{(n_s)}$ using (4.24) and neuron potential.
  3. Update receptive fields $Q^{(t)}$using (4.21) and spiking rate evaluated in 2-(c).

statistically equivalent to the solution obtained from non-spiking LCA. For the details of its derivation, please see [53].

4.2 Hardware implementation challenges

A binary spiking and discrete-time neuron model described above makes it possible to design a simple digital neuron, as illustrated in Fig. 4.5(a). The neuron is connected to N_p input pixels and $N-1$ neighboring neurons through point-to-point links. The neuron contains two memories: Q memory to store feed-forward connection weights (N_p entries in total, one per each pixel) and W memory to store feedback connection weights ($N-1$ entries in total, one per each neighboring neuron). The neuron performs leaky integrate-and-fire described in (4.3) in response to pixel inputs and neuron spikes.

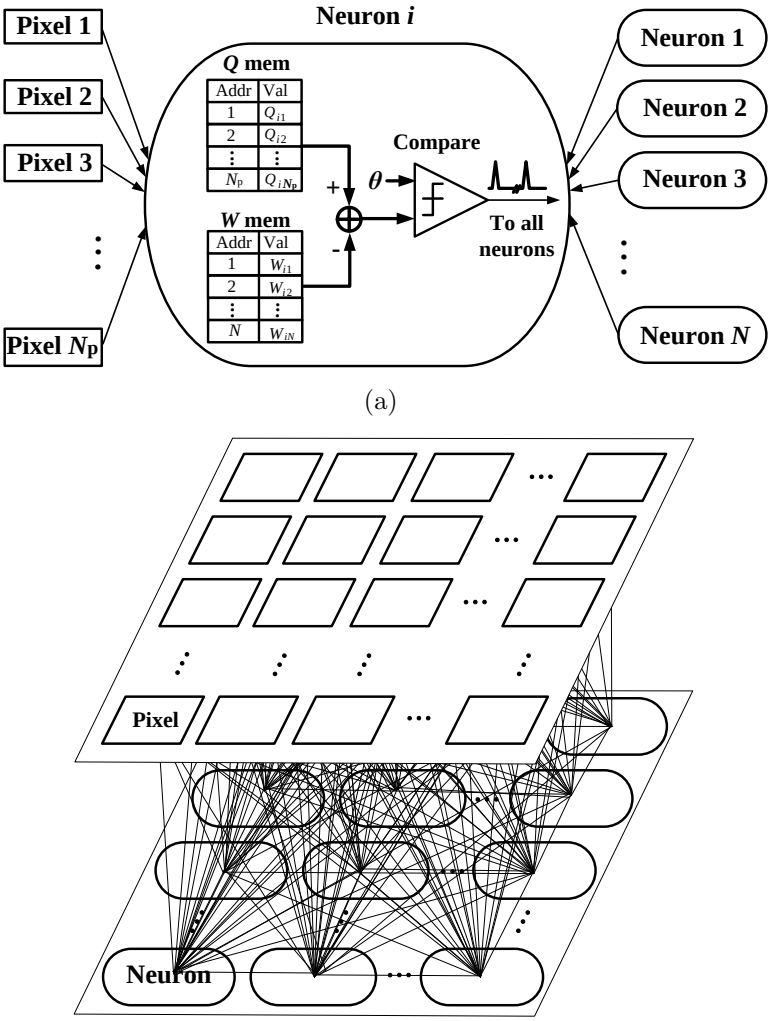
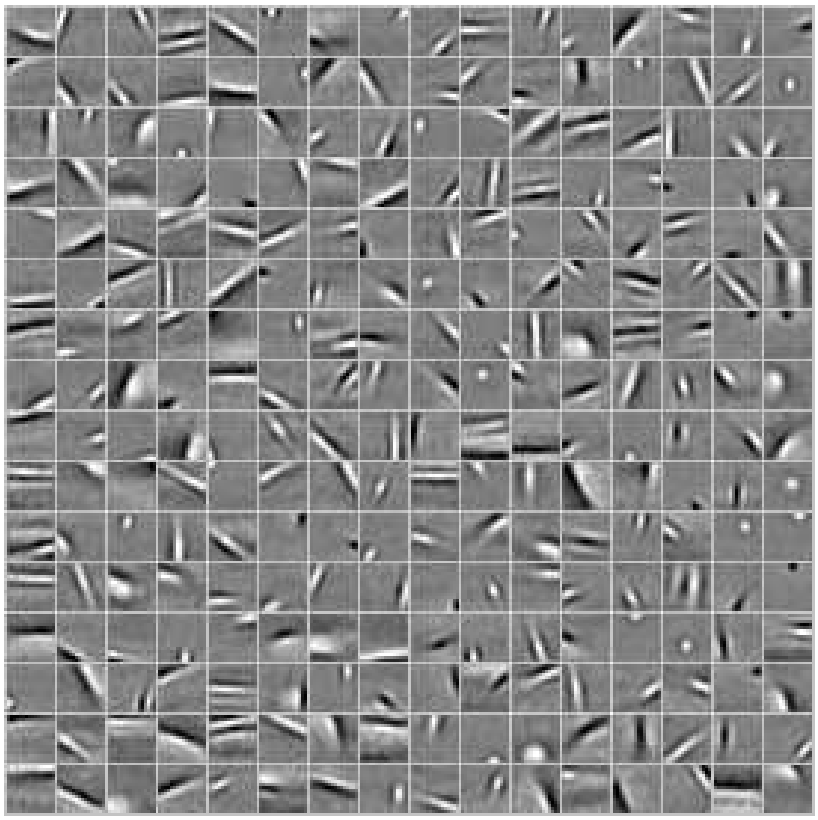


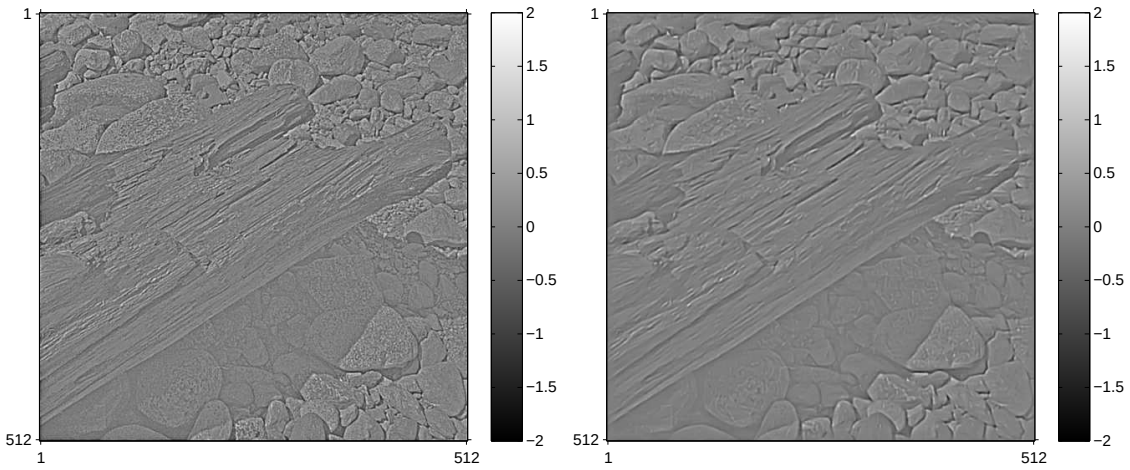
Figure 4.5: (a) A digital neuron design, and (b) a fully connected network for sparse coding.

The size of the neural network depends on the size of the input image patch. For example, to detect features in a 16×16 image patch of 256 pixels, an overcomplete network (i.e., the number of neurons in the network is greater than 256) is helpful to provide a good performance of image encoding [76]. In particular, we consider a 512-neuron network. Each neuron in this network requires a 256-entry Q memory and a 511-entry W memory, which easily dominate the size of the digital neuron. To implement the fully connected network [35], each neuron needs to be connected to 256 pixel inputs and 511 neighboring neurons. The fully connected network matches the biological neural network and it provides the highest throughput of f_{clk}/n_s 16×16 image patches per second, or $256f_{clk}/n_s$ pixels/s (px/s), where f_{clk} is the clock frequency, n_s is the number of neuron update steps for each inference. The fully connected network as illustrated in Fig. 4.5(b) is however impractical due to the high interconnection overhead that grows at N^2 for a network of N neurons.

Regardless of the practicality, the fully connected network is often the underlying assumption of neural network algorithms. The performance of the algorithms using the fully connected network can be simulated in software to provide the baseline reference. Fig. 4.6 shows the performance of a 512-neuron network in feature detection and image reconstruction after it has been trained using the SAILnet learning rule. In the beginning of each training, the W weights are set to zero, and the Q weights are initialized with Gaussian white noise. Fig. 4.6(a) shows 144 sample neurons' receptive fields (Q weights) that are obtained after learning. The learned RFs resemble the receptive fields recorded from the macaque monkey brain [76, 84]. Fig. 4.6(b) is the input 512×512 whitened natural image from [85]. Whitening of natural images is an operation that flattens the amplitude spectrum. Whitening is done by computing the 2D FFT of the original images, passing through a 2D ramp filter (since the amplitude spectrum of natural images tends to be 1/f, where f is the frequency), followed by 2D IFFT [9, 86]. The whitened image is divided to 16×16 patches as the inputs to the



(a)



(b) (c)

Figure 4.6: (a) 256 randomly selected receptive fields (each square in the grid represents a 16×16 receptive field), (b) whitened input image, and (c) reconstructed image using sparse code.

neural network for inference. Fig. 4.6(c) is the reconstruction of the image using the linear generative model (4.7) with the neuron spikes obtained from the fully connected network and the neurons' receptive fields (Q weights). The fidelity of sparse coding is measured by the error in the reconstructed image by the linear generative model. In the following we will use root-mean-square error (RMSE) as the image fidelity metric.

4.3 Simulation of spiking neurons

In this section, we simulate two sparse coding algorithms: SAILnet and spiking LCA. A study of the neuron spiking dynamics in SAILnet uncovers important design considerations involving the neural network size, target firing rate, and neuron update step size. Optimal tuning of these parameters keeps the neuron spikes sparse and random to achieve the best image fidelity. Similarly, a simulation based analysis of spiking LCA allows for the optimal tuning of neuron update steps, update step size, and threshold. In simulation, we assume neurons are fully connected.

4.3.1 SAILnet

In this section, we adapt neural network architectures for efficient and high-performance implementations of a sparse coding algorithm called the sparse and independent local network (SAILnet). Each neuron in the network has a receptive field, and receptive fields of neurons form a dictionary used to represent the input image. SAILnet enforces neuron activities in the network to be sparse and uncorrelated so as to perform sparse coding. A study of the neuron spiking dynamics in the network uncovers important design considerations involving the neural network size, target firing rate, and neuron update step size. Optimal tuning of these parameters keeps the neuron spikes sparse and random to achieve the best image fidelity. In simulation, we assume neurons are fully connected.

The digital neural network is a dynamic system, and its learning and inference are

dependent on the neuron update step size η (in (4.3)) and the target neuron firing rate p (in (4.10)and (4.11))). η controls the step size of the neuron potential update. The smaller the η (more time steps), the closer the discrete-time system mimics a continuous-time system for a higher accuracy, but the longer it takes for learning to converge, so intuitively η determines the tradeoff between accuracy and throughput. p controls the target firing rate, and is set to a low value to maintain sparse firing. A low p is also appealing as the sparse firing results in sparse communication and low power consumption. However, the sparseness is relative to the neural network size. A small network is not likely to support a low firing rate.

We analyze the influence of η and p and relate them to the neuron spike rate pattern that underpin the performance of the SAILnet algorithm. In the following experiments, we first train the network based on given η and p values, and then perform inference using the trained network. A fully connected network is assumed. The focus here is on the dynamics of the network when it is performing inference, consistent with the motivation outlined in the background section.

4.3.1.1 Spike rate pattern

The neuron spike rate pattern is depicted in Fig. 4.7 as the average spike rate p_s at each time step across a network of 512 neurons when performing inference with a target firing rate of $p = 0.04$. η is varied between 2^{-3} to 2^{-6}, and the inference window w is set to 3τ. η determines the number of time steps in the window w, i.e., $n_s = w/(\eta\tau)$. For example, if $\eta = 2^{-5}$, $n_s = 96$. For a fully connected digital neural network operating at a clock frequency of f_{clk}, the sparse coding throughput is f_{clk}/n_s image patches per second. The throughput of the network is inversely proportional to n_s, and thus proportional to η.

To measure the spike rate, the neuron potential is reset to 0 before performing each inference. When presented with an input image patch, some neurons will start

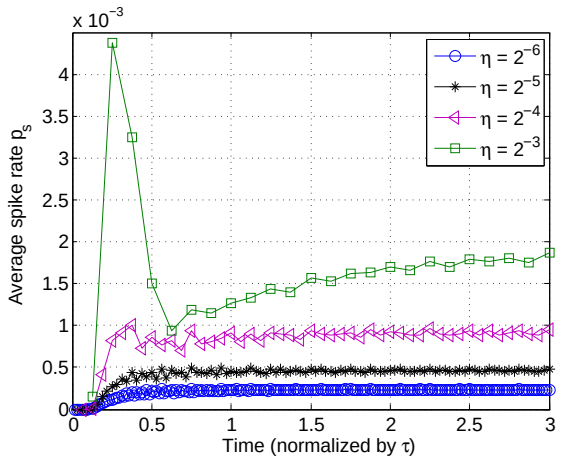
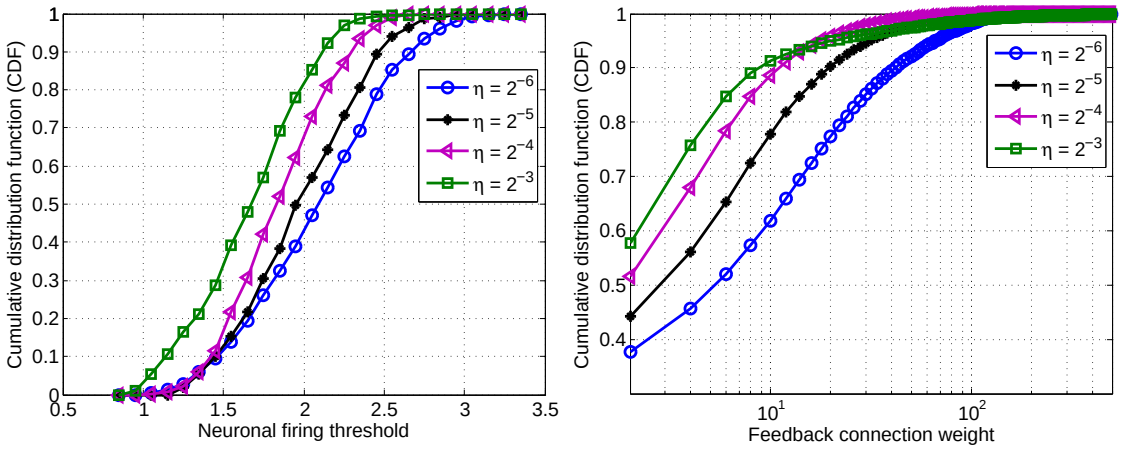


Figure 4.7: Average spike rate at each time step across a network of 512 neurons when performing inference with a target firing rate of $p = 0.04$.

to charge up and fire, leading to a rise of the spike rate until it starts to settle to a steady state. The target firing rate p determines the average spike rate summed over the inference window w, i.e., $p \approx \sum_{i=0}^{n_s} p_s[i]$, where $p_s[i]$ is the spike rate at time step i. A larger step size η means fewer time steps n_s, and a higher spike rate in each time step to meet the a given target firing rate p.

Varying η has a pronounced effect on the spike rate pattern, as seen in Fig. 4.7. A large η results in a large peaking of the spike rate, which is attributed to the quick rise of neuron potentials: many neurons fire together after a few initial steps and then inhibitions take effect to silence most of the neurons. The bursts of neuron spikes are not desirable for implementation, because they lead to competitions for hardware resources and communication bandwidth. A small η results in a sparse, random, and more evenly distributed spike rate over time, and a more efficient utilization of hardware resources.

The influence of η can be understood by the distribution of the neuron firing threshold and the distribution of the feedback connection weights (inhibitory weights). The neuron firing threshold controls the sparseness of neuron spikes [76]: the higher the threshold, the more difficult it is to reach the threshold, thus fewer spikes are



(a) (b)

Figure 4.8: (a) CDF of neuron firing thresholds, and (b) CDF of feedback connection weights for different η values.

expected. The feedback connection weight controls the correlation between neuron spikes [76]. A higher positive feedback connection weight indicates a strong inhibition – when one neuron spikes, the other neurons will be strongly inhibited to de-correlate spike activities. Fig. 4.8(a) shows the cumulative distribution function (CDF) of the neuron firing threshold, and Fig. 4.8(b) shows the CDF of the feedback connection weights for different η values. A smaller η results in a higher neuron firing threshold and feedback connection weights on average, which confirm the observation that a smaller η produces more sparse and random spikes that are amenable for an efficient implementation.

The smaller update step size improves the fidelity of sparse coding, as evidenced in the lower RMSE in image reconstruction by the linear generative model, as shown in Fig. 4.9. Therefore it is advantageous to choose the smallest η that meets the throughput requirement.

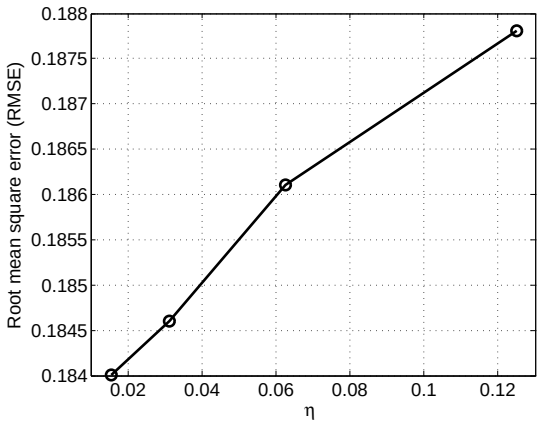


Figure 4.9: RMSE of image reconstruction by the linear generative model when varying step size η.

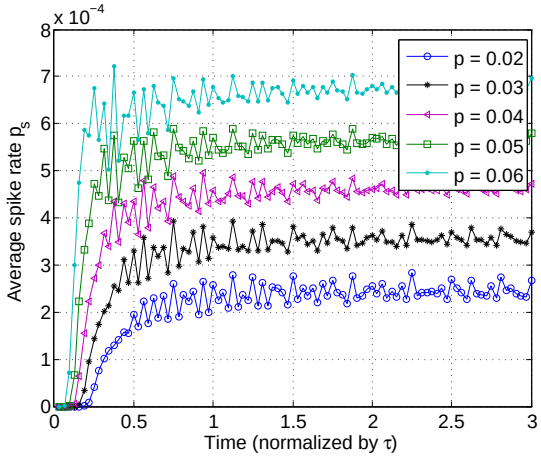
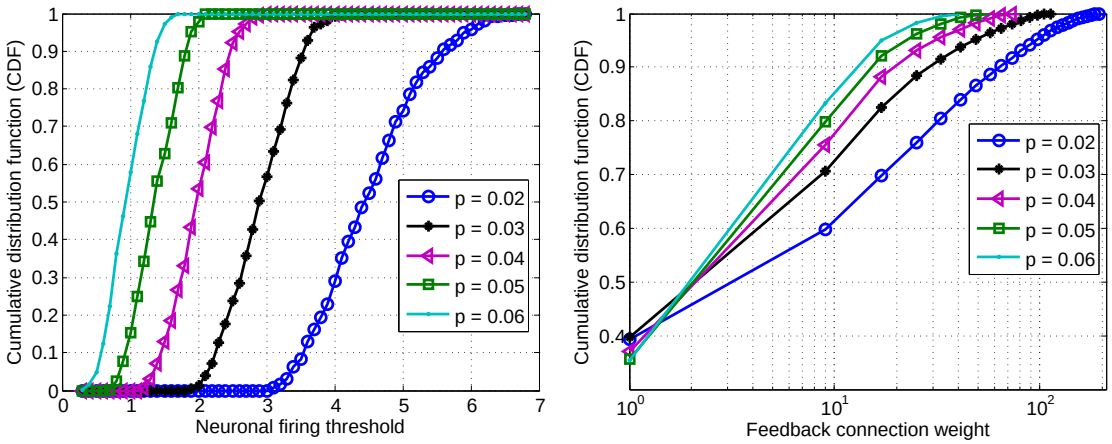


Figure 4.10: Average spike rate at each time step across a network of 512 neurons when performing inference with $\eta = 2^{-5}$.

4.3.1.2 Target firing rate

The target firing rate p determines the average spike rate. A low p results in a low average spike rate, as seen in Fig. 4.10 for a network of 512 neurons with $\eta = 2^{-5}$ and $w = 3\tau$. Reducing p raises the firing thresholds and the feedback connection weights, as shown in Fig. 4.11(a) and (b), and creates a more sparse and random spiking network.

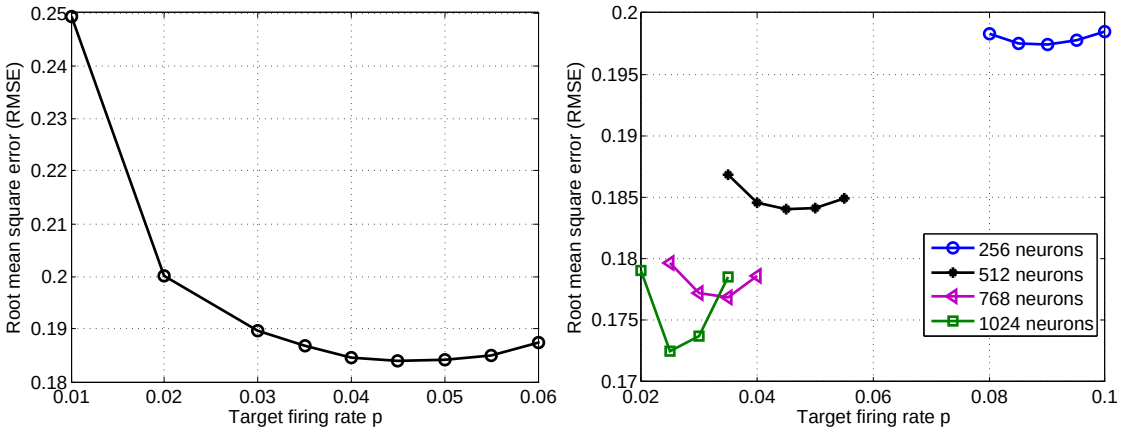


(a) (b)

Figure 4.11: (a) CDF of neuron firing thresholds, and (b) CDF of feedback connection weights for different p values.

A low target firing rate p is attractive for an efficient implementation, but a very low p raises the RMSE, as seen in Fig. 4.12(a). In a network of 512 neurons, $p = 0.01$ results in an average of only $512 \times 0.01 = 5.12$ spikes over the inference window. The linear generative model (4.7) contains less than a handful of terms, which are insufficient for sparse coding. Raising p to 0.02 doubles the number of spikes and reduces the RMSE. Continued increasing of p improves the RMSE until p reaches about 0.045, or about 23 spikes over the inference window. The RMSE then worsens as the spikes start to crowd. Therefore, while we optimize the SAILnet algorithm for sparse and random spikes, a minimum number of spikes is needed for a good RMSE.

To verify the minimum number of spikes necessary, we designed networks of 256,



(a) (b)

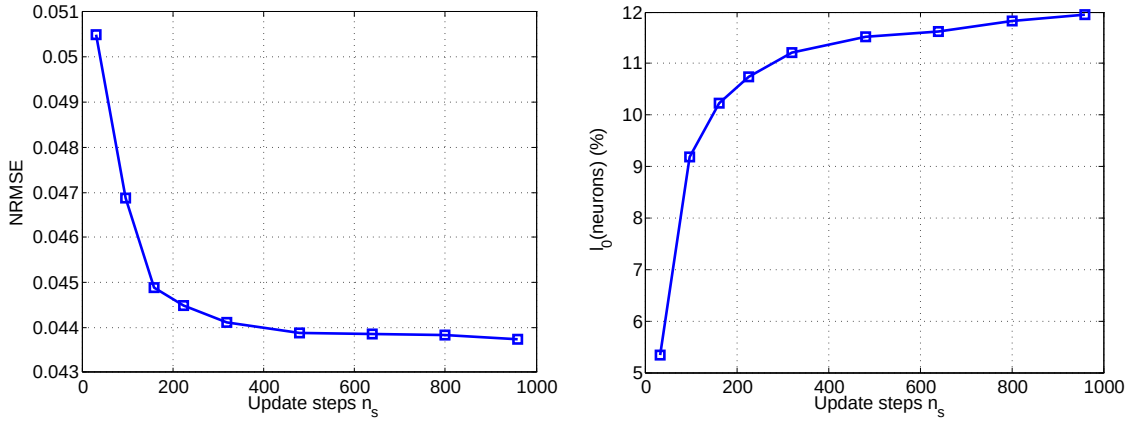
Figure 4.12: (a) RMSE of image reconstruction by the linear generative model when varing target firing rate p. (b) RMSE of image reconstruction by the linear generative model when varying target firing rate p and network size.

512, 768, and 1024 neurons, and analyzed the choice of p, as illustrated in Fig. 4.12(b). A lower RMSE is attainable in a larger network, but only with a good choice of p. The optimal p is lower in a larger network: about 0.09 in the 256 network, 0.045 in the 512 network, 0.03 in the 768 network, and 0.025 in the 1024 network, which all point to the almost constant optimal number of spikes to be around 20 to 25 regardless of the size of the network.

The above analysis yields two important insights: (1) a minimum number of spikes is needed for a good RMSE, and more spikes beyond the minimum are not needed, and (2) sparse and random spikes result in better RMSE. The two insights guide the selection of the neural network size along with the target firing rate for the best tradeoff between RMSE and implementation cost, and the selection of the update step size for the best tradeoff between RMSE and throughput.

4.3.2 Spiking LCA

In this section, we investigate the impact of tuning parameters on the performance of sparse coding. Particularly we focus on neuron update steps (n_s), neuron update



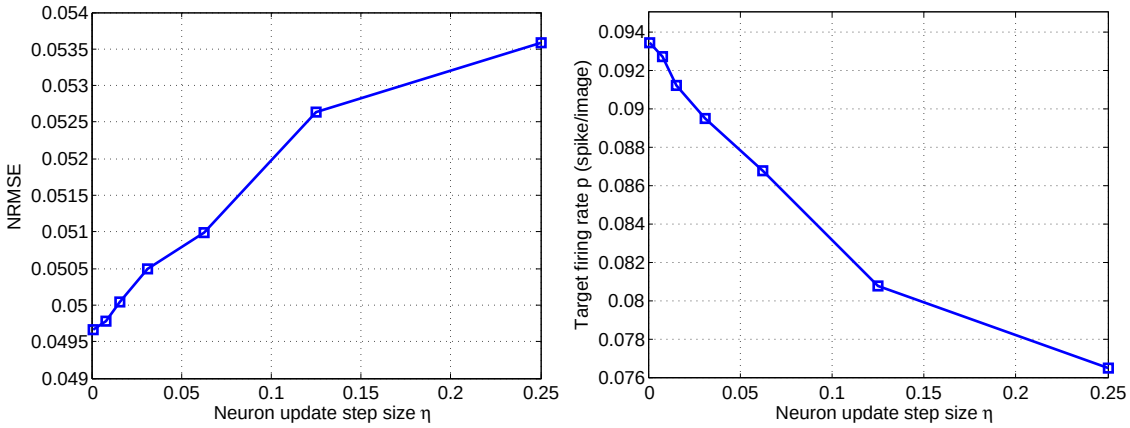
(a) (b)

Figure 4.13: (a) Normalized Root mean square error in the reconstructed image at each neuron update step in spiking LCA, and (b) the activities of neurons.

step size (η), and the neuron threshold (λ). To check the performance, we check the image reconstruction error. Input to the 256-neuron network is 512×512 whitened natural images [85], and each input image is divided into 16×16 overlapped patches. We measure the spiking rate of neurons after the presentation of each image patch. The spiking rate can be calculated by measuring spike count divided by the inference window, $n_s \times \eta$ seconds. An input image patch is reconstructed using the linear generative model, and each pixel of the 512×512 image is reconstructed by averaging corresponding pixels of reconstructed image patches.

4.3.2.1 Neuron update step

We study the trade off between the performance of sparse coding and the throughput of sparse coding. The number of neuron update steps implies the number of steps to update the neuron potential in response to an input image patch. Particularly, we measure the reconstruction errors in normalized root mean square error (NRMSE) between the input and the reconstructed image. Fig. 4.13(a) shows the NRMSE with different update steps. λ and η are set to 1.7 and 2^{-5} respectively for this case, and



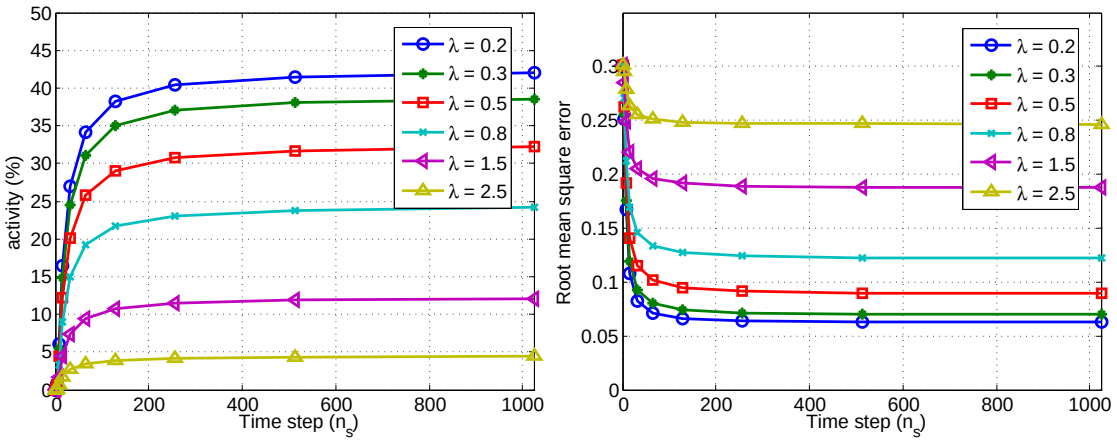
(a) (b)

Figure 4.14: (a) Normalized root mean square error in the reconstructed image at each neuron update step size, and (b) the spike count or target fire rate of neurons in the network.

n_s changes from 32 to 960. As the number of update steps increases, the reconstruction error monotonically decreases, and the improvement is negligible after 400 steps. More update steps help to estimate the spiking rate more accurately, since we are counting spikes over a longer inference window. Therefore, neurons in the network have more chances to spike, as shown in Fig. 4.13(b), resulting in a better image fidelity. However, a long inference window lowers the throughput of sparse coding, so there is a tradeoff between image fidelity and throughput.

4.3.2.2 Update step size

We explore the effects of the neuron update step size on the performance of sparse coding. We set the neuron threshold to 1.7 in order to have less than 10% neurons spike, and the update steps as $1/\eta$ to keep the inference window (ηn_s) to 1 second. Fig. 4.14(a) shows the reconstruction error with different update step size. A smaller update step size lowers the NRMSE because the digital neuron model better approximates continuous neuron model [53]. Therefore, we observe that more neuron spikes with a smaller update step size as shown in Fig. 4.14(b), improve the performance of



(a) (b)

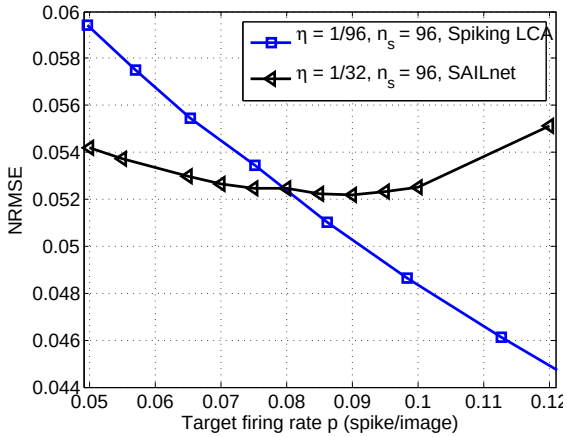
Figure 4.15: (a) Activity of neurons at each neuron update step. (b) Root mean square error in the reconstructed image with different neuron update offsets.

image reconstruction by representing input using many receptive fields (or features). Note that some (active) neurons have very low spiking rate. However, a downside of using a small update step size is a low throughput of inference because of a large number of neuron update steps.

4.3.2.3 Update step offset

The offset of neuron update step controls the amount to be accumulated to the neuron potential in every update step, and it affects spiking rate as formulated in (4.22). A high offset reduces the update amount, so it takes longer for the neuron to accumulate its potential to reach the threshold (or 1), which results in sparse neuron spikes, as shown in Fig. 4.15(a). For this simulation, the update step size is set to 2^{-4}. The sparse activities as a result of a high offset is beneficial for efficient hardware implementation [33]. However, a high offset increases image reconstruction error due to coarse sparse approximation, as shown in Fig. 4.15(b).

We compare spiking LCA with SAILnet by measuring the reconstruction error with different firing rate, as shown in Fig. 4.16. With high target firing rate due to



(a)

Figure 4.16: NRMSE of spiking LCA and SAILnet with different target firing rate.

a small update offset, spiking LCA outperforms SAILnet for this case. Note that the SAILnet local learning rule is enabled by the assumption that the activities of neurons are sparse [76]. A simulation study confirms that receptive fields hardly capture features in input images if a SAILnet network is trained using a high target firing rate. Note that SAILnet and spiking LCA are simulated with the same update steps in order to have the same throughput, and the update step size for spiking LCA is set to $1/n_s$ in order to transform the spiking rate measured in spiking count/second into the target firing rate measured in spike count [76, 31] for SAILnet.

In this section, we explored the effects of neuron update steps, neuron update step size, and neuron update offset on the algorithm performance. A large number of update steps and a smaller update step size improve the performance, but lowers the throughput. A small offset provides a good image fidelity, but it increases the activities of the neurons.

4.4 Summary

This chapter introduces sparse coding that mimics the human vision processing. In particular, SAILnet is considered for implementing sparse and uncorrelated spiking

neural network, and spiking LCA is studied to be used in the front-end of neuromorphic object recognition processor. We describe a fully-connected network and a digital spiking neuron architecture, and explore the neuron dynamics.

By exploring the neuron spiking dynamics of the SAILnet algorithm, we show that a low target firing rate and a small neuron update step size are necessary to maintain sparse and random neuron spikes for an efficient use of hardware resources. An optimal target firing rate depends on the network size, and a neuron update step size is set as a minimum amount that satisfies the throughput requirement.

The study of the spiking LCA shows that the reconstruction error converges to a low level after a sufficient number of update steps (or iterations). Similar to the SAILnet algorithm, a small update step size provides a better image fidelity, but it reduces the throughput. Unlike the SAILnet algorithm, the spiking LCA algorithm further reduces the reconstruction error as we increase target firing rate.

CHAPTER V

Architecture for sparse coding

Sparse and random spiking resulted from the optimal tuning of SAILnet plays a key role toward designing hardware architecture for SAILnet. This chapter investigates two practical architectures: a bus architecture that provides efficient neuron communications, but results in spike collisions; and a ring architecture that is more scalable than the bus, but causes neuron misfires. We show that the spike collision rate is reduced with a sparse spiking neural network, so an arbitration-free bus architecture can be designed to tolerate collisions without the need of arbitration. However, the bus speed can be limited by the network size due to the bus loading. To reduce neuron misfires in the ring architecture, we design a latent ring architecture to damp the neuron responses for an improved image fidelity. However, large ring size inevitably causes long latency for the spike propagation. To combine the benifits of both the bus and the ring architecture, this chapter proposes a hierarchical arbitration-free bus-ring architecture for the improved throughput and image fidelity. The performance of the bus, the ring and the hierarchical architecture are compared using synthesis.

This chapter is based in part on [31].

5.1 Neural network architectures

For a few decades, practical hardware architectures for spiking neural network have been investigated. Among them, we explore the bus and ring architectures and investigate the feasibility of adoption in SAILnet for a high throughput and scalable hardware implementation.

5.1.1 Bus Architecture

A bus allows many neurons to be connected [87, 36], replacing individual point-to-point links altogether. A bus can be used to connect pixel inputs to the neurons using a pixel bus, and connect neurons together using a neuron bus.

Once the inputs are complete, neurons will be ready to perform integrate and fire and generate spikes. The spikes are exchanged on the neuron bus. Since the neuron spikes are random, arbitration is required to resolve conflicts when multiple neurons try to access the bus at the same time. An arbiter decides the order that the access is granted depending on a pre- or a dynamically determined priority. The design of an arbiter is complicated by a large neural network, as the arbiter needs to serve many neurons and handle large fan-in and fan-out connections, and the service time is critical as excessive delays alter the dynamics of the neural network and degrade the performance. Solutions have been proposed to structure the arbiter design to reduce its fan-in and fan-out connections and improve its service time, but increasing the hardware cost [44, 41, 46, 45]. The design of the neuron bus is the focus of the following discussion.

AER is a popular time-multiplexing communication protocol for the neuron bus. Fig. 5.1 explains the protocol [39, 40]. When a neuron fires, the AER encoder puts the address of the neuron on the AER bus and asserts a request REQ. All neurons are attached to this bus. Upon hearing REQ, neurons read the address from the bus and perform integrate and fire. If neuron spikes are very sparse, AER enables an efficient

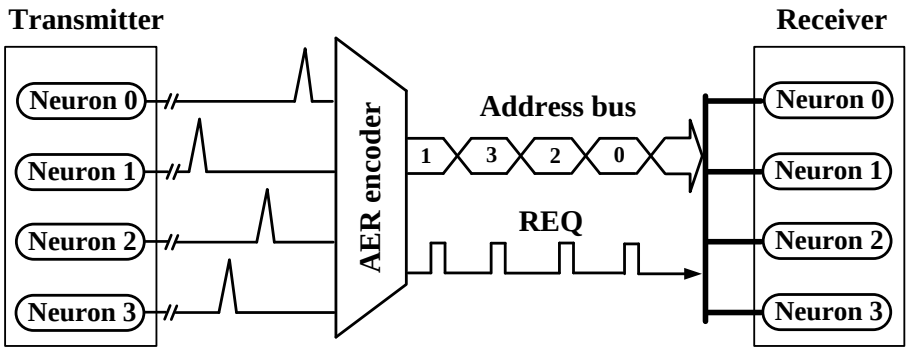


Figure 5.1: Neuron communication via AER protocol.

sharing of the bus. When there are more than one spike at the same time, it results in a collision, and arbitration is required to resolve the conflict. To be able to handle multiple requests in a timely manner, a synchronous AER bus needs to operate at a higher clock speed. If spikes happen in bursts, the bus speed has to be increased further. An asynchronous AER bus is potentially beneficial, but in this work we focus on a synchronous system.

5.1.2 Ring Architecture

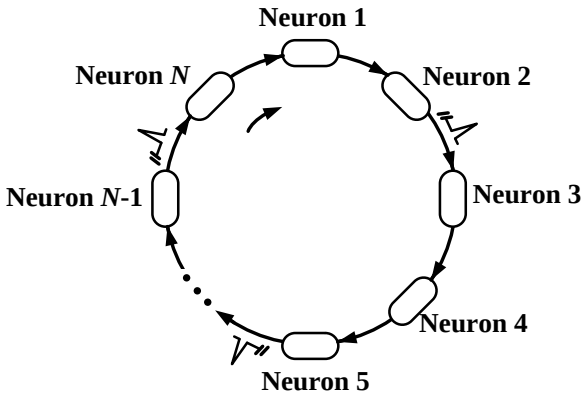


Figure 5.2: A ring architecture.

A ring [37] is a serial relay architecture as shown in Fig. 5.2. In the simplest setup, the communication in a ring is unidirectional, i.e., a neuron can only talk to one neighboring neuron. When a neuron spikes, it passes the spike as an address-

event to the next neuron, who then passes the address-event one step further, and so on.

In a digital implementation, a spike address-event steps through one neuron every clock cycle. For a spike to be heard by all neurons, it will have to travel along the entire ring, which takes $N-1$ clock cycles, where N is the size of the network. Once an address-event reaches its originating neuron, the event is deleted. One advantage of the ring over the bus is that it is more scalable because all the communications are local, and it eliminates all collisions. The throughput of the ring architecture is f_{clk}/n_s image patches per second. The short connections between neighboring neurons allow the ring to run at a higher clock frequency than a fully connected network or a bus.

Even though the throughput of the ring architecture can be as high as the fully connected network or the bus, a choice of $n_s < N-1$ will result in the incomplete propagation of spikes, since it takes a minimum of $N-1$ clock cycles for a spike to be propagated through the ring. As a result, some of the spikes are lost. Another consequence of the serial spike propagation along the ring is that the inhibitions due to neuron spikes do not take effect immediately, allowing neuron potentials to grow without inhibitions and fire mistakenly. The delayed inhibitions eventually take effect to suppress the spike rate later in the propagation. The two factors, spike losses due to incomplete spike propagation, and neuron misfires due to delayed inhibitions, worsen the fidelity of sparse coding.

5.2 Impacts of scalable architectures

The use of arbitration-free bus can improve the throughput and reduce hardware costs over conventional AER designs. However, collisions caused by no arbitration may degrade the image fidelity. On the other hand, the scalability of the ring is appealing for the design, but the neuron misfires for an improved throughput need to be resolved to provide a good image fidelity. This section presents an arbitration-

free architecture, analyzes collision rate of the bus architecture to confirm a practical use of the arbitration-fee design, and proposes a latent ring architecture to mitigate effects of the neuron misfires.

5.2.1 Arbitration-free bus

The power-consuming arbitration and higher bus speed are used to resolve spike collisions. If the collisions can be tolerated, arbitration is removed and the bus will run at the same clock speed as neurons, leading to a more efficient bus architecture. The sparse and independent neuron firing backed by the SAILnet algorithm [76] is promising, as the spike rate is kept low and the spikes are random, making the collision rate much lower than a conventional neural network. It is then plausible to adopt an arbitration-free bus that tolerates spike collisions.

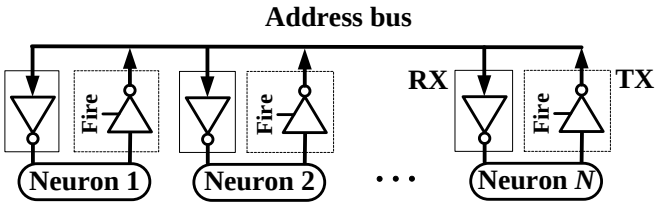


Figure 5.3: Neuron communication via arbitration-free bus.

An arbitration-free bus architecture can be designed as in Fig. 5.3. Each neuron is equipped with a tri-state transmitter (TX) and an inverter as receiver (RX). The TX and RX are assumed to be symmetrical, i.e., the pull-up and pull-down strength are balanced. (The assumption is used to simplify the analysis. The synchronous communication scheme does not require the perfect matching of TXs and RXs for the correct operation. However, mismatches will reduce the bus speed.) When a neuron fires, it puts its address (composed of multiple bits) on the shared neuron address bus. All neurons are attached to the address bus. Upon detecting an address-event, neurons will read the address and perform integrate and fire. The bus runs at the same clock frequency as the neurons. A collision occurs when multiple neurons spike

at the same time, as they will all attempt to drive their addresses onto the bus. We assume a collision resolution scheme to match the implementation in Fig. 5.3: if a bus line is driven by multiple neurons, the pull-up and pull-down strength determine the "winning" bit. For example, if there are x neurons pulling up a line and y neurons pulling down a line, the winning bit will be 1 if $x > y$, 0 if $x < y$, and a random draw if $x = y$ (as the voltage level will be in the undetermined region and the RX output will be determined by noise). The throughput of the bus architecture is f_{clk}/n_s image patches per second, same as the fully connected network.

5.2.2 Spike collision analysis

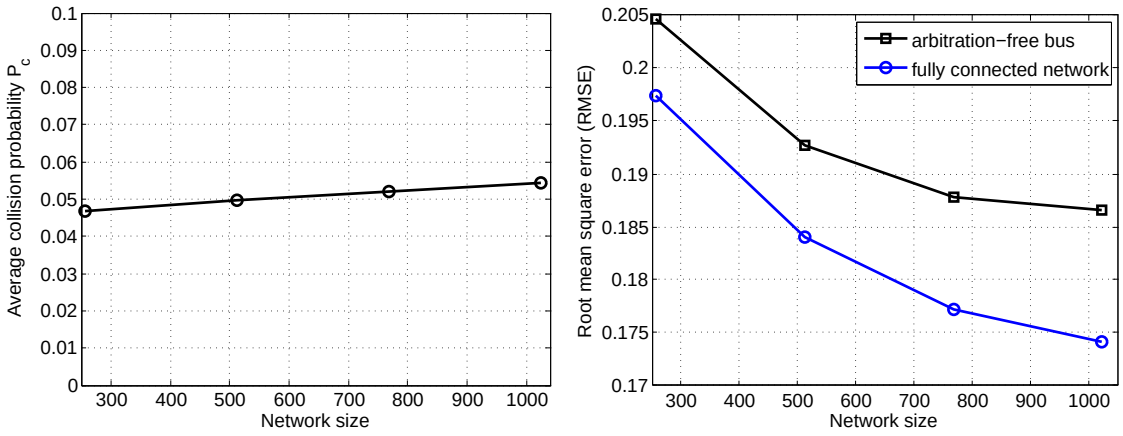
A collision on the neuron address bus results in spike corruption. The winning neuron address in a collision may not match any of the competing neurons involved in the collision. Therefore, the fidelity of sparse coding will be sacrificed. To minimize the degradation, the collision rate needs to be kept low.

The spike collision probability can be analytically derived by assuming that the spikes are independent. The independent spikes assumption is backed by the SAILnet algorithm [76]. A collision occurs when two or more spikes occur in the same time step, thus the average collision probability in each time step is given by

$$P_c = 1 - (1 - p_s)^N - Np_s(1 - p_s)^{N-1},$$

where N is the size of the network, which is assumed to be large, and p_s is the spike rate in each time step, which is assumed to be low and close to 0. By Taylor series expansion of $(1-p_s)^N$ and $(1-p_s)^{N-1}$ at $p_s = 0$, and keeping only the first two terms in each expansion, we get

$$P_c \approx 1 - (1 - Np_s) - Np_s(1 - (N - 1)p_s) \approx (Np_s)^2,$$



(a) (b)

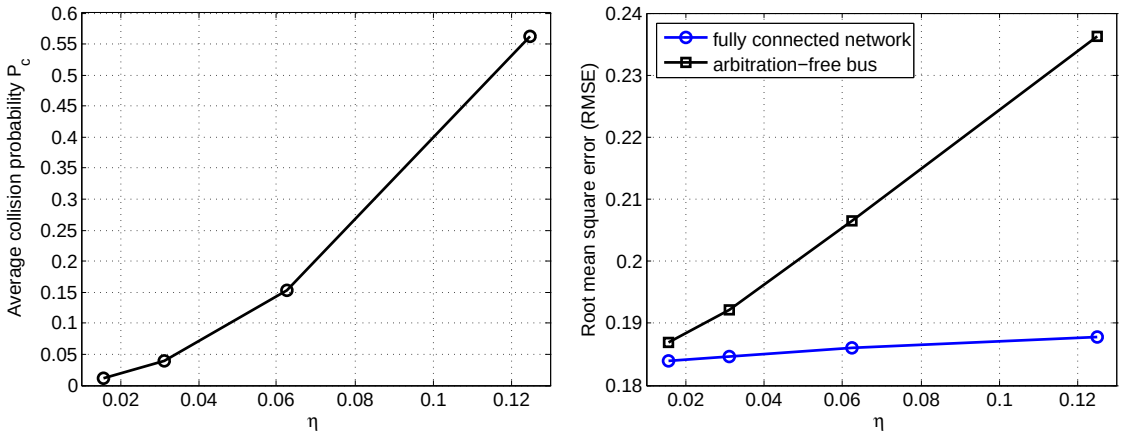
Figure 5.4: (a) Collision probability in 256, 512, 768, 1024-neuron arbitration-free bus with $p = 0.09$, 0.045, 0.03, and 0.0225, respectively, and (b) corresponding RMSE of image reconstruction by the linear generative model.

Assume that p_s is approximately the target firing rate p averaged over the n_s time steps, i.e., $p_s \approx p/n_s$, the above equation can be written as

$$P_c \approx (Np/n_s)^2. \tag{5.1}$$

Note that Np is the total number of spikes over the inference window, which remains approximately constant if the target firing rate p is optimally set based on the size of the network, as discussed previously. The result (5.1) suggests the collision probability's quadratic dependence on the total number of spikes within the inference window averaged over the number of time steps. In a network of 512 neurons with a target firing rate of $p = 0.045$, $n_s = 96$, the average collision probability $P_c \approx 5.8\%$, which is low.

As an experimental verification of (5.1), the arbitration-free bus is first trained to learn the feed-forward and feedback connection weights and firing thresholds, and then used to perform inference. Fig. 5.4(a) shows the average collision probability of the four networks of size $N = 256$, 512, 768, and 1024 with a target firing rate $p =$



(a) (b)

Figure 5.5: (a) Collision probability in a 512-neuron arbitration-free bus with $p = 0.04$, and (b) corresponding RMSE of image reconstruction by the linear generative model.

0.09, 0.045, 0.03, and 0.0225, respectively. The neuron update step size $\eta = 2^{-5}$, and the number of time steps $n_s = 96$. The average collision probability of each of the four networks is about 5%, which agrees with the analytical result. The low collision probability results in a small increase of the RMSE in image reconstruction using the linear generative model, as shown in Fig. 5.4(b).

The dependency of the collision probability on η is plotted in Fig. 5.5(a) for a network of 512 neurons that perform inference with a target firing rate $p = 0.04$. The neuron update step size η is varied from 2^{-4} to 2^{-6}, and the average collision probability decreases quadratically, confirming the analytical result in (5.1). A small η helps spread spikes over more steps, resulting in a much lower collision probability and RMSE, as shown in Fig. 5.5(b).

The efficient arbitration-free bus architecture provides the same throughput as a fully connected network. It also avoids using AER that requires a bus arbiter and a higher bus speed. To implement the arbitration-free bus architecture, the neuron update step size η needs to be kept sufficiently low to maintain a low spike collision rate for a good RMSE.

5.2.3 Latent ring

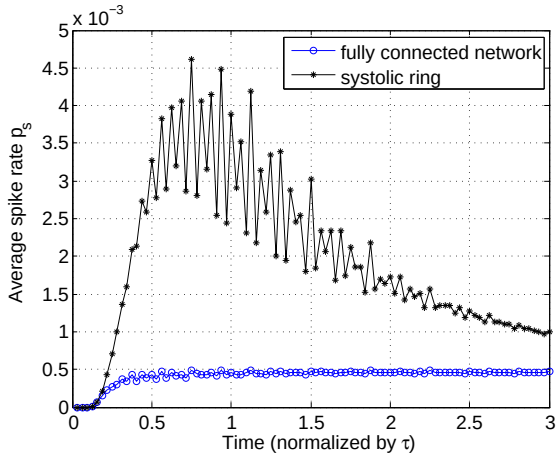


Figure 5.6: Average spike rate at each time step across a 512-neuron ring when performing inference with a target firing rate of $p = 0.04$.

Unlike in the bus where the spikes reach all neurons within one clock cycle, the serial spike passing in the ring takes much longer to reach all neurons. Neuron potentials will grow, and without seeing spikes immediately, the potential will grow to high levels, resulting in many more spikes. The majority of the spikes are actually misfires, which contribute to errors. The spike rate pattern of a network of $N = 512$ neurons in a ring with update step size of $\eta = 2^{-5}$ and number of update steps $n_s = 96$ is shown in Fig. 5.6. The distinctive spike rate pattern of the ring is compared with the fully connected network, showing the spike rate grows up to an order of magnitude higher due to neuron misfires, followed by a depression as the inhibitions take effect to suppress the spikes. Note than since $n_s < N - 1$, spikes do not reach all neurons. The neuron misfires and spike losses result in a high RMSE.

The serial spike passing along the ring slows down the inhibitions, whereas the neurons are active in every update step and ready to fire. The mismatch between the inhibitions and excitations cause many neurons to misfire. To improve the fidelity of sparse coding, the inhibitory and excitatory effects need to be balanced. A simple scheme is to implement a holding policy: each neuron is allowed to propagate one

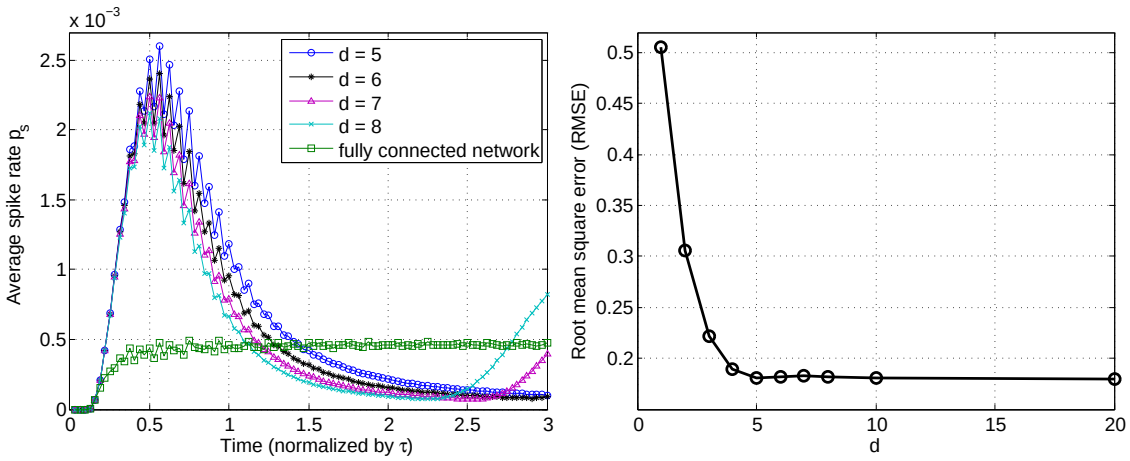
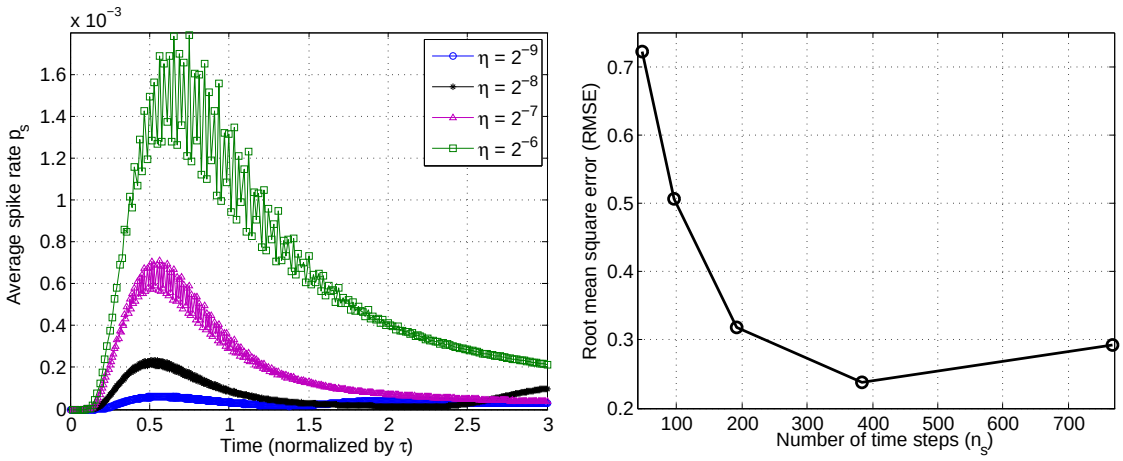


Figure 5.7: (a) Average spike rate of a 512-neuron ring with holding, and (b) corresponding RMSE of image reconstruction by the linear generative model.

spike every cycle, but perform inference update only once every d cycles. For example, setting $d = 2$ will allow each neuron to update once every two cycles. The number of inference steps is still n_s, so the spikes will propagate to dn_s neurons. The holding policy reduces the excessive excitatory strength. Longer spike propagation also reduces spike losses. As Fig. 5.7(a) shows, after implementing holding, the misfire rate is lower and the RMSE improves until d reaches about 6, as shown in Fig. 5.7(b). Note that in the example of $N = 512$ and $n_s = 96$, when $d = 6$, $dn_s \approx N$, i.e., spikes are allowed to propagate around the entire ring, eliminating spike losses.

The holding policy is one way of implementing a "latent" ring that damps neuron responses to adapt to the slow spike propagation. During holding, each neuron disables integrate and fire. To implement holding of d cycles, a d-entry memory needs to be added to each neuron to store up to d address-events. The latent ring decreases the throughput to $f_{clk}/(dn_s)$ image patches per second, where dn_s should be set to close to N for the best RMSE.

An alternative approach to damp the neuron response is to use a smaller neuron update step size η. A smaller η increases the number of update steps n_s for a fixed



(a) (b)

Figure 5.8: (a) Average spike rate of a 512-neuron ring by changing update step size η, and (b) corresponding RMSE of image reconstruction by the linear generative model.

window w. Fig. 5.8(a) shows the spike rate pattern as η is reduced from 2^{-4} to 2^{-9}, the misfire rate is reduced significantly. The RMSE improves with a lower η, as indicated in Fig. 5.8(b), and the best RMSE is reached between $\eta = 2^{-7}$ and 2^{-8}, or when $n_s \approx N$.

A latent ring architecture is scalable. The serial spike propagation slows down the inhibitions and neuron responses are damped to match the slow spike propagation for a good RMSE. The damping decreases the throughput, but the clock frequency of the ring architecture is higher than the bus architecture.

5.3 Hierarchical architecture

The arbitration-free bus architecture is efficient and provides high throughput but its clock speed is limited by capacitive loading, so the number of neurons that can be connected to a bus is limited. The latent ring architecture is scalable, but the extra delay introduced hurts the throughput. Therefore, we propose to combine bus and ring in a hybrid architecture to achieve the best of both.

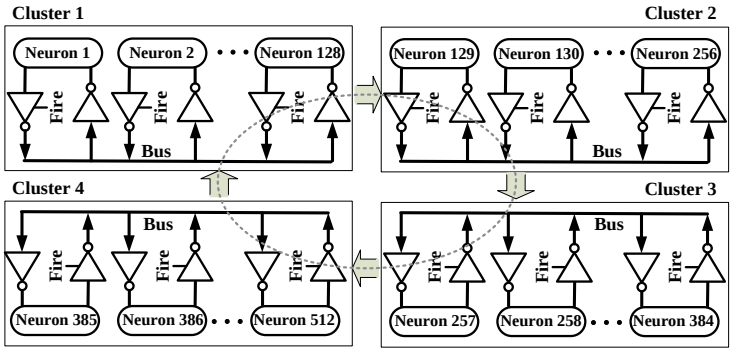
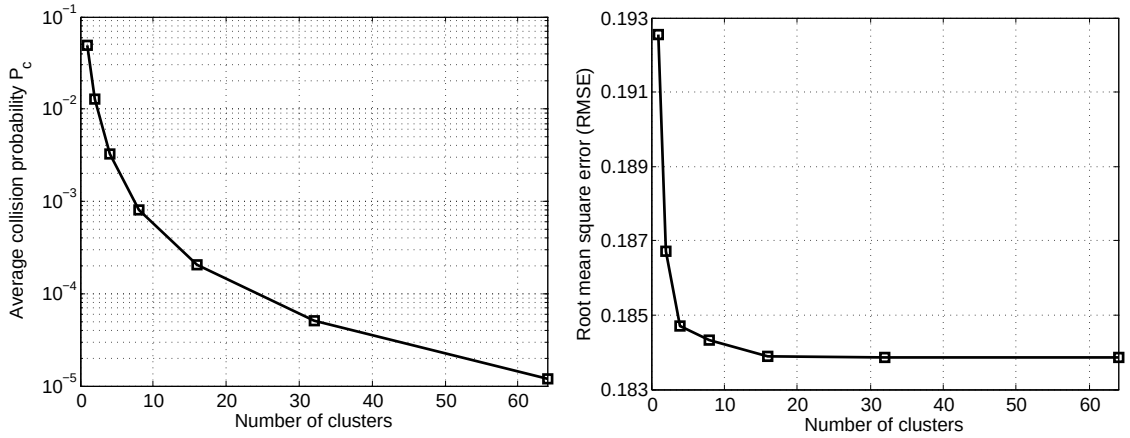


Figure 5.9: A 512-neuron 2-layer ring-bus architecture, consisting of 4 neuron clusters.

5.3.1 Hybrid bus-ring architecture

The hierarchical bus-ring architecture is implemented in a two-level hierarchy, as illustrated in Fig. 5.9. At the first level, neurons are grouped into local clusters, and the neurons in a cluster are connected in a bus. The cluster size is limited to control the capacitive loading of the bus to maintain a high clock speed. At the second level, a small number of clusters are connected in a short ring. The length of the ring is kept short to minimize the communication latency. The spike address events are generated in local clusters and then propagated through the ring to broadcast to all other clusters. The throughput of the hybrid architecture is f_{clk}/n_s image patches per second. The hybrid architecture is scalable as buses are kept short and local, and the communication is faster than a global bus or a long ring.

With short local buses in the hybrid architecture, the probability of spike collisions is reduced. Using equation (5.1) and assuming that the spikes are independent, the collision probability is reduced quadratically with more clusters for a fixed neural network size, as delineated in Fig. 5.10(a). With a reduced collision probability and faster spike propagation through a short ring, the hybrid architecture allows for an efficient implementation of SAILnet sparse coding algorithm to achieve an excellent RMSE shown in Fig. 5.10(b). The next section compares hardware implementation results of the hybrid architecture with those of the arbitration-free bus and the latent



(a) (b)

Figure 5.10: (a) Collision probability in the arbitration-free bus in a different number of clusters, and (b) corresponding RMSE of image reconstruction by the linear generative model. A 512-neuron network with p=0.045 and $n_s = 96$ is considered.

ring demonstrate its advantages.

5.3.2 Chip synthesis results

As a proof of concept, we synthesized a 512-neuron network for sparse coding in a TSMC 65nm CMOS technology. One design was implemented in an arbitration-free bus architecture, one was implemented in a latent ring architecture with a holding factor $d = 4$, and a third design was implemented in a hybrid bus-ring architecture with a 4-stage ring connecting 4 128-neuron buses.

Learning and inference are both soft operations that are intrinsically noise tolerant. The feed-forward Q weight and the feedback W are quantized to 4 bits with minimal impact on RMSE, saving significant memory and complexity. To improve the area utilization, the W and Q memories of 32 neurons are grouped together to increase the word size and amortize the addressing overhead. Memory grouping is feasible in a bus architecture or a hybrid architecture because all neurons in a bus or a cluster (in the hybrid architecture) will be accessing the same address in the Q memories

in response to a pixel input, and the same address in the W memories in response to a neuron spike. However, memory grouping is not possible in a ring architecture, because each neuron receives spikes at the different times, and the memory accesses are not synchronized. With a holding policy, neurons perform updates every d cycles, so the memories of d neurons are grouped together to save area. The three designs are synthesized using Synopses Design Compiler, and place-and-routed using Cadence Encounter. The results are compared in Table 5.1, where core area refers to the area of the design excluding peripheral circuits, such as clock generation, testing circuits, and IO pads. The designs are built using standard cells and SRAMs, and the memory capacity and standard cell count are also reported.

The area of the bus and the hybrid architectures are smaller than the ring because memories can be grouped together into larger arrays to save area. The bus architecture runs at a maximum clock frequency of 357MHz (2.8ns clock period), limited by the capacitive loading of the bus. The ring and the hybrid architecture can run faster at up to 385MHz and 370MHz, receptively. Note that the 512-neuron network is still relatively small to see a notable difference in clock frequency. For a larger network, we expect the difference will be more pronounced.

At 357MHz, the power consumption of the three architectures can be compared. The ring architecture consumes the highest power, due to the smaller memory arrays that incur a higher overhead. The bus architecture consumes less power with the help of larger memory arrays that amortize the overhead. The hybrid architecture consumes the least power due to larger memory arrays and a reduction in bus loading. At 357MHz, the throughput of the bus architecture and the hybrid architecture are 952Mpx/s, which is fast enough to process 3980×3980 image frames at 60 frames per second at a low energy consumption of 0.522nJ/px for the bus architecture and 0.486nJ/px for the hybrid architecture. The ring architecture consumes higher energy due to a lower throughput.

Table 5.1: 65nm CMOS Chip Synthesis Results

Architecture	Bus	Ring	Hybrid
Network size	512	512	512
Memory size	1.5Mb	1.5Mb	1.5Mb
Core area	3.47mm^2	4.08mm^2	3.47mm^2
Frequency	357MHz	357MHz	357MHz
Standard cells	461,691	502,821	482,862
Power	497mW	538mW	463mW
Throughput	952Mpx/s	238Mpx/s	952Mpx/s
Energy	0.522nJ/px	2.261nJ/px	0.486nJ/px
RMSE	0.192	0.189	0.185

The ring architecture is more scalable than the bus architecture. To support an even larger network, the area and power of the ring architecture are expected to scale up linearly, and the energy per pixel is expected to stay relatively constant. However, the ring delays spike propagation, which lowers the throughput. The hybrid bus-ring architecture divides the bus into smaller segments using a hierarchy, which is more scalable than a flat bus architecture, and it improves the throughput and RMSE over the latent ring architecture.

5.4 Summary

In this chapter, we present the design of efficient network architectures for the SAILnet sparse coding algorithm. Impacts of the architectures are analyzed, and improvements have been proposed.

For a practical implementation of SAILnet, three network architectures are considered: a bus architecture that provides a shared medium for neuron communications, and a ring architecture that serializes neuron communications. The bus architecture results in spike collisions and requires access arbitration. We show that the collision

rate is quadratically dependent on the number of spikes averaged over the number of neuron update steps. Keeping the spikes sparse and random by a small neuron update step reduces the collision rate to about 5%, low enough that the errors due to collisions are tolerated by the SAILnet algorithm with only a small impact on the RMSE. We design an efficient arbitration-free bus architecture that tolerates spike collisions, and removes the AER and access arbitration that are necessary for a conventional bus architecture.

A conventional ring architecture propagates spikes serially, delaying inhibitions and causing neurons to misfire. The misfires are fundamentally due to the mismatch between slow inhibitions and fast neuron responses. To reduce the neuron misfires, the neuron responses are damped by a holding policy in a latent ring architecture, where each neuron is allowed to propagate one spike every cycle, but only allowed to perform inference update once every d cycles ($d > 1$). Alternatively, the neuron responses are naturally damped by a small update step size.

The arbitration-free bus architecture and the latent ring architecture are combined in a hybrid bus-ring architecture to achieve a better scalability than the bus architecture, and a higher throughput than the ring architecture. Synthesis, place-and-route in 65nm CMOS show that the hybrid architecture occupies the same area as the bus architecture, and it consumes the lowest power. At 357MHz, the hybrid architecture achieves a throughput of 952Mpx/s at 0.486nJ/px. The proof-of-concept designs demonstrate the high throughput and energy efficiency of practical implementations of sparse coding.

CHAPTER VI

Sparse coding ASIC implementation

In this chapter, we describe the design of a sparse coding ASIC. The ASIC chip

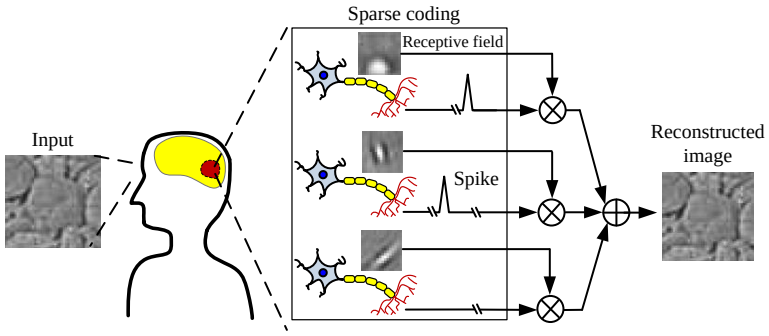


Figure 6.1: Sparse coding mimicking neural coding in the primary visual cortex. The input image can be reconstructed by the weighted sum of receptive fields of model neurons.

implementation and testing are done in collaboration with Phil Knag and Thomas Chen.

Sparse and independent local network (SAILnet) [76] learns RFs through training a network of model neurons, and infers the sparse representation of the input image using the most salient RFs, as illustrated in Fig. 6.1. Inference based on the learned RFs enables efficient image encoding, and detecting features and objects [9, 88]. However, the implementation of an energy-efficient high-throughput sparse coding processor is faced with challenges of on-chip interconnect and memory bandwidth

This chapter is based in part on [32, 33].

to support the parallel operations of hundreds or more model neurons. Existing hardware designs cannot be adapted for sparse coding [43, 42], and they often resort to off-chip memory and processing [42, 41, 45].

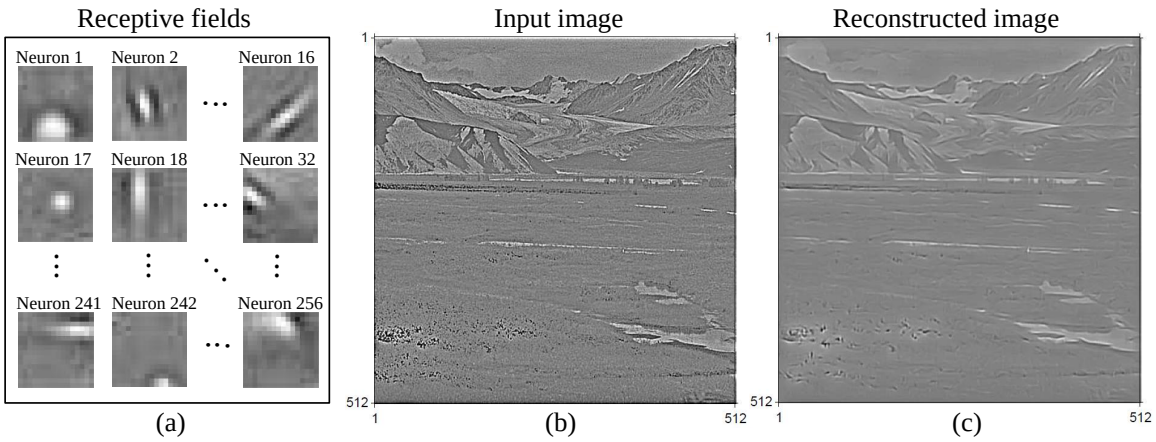


Figure 6.2: (a) Receptive fields learned by model neurons through training, (b) an input image presented to the sparse coding ASIC, and (c) the reconstructed image based on the neuron spikes obtained by inference.

We develop the first fully integrated sparse coding ASIC that consists of 256 digital neurons, 64K feed-forward synapses, and 64K feedback synapses. The sparse coding chip performs both unsupervised learning and inference on-chip. The RF of each model neuron is initialized with random noise, and each neuron learns its RF through training images. After learning converges, the chip is able to perform inference to encode images by the sparse activation of neurons, i.e., neuron spikes. To check the fidelity of inference, the input image can be compared with its reconstruction by the weighted sum of the RFs of the activated neurons Fig. 6.2.

6.1 Simulation of hierarchical architecture

Hierarchical architectures [41, 46, 45, 47, 48, 89] provide scalability and flexibility for the mapping of large-scale neural networks. In this section, we present a 256-neuron hierarchical bus-ring architecture for the SAILnet sparse coding algorithm.

The architecture includes two levels of hierarchy. In the bottom level, an arbitration-free bus [31] is used and in the top level the ring connects the output of buses, as illustrated in Fig. 5.9. Neurons are placed in the bottom level and separated into several neuron groups. In each group, neurons are connected to an address bus. A spike is encoded into a neuron ID (NID) for communications. An NID generated ascends to the top level of the hierarchy and is propagated through the ring, and neurons in the bottom level receive NIDs from the top level by tapping their corresponding ring nodes.

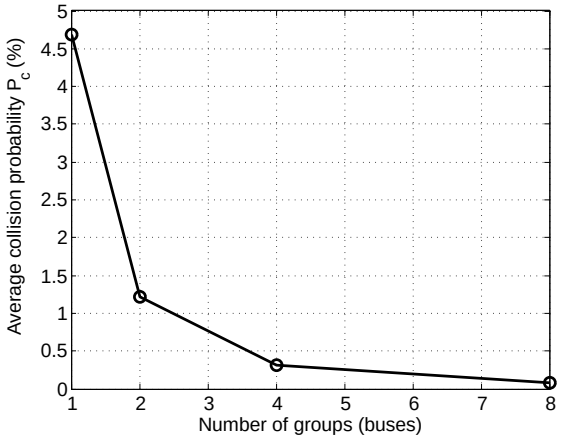


Figure 6.3: Collision probability of the hierarchical ring-bus architecture made up of 256 spiking neurons.

The bus architecture causes collisions if multiple neurons spike simultaneously. The collisions corrupt a resulting NID [31]. The occurrences of the collisions can be vastly reduced in a smaller neural network. The collision probability on the address bus decreases quadratically as the neuron group size is reduced, as discussed in Section 5.3. The quadratic relationship is verified in simulation, shown in Fig. 6.3. In the simulation, a 256-neuron network is trained by presenting 16×16 patches extracted from natural images. The target firing rate is $p = 0.09$ and the number of time steps is $n_s = 96$. Note that as we increase the number of groups, the hierarchical architecture becomes the ring architecture and the average collision probability on the address bus approaches zero.

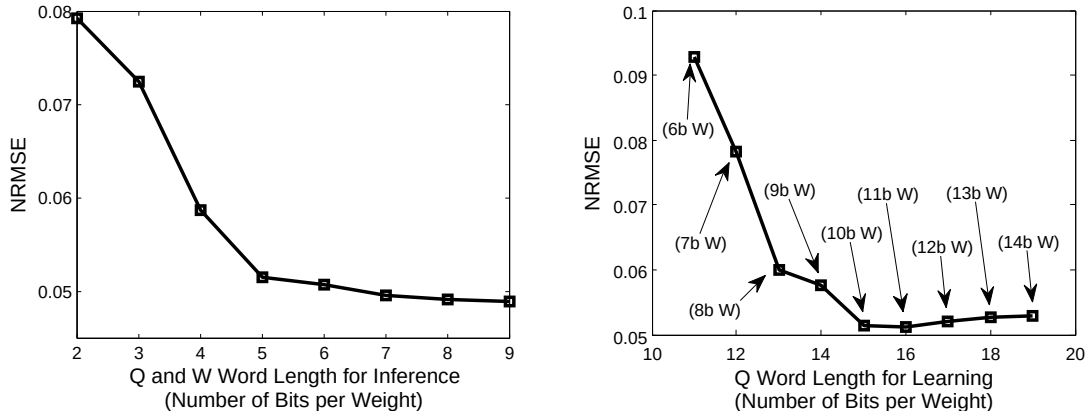


Figure 6.4: (a) Q and W weight quantization for inference and (b) learning.

Though the fixed-point quantization reduces the hardware cost and provides a fast processing capability, a floating-point to fixed-point conversion may degrade the algorithm performance. This section applies the fixed-point conversion to the SAIL-net algorithm, and empirically verifies the feasibility of the conversion. Fig. 6.4(a) shows the RMSE error of reconstructed image by applying fixed-point quantization to receptive field (Q) and feedback (W) memory during inference, and Fig. 6.4(b) shows the RMSE with different word length of Q and W weights. The results are from the simulation of a ring-bus architecture implemented in MATLAB where the bus size is 64, and the ring size is 4. The baseline comparison is the reconstructed image of the floating-point quantized algorithm in a 256-neuron network that is fully connected [31]. A 4-bit Q weight and a 4-bit W weight in inference and a 13-bit Q weight and a 8-bit W weight in learning still provide a good sparse coding performance [33]. Based on the quantization study, in the next section we design a practical hierarchical architecture for sparse coding.

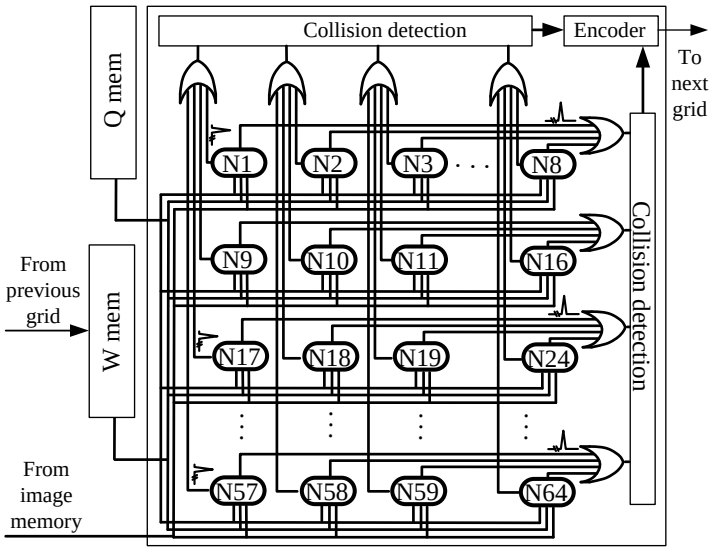


Figure 6.5: 2D bus of a cluster of 64 neurons. Spike collisions are detected and tolerated to save power.

6.2 Architectural design

Each model neuron in the sparse coding chip is a compute node that performs leaky integrate-and-fire [76]. In this section, a two-layer network is designed to allow all neurons to communicate efficiently. First, we address a local 2-D bus design in the first level of the hierarchy. A cluster of 64 neurons are connected in a 2D grid that improves the communication delay over a 1D bus. The root nodes of four grids are connected in a 4-stage systolic ring in the second level. The grid size (or the number of neurons connected in a grid) is designed to limit the wire loading for sub-ns timing and bound the spike collision rate; and the ring is kept short to reduce latency. The learning part of sparse coding has usually been implemented off-chip due to the required high computational loads. We present a snooping core attached to the ring that performs approximate learning to reduce the computational loads. We verified in the previous section that a 5% or lower collision rate is tolerated without causing any noticeable degradation in fidelity.

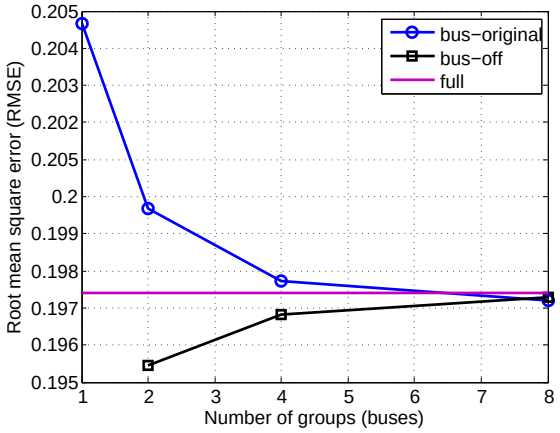


Figure 6.6: RMSE of the reconstructed images for an arbitration-free 2D bus design, an arbitration-free flat bus design, and fully-connected network.

6.2.1 Arbitration-free 2D bus

A conventional design of the bus architecture contains a neuron grid, row and column arbiters, and a handshake [87, 36]. The conventional design enables a parallel to serial spike transmission on the 2D address bus without collisions on the bus. However arbitration and handshake add complexity to the design of the bus, and require additional memory for spike transmissions. Alternatively, an arbitration-free bus design presented in Chapter V is more efficient to implement.

For an efficient bus design, we propose a new arbitration-free 2D bus structure. A conventional 2D bus is used to place neurons and each neuron is connected to its row and column address buses. Once a neuron spikes, its NID is loaded to the buses. Different from others [87, 36], we simplify the collision detection and management by turning on the bus if only one neuron in the grid spikes and turning it off otherwise. This scheme allows us to remove the row and the column arbiters and the handshake. A 64-neuron arbitration-free 2D bus is illustrated in Fig. 6.5. An OR gate outputs 1 if any neurons in the same row/column spike. The row/column collision detection block calculates the number of OR gates that output 1. When the number calculated by either the row or the column collision detector is greater than or equal to two, we

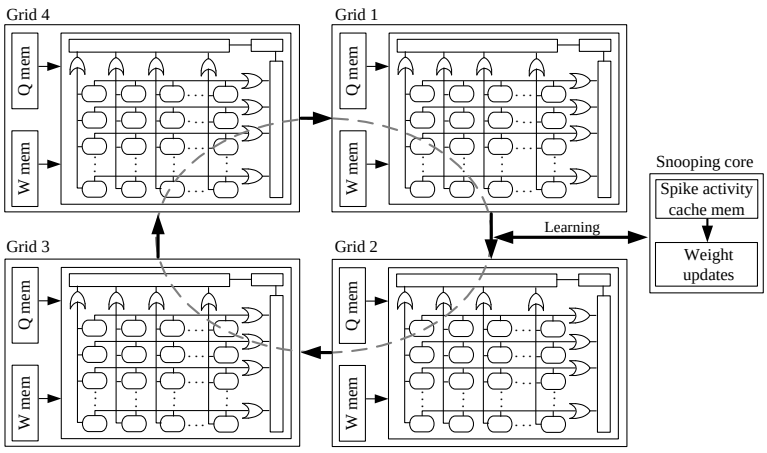


Figure 6.7: 4-stage systolic ring connecting 4 2D local buses. A snooping core is attached to the ring to record neuron spikes for learning.

say that a collision is detected. Q and W memory blocks are attached to the grid. We use a broadcast pixel bus to accommodate parallel neuronal potential updates. Experiments shown in Fig. 6.3 verifies that the probability of spike collisions can be reduced by grouping a small number of neurons, so our bus design, in turn, does not degrade the performance of sparse coding, as delineated in Fig. 6.6.

6.2.2 2-layer bus-ring architecture

The output of a grid is represented as a packet {REQ, NID}, and the grids can be modeled as super neurons where their inputs are incoming packets and pixels. For a scalable VLSI implementation, we arrange the super nodes in a systolic ring structure. In this work, we build a hierarchical 256-neuron network where a ring contains 4 super nodes, each of which has 64 neurons placed in a 2D bus, as illustrated in Fig. 6.7.

During inference, neurons in each neuron grid listen to the packets transmitted from neighboring nodes and update their neuronal potentials by considering feedback connection weights. We allow each neuron to fire every clock cycle assuming that the packets propagate to the next ring node in one clock cycle, and the potential update is done in one clock cycle. An NID is discarded once it returns to the grid where it is generated.

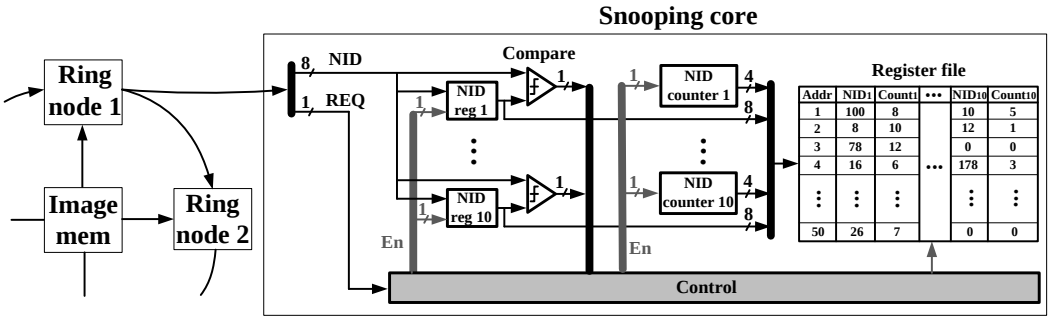


Figure 6.8: Recording of NIDs and their spike counts during inference. The snooping core stores NIDs and spike counts to the register file for 50 patch presentations.

6.2.3 Snooping core and approximate learning

Learning and inference make use of the same network of model neurons, but learning also updates Q and W weights, which dictates the learning speed. Learning is done by a snooping core attached to the top-level ring to listen to neuron spikes and record the activities in a cache, illustrated in Fig. 6.7. After a batch of training images, the snooping core reads the cache and makes weight adjustments according the SAILnet learning rules [76]. To accelerate learning and reduce the cache size, we implement approximate learning to record the activities of only a few neurons that spike for each input image patch, as illustrated in Fig. 6.8. In our design, the snooping core records 10 dominant NIDs and their spike counts, and the cache has a capacity of 50 patches. Experimental evidence points to the fact that the neurons that spike first tend to be the most active. The remaining neuron activities play a minor role, and can be safely ignored.

Initiated by the loading of NIDs and their spike counts from the activity cache, learning performs three updates: update of feed-forward connection weights Q; update of feedback connection weighs W; and update of the threshold. Updating Q is the most computationally intensive as the update depends on the input image, current weights, and spike counts, as formulated in (4.12). To reduce hardware cost and to achieve fast Q update, we approximate the Q update in two ways. One is to allow

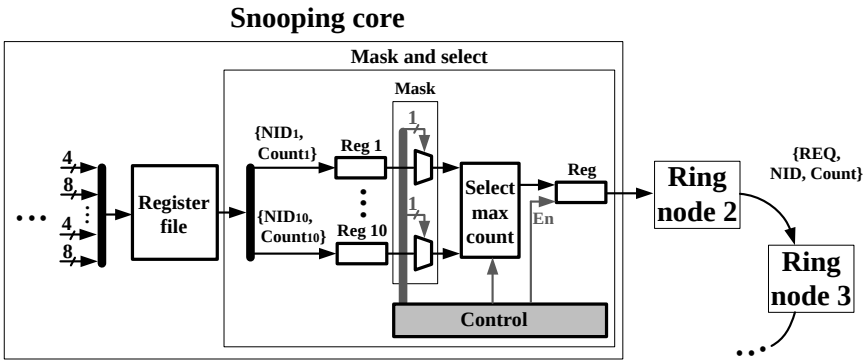


Figure 6.9: Approximation of the Q update. Transmit a maximum count to each ring node.

Table 6.1: Mapping of a square of the spike count to a bitshift operation

s_i	${s_i}^2$	mapping of ${s_i}^2$	bitshift (<<)
1	1	1	Do nothing
2	4	4	2
3	9	8	3
4	16	16	4
5	25	32	5
6	36	32	5
else	-	64	6

the Q update of "one" neuron per grid. To implement this scheme, we categorize recorded NIDs into four groups, each of which corresponds to the largest spike count in each group, as illustrated in Fig. 6.9. As a result, the snooping core propagates at most four packets {REQ, NID, s_{NID}}, each of which targets a ring node. Once the four nodes in the ring receive their packets, each node approximates the Q update further. The square of spike count, s^2, is mapped to one of 6 possible values by using a look-up table, summarized in Table. 6.1. Since the mapped number is a power of 2, the computation of $s^2 Q_{ik}^{(m)}$ can be approximated by a bitshift operation of $Q_{ik}^{(m)}$. Through the two approximations, the Q updates are efficiently computed, and the updates are performed in parallel over the four ring nodes.

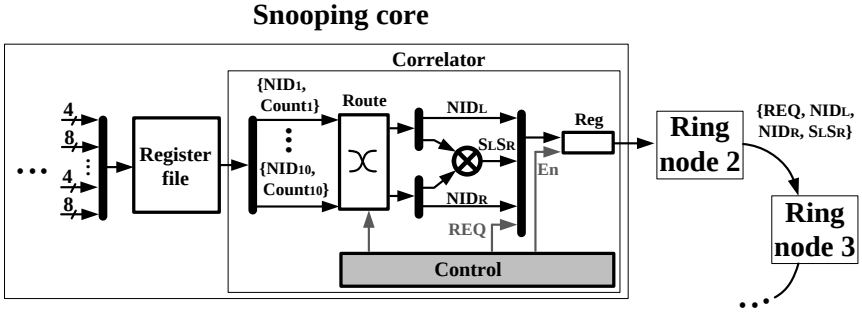


Figure 6.10: Computation of correlations for W update.

The W and threshold updates are relatively straightforward. The update of W is governed by the correlation of spike counts, as formulated in (4.11). NIDs and their spike counts in the memory are loaded and the correlation is recursively computed, as illustrated in Fig. 6.10. The snooping core sends out packets, each of which is defined as {REQ, $\mathrm{NID_L}, \mathrm{NID_R}, s_{\mathrm{L}} s_{\mathrm{R}}$} where $\mathrm{NID_L}$ and $\mathrm{NID_R}$ are the NIDs of two neurons being connected by a feedback connection. Once the two neurons receive the packet, they update the feedback connection weight. To update the threshold of a neuron, the snooping core transmits a packet {REQ, $\mathrm{NID_{Th}}$, s_{Th}}, which targets a specific neuron whose NID is $\mathrm{NID_{Th}}$. Packets for the Q, W, threshold updates are distributed following the hierarchy from the top layer (ring) to the bottom (2D

bus). The updates are performed in parallel in the bottom layer of the hierarchy. Simulation in MATLAB confirms that the approximate computing for learning does not degrade the algorithm performance.

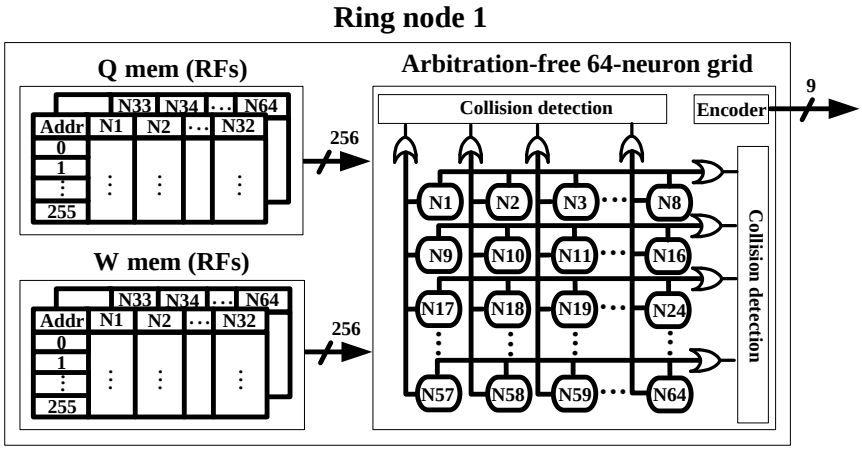


Figure 6.11: Illustration of register files (RF) banks in the ring node 1 for the inference.

6.2.4 Memory partition

Each neuron in the neuron grid can be considered as a computing unit, and it stores RFs, termed Q weights (16×16 image patch) and synaptic strengths, termed W weights (256 entries for each neuron). A high throughput sparse coding can be achieved if all neurons in the network work as parallel computing units with parallel memory accesses. Through fixed-point quantization and an optimal memory organization, we allow all neurons in the network to compute in parallel with an efficient memory access.

The SAILnet algorithm consists of an inference phase and a learning phase. During inference, the memory read operation needs to be synchronized over all model neurons in the network. When a pixel value is provided to a neuron grid, neurons in the grid need to read their Q weights concurrently to calculate excitations. Furthermore, when neurons in a grid see a spike transmitted by its previous grid, they need to read W weights at the same time to compute inhibitions. For a higher memory efficiency,

we store the Q weights and the W weights of a cluster of neurons together in Q and W memory. Based on simulations, we find that a coarse quantization of 4-bit feed-forward weights and 4-bit feedback weights still provides a good performance with minimal loss in image encoding accuracy. Each grid includes register files to store weights. The register file is preferable than the SRAM for this case due to the area efficiency. 32 neurons in a neuron grid share one 32-Kbit register file to store Q weights and another 32-Kbit register file to store W weights. Each row of the register files contains 32 Q/W weights, each of which is the weight corresponding to a neuron in the grid, as illustrated in Fig. 6.11.

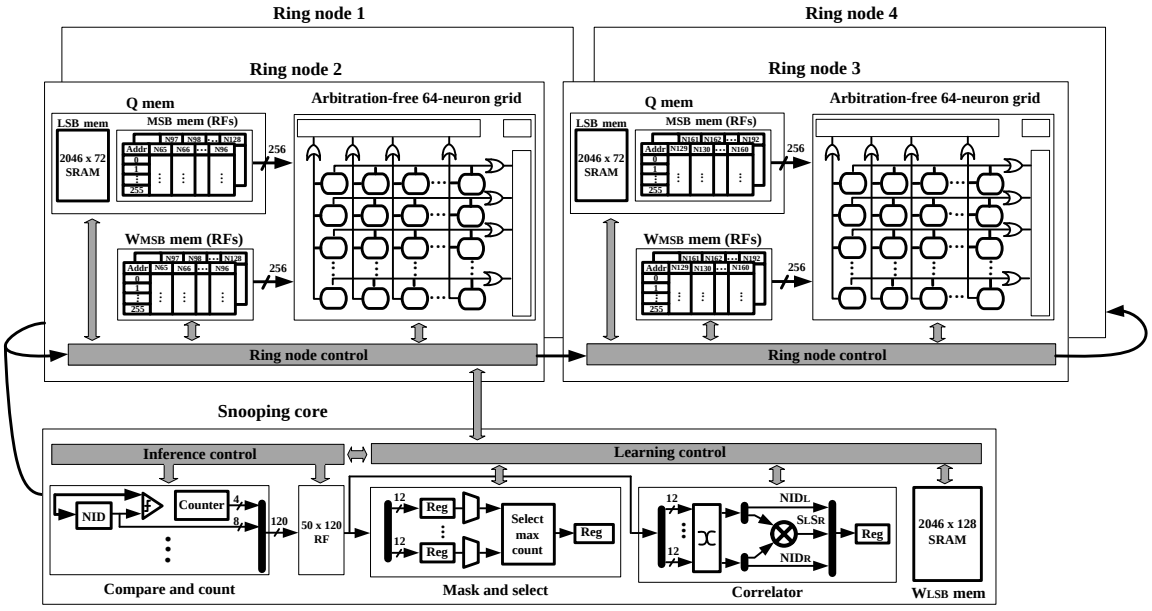


Figure 6.12: Overall design of the hierarchical bus-ring architecture and a snooping core that support inference and learning.

During learning, the weights are incrementally updated, and a high precision is required. Through the quantization study, we optimize the wordlength of Q and W weight to 13 bits and 8 bits respectively to minimize storage and guarantee convergence. The weight updates happen occasionally and always follow inference, so it is efficient for learning to share memory banks with inference. In addition to the core

Q mem and W mem, we implement Q auxiliary memory and W auxiliary memory to support learning. The auxiliary memory banks are separated from the core memory banks, and are implemented in SRAM to achieve a higher memory density and a small area. Each neuron grid has a 144-Kbit Q auxiliary memory to enable parallel updates of Q weights. Since the W update happens less frequently than the Q update, so the W update is not the bottleneck of learning. To optimize the area of the W auxiliary memory, we use one 256-Kbit SRAM for all neurons in the network. During learning, the auxiliary memory banks are updated and the carry outs are used to update the core memory.

We exploit the difference in memory usage between learning and inference and place the core memory and auxiliary memory on separate supply rails: the core section to support inference, and the auxiliary section that is only powered on for learning. The core memory is implemented in high-bandwidth register file to support real-time inference. The auxiliary memory is implemented in a lower-bandwidth SRAM to provide the extra bits needed for learning. As learning is called less frequently, the power consumption of SRAM for learning becomes negligible. On-chip learning is nonetheless orders of magnitude faster and lower power than off-chip learning. The overall design with the memory map is illustrated in Fig. 6.12. For this design, we used an ARM memory compiler to generate register files and SRAMs for the weight storage.

6.3 Pipeline of inference and learning

To obtain good receptive fields, neurons in the network continually update the feed-forward weights in response to the change of input images. They update their feedback connection weights and thresholds as well in order to maintain sparse neuron spikes in the network.

The algorithm functions in two phases: inference phase which records neuron

spiking in response to a batch of static images. We choose 50 as the batch size; and learning phase which performs the updates. We define one iteration as the inference of a batch of images followed by weight updates. Note that learning depends on inference, so inference and learning need to be serialized within an iteration. We empirically verify that the weight updates converge after about 10^4 iterations in most of the cases that we experiment with.

The inference of an image is done by computing excitations, spiking and propagating spikes, and recording spikes. The excitation of each neuron is proportional to the overlap between an input image and the receptive field (Q weights) of each neuron, and the computation is done by memory read (RE) and multiply-and-accumulation (MU). Once the excitation is computed, neurons start spiking (SP). The spiking of neurons in the network is synchronized, and a spike generated in one neuron grid is propagated to neurons in the others. During the spike propagation, the snooping core taps the ring and writes an NID of each spike to memory. The throughput of inference is $n_{\mathrm{p}} f_{clk}/n_{\mathrm{s}}$ pixel/second.

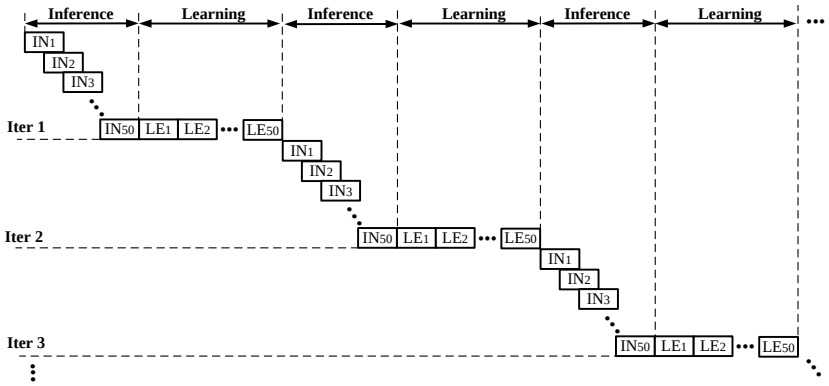
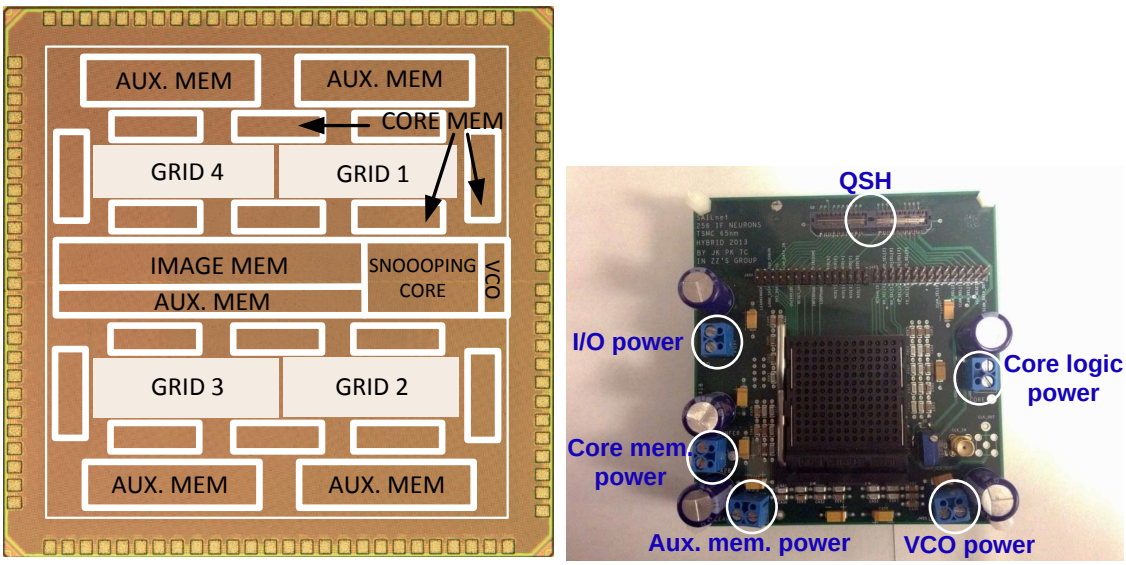


Figure 6.13: Pipeline of inference (IN) and learning (LE) of 50-patch batches.

The completion of inference of 50 patches triggers the weight and threshold updates. The updates are performed serially in patches, i.e., we update Q weights, W weights, and the thresholds by considering spikes obtained from inference of patch 1, and update them again with respect to the inference of patch 2, and so on. For

a given patch, Q weight, W weight, and threshold are updated in parallel. The Q updates are efficiently parallelized over ring nodes, which is achieved by masking and selecting a maximum spike count as discussed in the previous section. The correlator in the snooping core calculates correlations of at most 10 NIDs per patch, so we selectively update entries of W memories. To update the thresholds, NIDs and their spike counts are propagated through the ring. The pipeline chart of inference and learning of batches of 50 images is described in Fig. 6.13. The throughput of learning is $n_{\mathrm{p}} f_{clk}/(n_{\mathrm{s}} + n_{\mathrm{p}})$ pixel/second.



(a) (b)

Figure 6.14: (a) Chip photograph. (b) Printed circuit board for the testing.

6.4 Measurement results

The sparse coding ASIC test chip is implemented in TSMC 65nm CMOS, as illustrated in Fig. 6.14(a), and the chip is fully functional. 102 bond pads on a chip are wired to 84 pins in a ceramic pin grid array (CPGA) package, as illustrated in Fig. 6.14(b). The 4-layer printed circuit board has five power connectors: core logic, core memory, auxiliary memory, VCO and I/O. During inference all but auxiliary

memory are powered on, and during learning, all supplies are powered on. In the beginning of the testing, 50 16 × 16 image patches, receptive fields and feedback weights are scanned in to an SRAM. The test chip infers the input by extracting natural features of the input. The chip also learns the receptive fields by updating the feed-forward and feedback weights to adapt to changing environment.

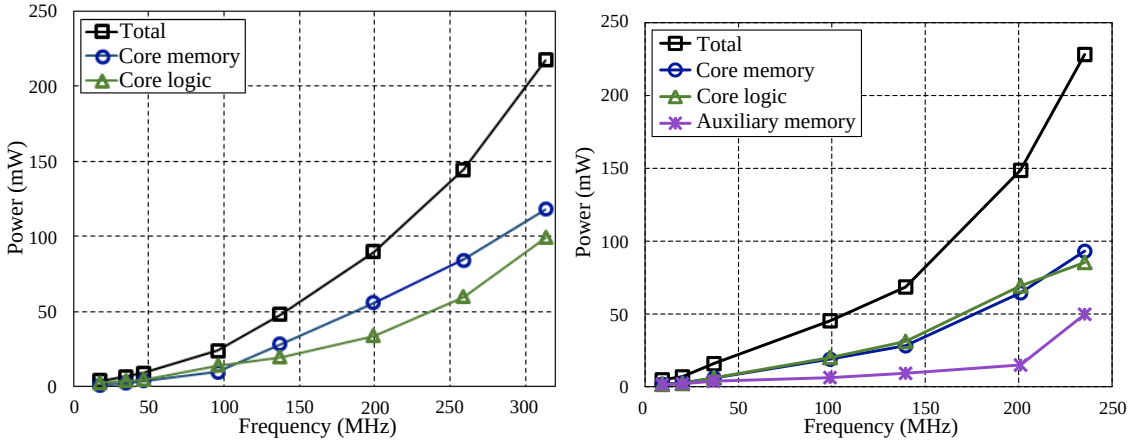


Figure 6.15: (a) Measured inference power consumption and (b) learning power consumption.

Inference is carried out in steps for each 16 × 16 input image patch. For a high fidelity, the number of steps is set to at least 64, which translates to an inference throughput of $\frac{16\times16}{64} f_{\text{clk}}$ pixel/s where f_{clk} is the clock frequency. An average power consumption of inference (P_{inf}) is measured by repeating the computations of inference for 1 million loops, and the energy efficiency of inference is $P_{\text{inf}}/((16 \times 16/64) f_{\text{clk}})$ J/pixel. Inference operates at a maximum 310MHz, consuming 218mW at 1.0V and room temperature, which translates to a maximum inference throughput of 1.24Gpixel/s (Gpx/s) at an energy efficiency of 176pJ/px, as summarized in Fig. 6.15(a) and Table 6.2.

To enable learning, the auxiliary memory is powered on, and the test chip performs inference of one image patch followed by updating weights stored in core and auxiliary memory. The number of steps for inference is set to 64, and the weight

Table 6.2: Chip summary

Technology	TSMC 65nm GP CMOS			
Core Area	1.75mm × 1.75mm (Core logic: 1.16mm^2, Core mem: 1.01mm^2, Aux. mem: 0.89mm^2)			
Chip Area	2.11mm × 2.11mm (4.45mm^2)			
	Inference		Learning	
Frequency (MHz)	35	310	20	235
Core logic (V)	0.53	1.00	0.50	1.00
Core mem (V)	0.44	1.00	0.58	1.00
Aux. mem (V)	0.00	0.00	0.60	1.00
Throughput (Mpixel/s)	140	1240	16	188
Power (mW)	6.67	218	6.83	228.1
Energy Efficiency (pJ/pixel)	47.6	175.8	426.9	1213

update is carried out in additional 256 clock cycles. Therefore, the throughput can be calculated as $(16 \times 16/(256 + 64))f_{\text{clk}}$ pixel/s. The energy efficiency of learning is $P_{\text{learn}}/(0.8f_{\text{clk}})$ J/pixel. Learning consumes 228mW at 1.0V and 235MHz for a throughput of 188Mpx/s, as shown in Fig. 6.15(b) and Table 6.2. A training set of 1 million 16×16 image patches is completed in 1.4s.

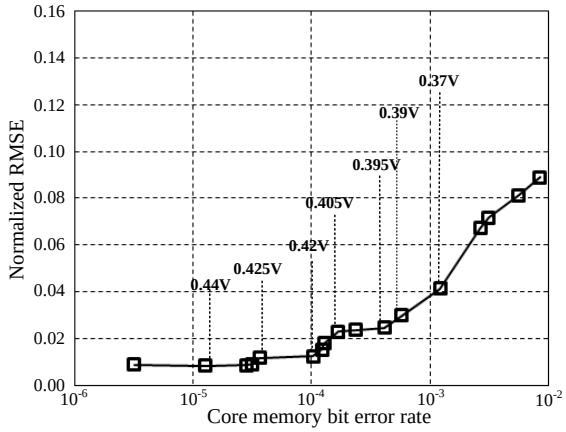


Figure 6.16: Measured normalized root-mean-square error (NRMSE) in inference with increasing core memory bit error rate. The core memory supply voltage is annotated.

The sparse coding algorithm is error tolerant, and with on-chip learning, errors can be corrected by on-line training. Our measurements indicate a gradual degradation of

the normalized root-mean-square error (NRMSE) of the reconstructed image (measure of fidelity) until the core memory supply is lowered to 390mV, where a nearly 10^{-3} core memory bit error rate results in no more than 0.03 NRMSE in inference, as illustrated in Fig. 6.16. The error tolerance is exploited to reduce power. The core memory supply voltage can be reduced to 440mV, while still keeping NRMSE within 0.01. Together with voltage scaling the core logic, the inference power consumption is reduced to 6.67mW for an inference throughput of 140Mpx/s, improving the energy efficiency 48pJ/px, as shown in Fig. 6.15(a) and Table 6.2. Learning requires writing to memory, which places lower bounds on the core and auxiliary memory supply at 580mV and 600mV, respectively. At these low supplies, the learning power consumption is reduced to 6.8mW for a learning speed of 16Mpx/s, as shown in Fig. 6.15(b) and Table 6.2. A comparison with recent literature is presented in Table 6.3. The on-chip learning capability, as well as the achieved high throughput and energy efficiency demonstrate the potential of the sparse coding ASIC for embedded vision processing tasks.

Table 6.3: Comparison with prior works

Reference	Mellola *et al.* [42]	Seo *et al.* [43]	This work
# Neurons	256	256	256
# Synapses	256K	64K	128K
Bitwidth of a Synapse	1 bit	4 bits	8 and 13 bits
Memory size	256Kbits	256Kbits	1.31Mbits
Interconnect	Crossbar	Crossbar	2-layer bus and ring
Algorithm	RBM	STDP	SAILnet
Learning	Off-chip	On-chip	On-chip
Application	Digit recognition	Pattern recognition	Image sparse coding
Technology	IBM 45nm	IBM 45nm	TSMC 65nm
Core Area	$4.2mm^2$	$4.2mm^2$	$3.1mm^2$
Energy metric	45pJ/spike	-	48pJ/pixel

6.5 Summary

In this chapter, we design a 256-neuron bus-ring architecture. Neurons in the network are divided into 4 neuron clusters, and 64 neurons in a neuron cluster are

connected with a 2D bus. Each output of the bus is connected to the 4-stage systolic ring. Each neuron cluster is designed to tolerate spike collisions. The cluster size and ring size are determined by considering the tradeoff between communication latency and image encoding accuracy. The SAILnet learning rule is further approximated to enable fast on-chip learning. To save power, memory is divided into core and auxiliary sections, and the auxiliary memory is powered off during inference.

A 65nm test chip achieves a throughput of 1.24G pixel/s at 1.0 nominal supply voltage, running at 310MHz. The test chip exploits the error tolerance of the soft SAILnet algorithm to reduce core memory supply voltage to 440mW at 35MHz, achieving an energy efficiency of 47.6 pJ/pixel.

CHAPTER VII

Neuromorphic object recognition processor

Continuing with the design of a sparse coding ASIC in the previous chapter, this chapter shows a higher-level application of sparse coding: object recognition, and its efficient hardware implementation. A sparse coding based IM for object recognition is illustrated in Fig. 7.1. The classifier attached to the IM takes sparse feature inputs, and classifies objects. In particular, we focus on the spiking LCA-based IM and a task-driven classifier due to the advantages of sparsity [75], binary neuron output [53], and high classification accuracy [90].

7.1 Simulation of spiking LCA IM

Feature extraction is enabled by parallel spiking neurons, each of which has its own feature vector or RF. If a neuron's feature vector matches features presented in the input, the neuron will spike as a result. Compared to other neural-inspired IMs such as the SAILnet IM [76], the LCA IM [75] provides two benefits. Feedback weights can be precomputed using the feature dictionary, and all neurons in the IM share the same neuron threshold. Because of sparsity imposed, the LCA IM allows only a few neurons to be activated in response to an input image, thereby reducing active power and enabling efficient neuron-to-neuron interconnect. Despite the progress on

This chapter is based in part on [34].

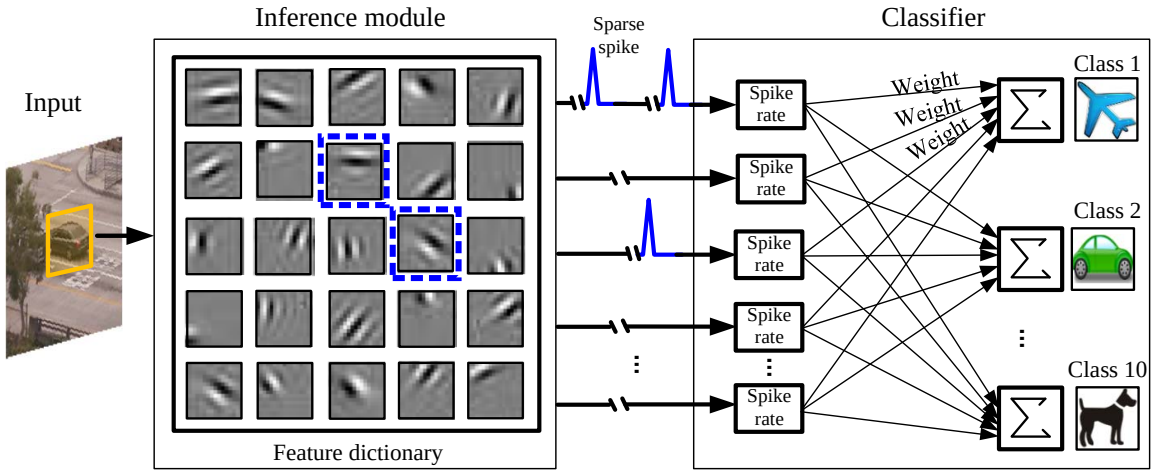
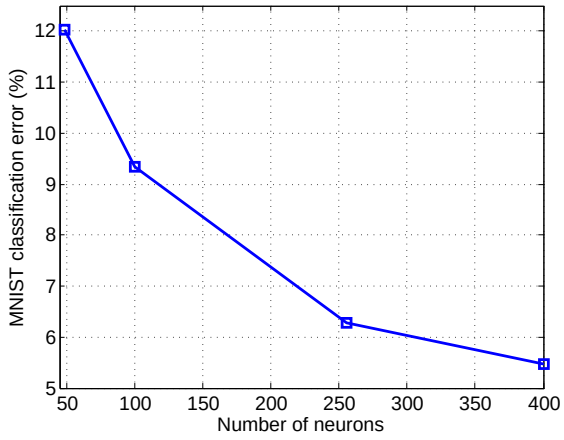


Figure 7.1: Sparse neuromorphic object recognition system composed of the spiking LCA inference module (IM) front-end and the task-driven classifier back-end. A sparse set of features are extracted to represent the input image. The weighted spiking rate is summed to vote the most likely object class.

efficient hardware architecture for sparse coding to address the routing complexity in the IM network [31, 33], a major hardware challenge still remains for implementing a large-scale neural network for LCA. The size of weight memory can be over 60% of the total chip area in order to perform on-chip inference and learning [33]. To address the memory challenge, this work uses the convolutional neural networks idea [91] to implement the spiking LCA IM [53]. In this section, we study the impacts of the RF size and the network size on classification accuracy in order to optimize the size of on-chip memory.

The size of on-chip memory in the IM is dominated by the weight memory that stores RFs and feedback weights. The size of RF memory grows linearly with the number of input pixels and the number of neurons in the network, and that of feedback weight memory grows quadratically with the number of neurons. Thus, the size of on-chip memory can be reduced if we use small input patches, and implement a small network.

To optimize the network size, we first check the performance of the conventional

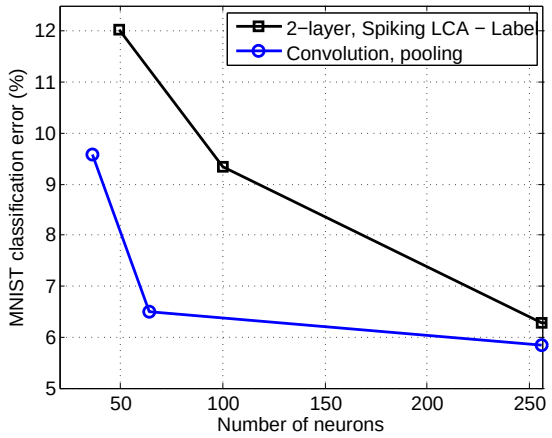


(a)

Figure 7.2: Errors in MNIST classification with different network sizes.

spiking LCA using the MNIST database [92]. Fig. 7.3 shows errors in classification with different network sizes. The spiking LCA IM is connected to a task-driven classifier where the weights of the classifier are trained using a regression model that minimizes the errors between a given label and the reconstructed label [90]. Note that learning in the IM is independent of learning of the classifier for practical hardware implementation, so the error of classification is about 5% higher than the results of the conventional task-driven learning approach [90]. We choose to implement a 256-neuron network, since the improvement of classification is negligible if the network size is above 256 in this case.

To reduce the RF size, we use the ideas of convolutional neural networks to implement the spiking LCA IM. Convolutional neural networks use a small number of RFs, and the size of each RF is less than the number of input pixels. There is already an existing approach that makes use of convolutional neural networks in implementing LCA [93]. However, this approach implements feedback weight memory based on all-to-all neuron interconnects in order to achieve overcompleteness. In comparison, we prune neuron-to-neuron connections to reduce the size of feedback weight memory by dividing the input image into overlapping patches and processing the image patches



(a)

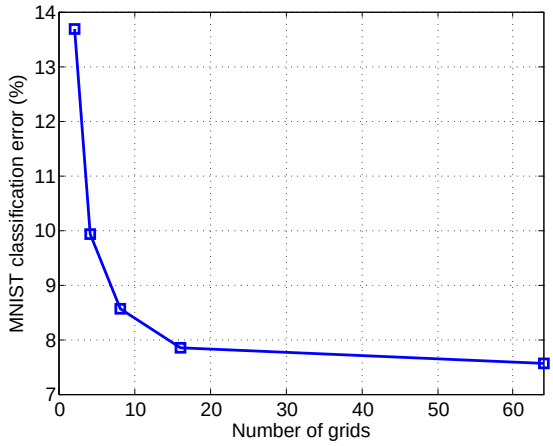
Figure 7.3: MNIST classification of the spiking LCA IM and the conventional spiking LCA.

independently. Therefore, the size of feedback weight memory is $O(N_{\mathrm{RF}}^2)$ where N_{RF} denotes the number of RFs.

In inference, the size of RF is set to 16×16. We take 20×20 pixels centered at a 28×28 MNIST test image. Fig. 7.3 compares our convolutional spiking LCA with the conventional spiking LCA. It is shown that 64 RFs using a stride of 4, i.e., 2 steps in each direction to cover an 20×20 input image, provides a comparable classification result as a conventional 256-neuron spiking LCA network. We tested the effects of smaller RF sizes such as 14×14 and 15×15. The classification error for these receptive field sizes is increased by 2% over a receptive field size of 16×16, so we choose 16×16 for the implementation. In training, we use (4.21) to update the 64 RFs.

7.2 Architectural design

An end-to-end object recognition processor consists of two major building blocks: inference module (IM) and classifier. In this section, we implement the spiking LCA IM [53] and a task-driven classifier [90]. In addition, we implement a light-weight



(a)

Figure 7.4: MNIST classification with different number of grids in the IM.

learning co-processor for on-chip learning.

7.2.1 Spiking LCA IM

A straight implementation of the spiking LCA IM includes 64 RFs and 64 digital integrate-and fire neuron to extract features in 16×16 image patches. The 64-neuron IM network is implemented using the 2-layer bus-ring architecture to address routing complexity and scalability [32, 33]. The grid size is determined by considering the tradeoff between hardware efficiency and inference accuracy. A large grid (a small number of grids) is more compact, but results in more simultaneous neuron spikes colliding over the grid, worsening the inference accuracy, as illustrated in Fig. 7.4; while a systolic ring preserves neuron spikes but a long ring costs more area and power. We choose a grid size of 8 for this IM design.

The 64-neuron digital IM is implemented in eight clusters, as illustrated in Fig. 7.5. Each cluster includes an 8-neuron grid, two collision detection blocks and an NID encoder. Eight neurons in each cluster are placed in a 2×4 grid, and are connected with column and row buses. Similar to the SAILnet IM design, the collision detection blocks listen to the column and row buses, and they disable the NID encoder if multi-

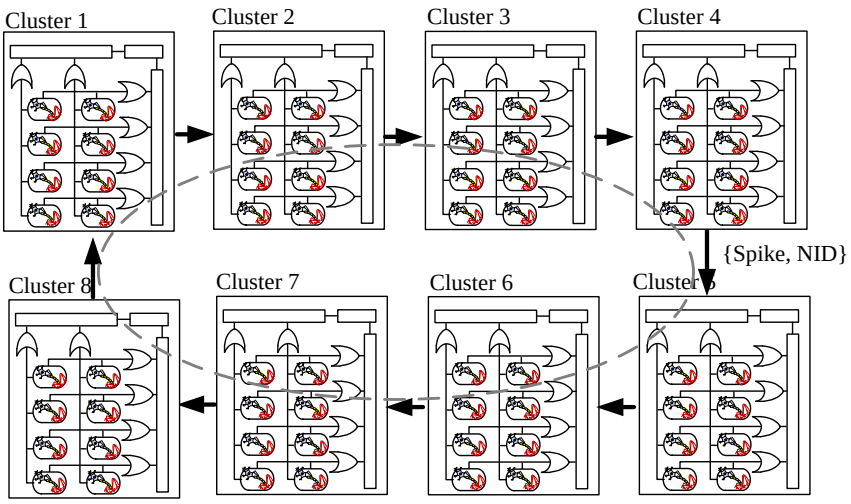


Figure 7.5: Inference module (IM) implemented in a 64-neuron spiking neural network.

ple spikes are detected in one clock cycle [32, 33]. The eight clusters are connected in an 8-stage systolic ring to propagate NIDs. To enhance the performance, neurons are pipelined to 2 stages. The neuron dynamics are tuned to achieve a high accuracy with less than 16% of neurons firing over a 2τ inference period (τ: neuron time constant, or ηn_s), saving the power.

The 64 RFs are quantized into a 14-bit resolution, so the size of RF memory is 224Kbits ($14 \times 64 \times (16 \times 16)$-bit). The 14-bit word length is the minimum needed for learning to converge properly. We use an unsupervised learning algorithm to train the 64 RFs. The learning algorithm is summarized in Table 4.3. In the inference phase, a simulation study shows that the 4 most significant bits (MSBs) are sufficient to provide high classification accuracy. To save power in inference, we divide the RF memory into core and auxiliary memory, and place them in two different supply rails. The core memory is implemented in two 32-Kbit register files to support high-memory bandwidth, and is powered on in inference and learning. The auxiliary memory is implemented in 160-Kbit SRAM, and it is powered on only in learning. The word length of feedback weights are optimized to a 4-bit resolution, and stored in eight

2-Kbit register files. Each 2-Kbit register file is placed next to an 8-neuron grid to provide feedback weights.

7.2.2 Sparse event-driven classifier

A task-driven classifier is integrated with the spiking LCA IM to recognize objects from ten object classes. Neuron outputs are connected to ten class nodes of the classifier. Each class node of the classifier represents one out of ten possible object classes. Each neuron-to-node connection has a weight. The weight quantifies the relation between the neuron's RF with the object class. A straight implementation of the classifier includes spiking rate calculators and multiply-and-accumulate units (MACs), as illustrated in Fig. 7.1. Each spiking rate calculator takes a neuron output to calculate the spiking rate of the neuron over an inference period, and $10N$ MACs calculate the weighted sum of spiking rate where N denotes the number of neurons in the IM. Each MAC output is the score of each object class. However, the straight implementation is expensive since the spiking rate calculator requires a division unit, and the MACs include costly multipliers.

For an efficient implementation of the classifier, the spiking rate calculators and the MACs are jointly designed, and they are implemented with only accumulators. The joint design is done in two phases. In the first phase, we remove the division unit in each spiking rate calculator. We notice that the spiking rate of a neuron is proportional to its spike count over an inference period, as formulated in (4.24). Therefore, a classifier using the weighted sum of spike count gives the same classification results as the conventional classifier design.

In the second phase of the classifier design, we leverage sparse binary (1 or 0) spikes to remove not only the multiplier in each MAC, but also the entire spiking rate calculator blocks. For an efficient implementation, we unroll the weighted sum of spike count. The multiplication of weight and spike count is equivalent to the

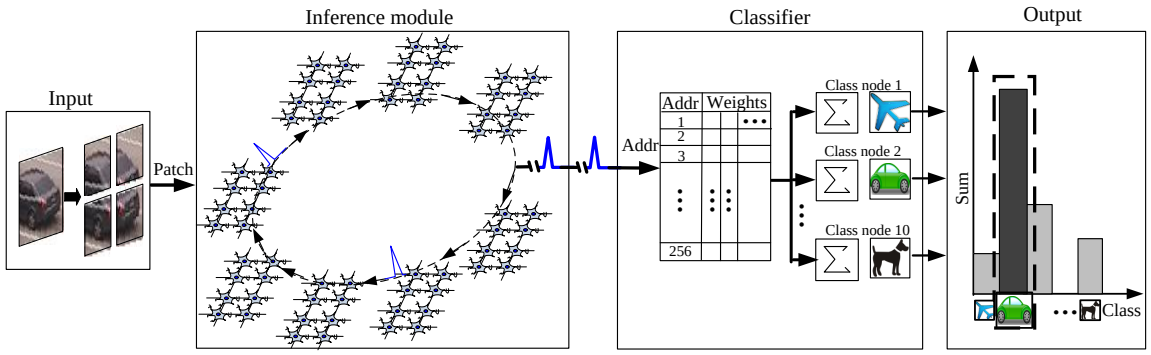


Figure 7.6: Spiking LCA IM and spike event-driven classifier.

accumulation of the weight whenever we receive spikes from the IM. Based on this observation, the weighted sum of spike count can be implemented with real-time accumulators where the weight accumulation is driven by spike events. Note that the number of accumulators required for the classifier design does not depend on the number of neurons in the IM network, but the number of object classes.

However, the transformation to the spike event-driven weight accumulation is complicated. Spikes generated by multiple neurons need to be serialized, which requires long latency to complete the weight accumulation. For an efficient implementation, the classifier is tightly integrated with the spiking LCA IM network. The classifier taps the systolic ring, and listens to the NIDs generated by the IM, as illustrated in Fig. 7.6. Since neuron spikes (or NIDs) are sparse, the event-driven classifier is idle most of the time. Furthermore, the classifier enables the weighted sum of spike count using only accumulators, so it saves 72% area and 65% power.

The 64-neuron IM network processes four overlapped image patches sequentially, and the dimension of the IM output is 256. Therefore, the classifier needs to store 256×10 weights to recognize 10 object classes. A supervised learning algorithm is used to train the weights [90]. Through simulation studies, the weights are quantized into 5-bit resolution, keeping classification errors less than 10%. The weights are consolidated, and stored in a 12.5-Kbit register file. A received NID from the IM

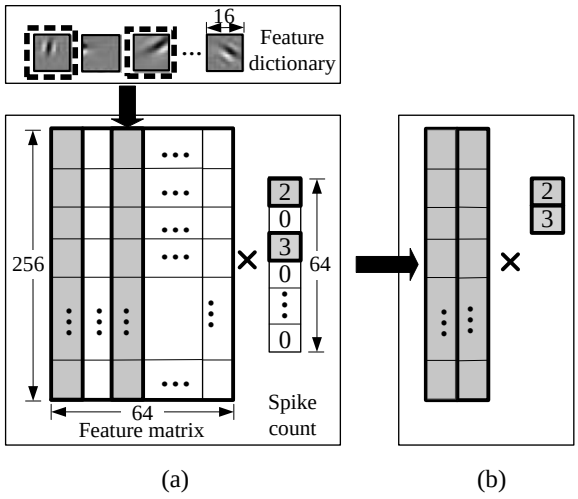


Figure 7.7: (a) Feature matrix and a 64-entry spike count vector multiplication to support learning. (b) Simplified vector-matrix product by taking advantage of sparsity.

is used to read the NID$^{\text{th}}$ word of the register file. The length of a word is 50-bit, concatenating ten 5-bit weights associated with ten class nodes.

7.2.3 Light-weight learning co-processor

Real-time learning is not necessary for practical applications, but on-chip learning reduces I/O power and it provides quick adaptation to changing environment. For this reason, a light-weight learning co-processor is integrated on chip to update the RFs and feedback weights. The RFs are developed iteratively following stochastic gradient descent formulated in (4.21) to minimize image encoding error and improve sparsity. Learning of the RFs is conveniently done on chip as all the information needed for learning including input pixels, neuron IDs, and spike rates are available on chip. The RF update triggers the update of feedback weights following $Q^TQ - I$.

Learning involves large vector and matrix multiplications that are naturally mapped to a vector processor. However, the vectors are sparse due to sparse neuron spikes, as shown in Fig. 7.7. We take advantage of this insight to design a scalar processor to cut over 84% of the workload and power. The low-cost scalar learning co-processor

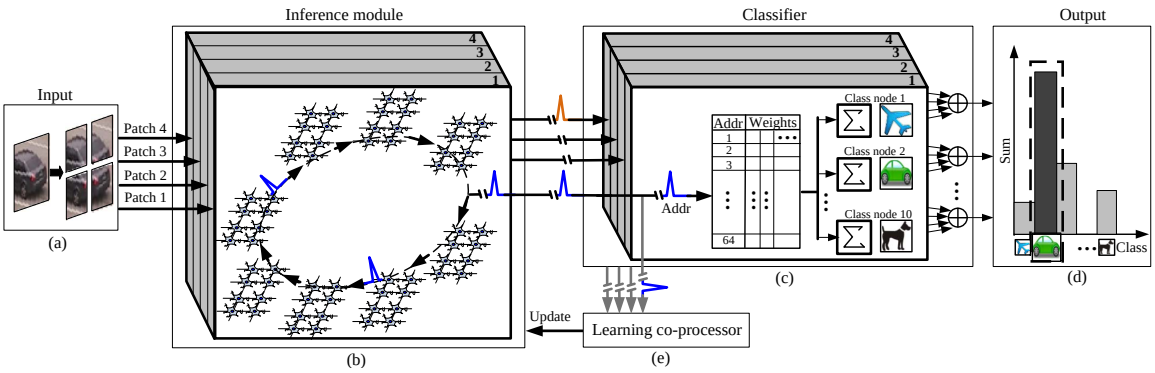


Figure 7.8: Object recognition processor with on-chip learning co-processor. (a) Four image patches. (b) Four 64-neuron spiking LCA IM networks. (c) Four event-driven sub-classifiers (d) Soft output of ten class nodes (e) On-chip learning co-processor.

provides three instructions to support learning: vector-matrix product, matrix scaling, and matrix-matrix product, which are all executed element-by-element in a serial fashion. The vector-matrix product and matrix scaling are used to compute the Q update, and the matrix-matrix product is used to compute the feedback weight update.

7.3 Performance enhancement

To enhance performance, we implement four independent 64-neuron IM networks, each of which operates on a 16×16 input image patch, as illustrated in Fig. 7.8. Since the four networks are identical, they share 64 RFs to extract features in input patches. By deploying the four neural networks in parallel, a high throughput and comparable inference accuracy can be achieved without the memory overhead associated with a large neural network.

The four IM networks extract features in four image patches in parallel, so the IM generates four sparse spike trains as its output. To enable a 4-way parallel weight accumulation, the event-driven classifier is divided into four sub-classifiers, each of which is integrated with an IM network. Furthermore, the 2.5-Kbit register file that

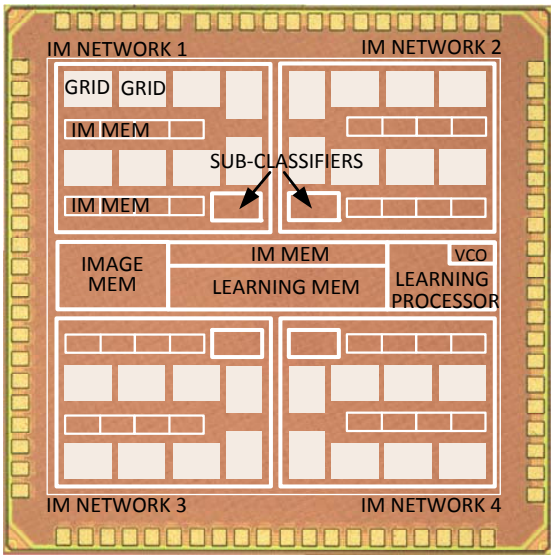


Figure 7.9: Chip microphotograph.

stores weights is divided into four memory blocks in order to increase the memory bandwidth by 4×. Each memory block is assigned to a sub-classifier. The scores calculated by the four sub-classifiers are summed to vote for the most likely object class, as illustrated in Fig. 7.8.

7.4 Measurement results

A test chip of the object recognition processor is fabricated in TSMC 65nm CMOS. Fig. 7.9 highlights sub-modules of the test chip. Four IM networks are placed in the corners of the chip, and each IM network includes 8 grids and 16-Kbit IM memory that store feedback weights. Each sub-classifier is integrated with an IM network for event-driven classification. An on-chip learning co-processor is implemented to update 64-Kbit IM memory and 160-Kbit learning memory centered at the chip, which store 64 RFs used by 4 IM networks. Also, the co-processor updates 64-Kbit IM memory distributed to four IM networks. We place IM memory and learning memory in different supply rails so that the learning memory is powered off to save power in inference. The weights of the classifier are updated by off-chip learning.

We collect IM outputs from a test chip, i.e., the IDs of the neurons that spike in response to a training image and their spike rates. The spike rates along with the training object label are used to train the classifier weights to minimize the MSE of the object label. Implementing an on-chip learning co-processor for the classifier remains our future work. Input images are scanned bit-by-bit into 100-Kbit image memory implemented in SRAM to accomplish the on-chip object recognition task and on-chip learning. Clock signals are generated by a VCO block.

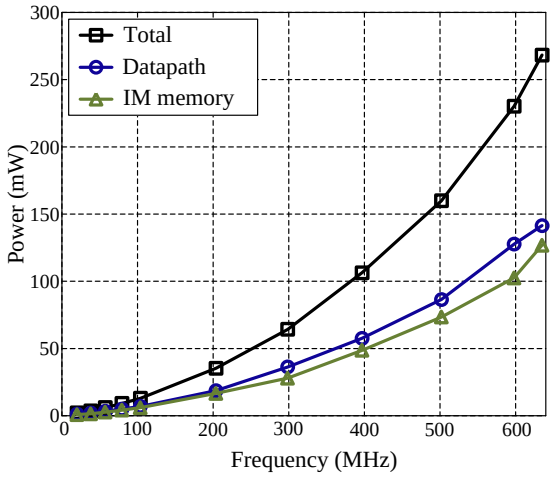


Figure 7.10: Measured power consumption of the object recognition processor.

We measure power consumption of the object recognition processor at room temperature. Through voltage scaling, datapath and IM memory power are measured at the minimum supply voltages for each clock frequency, as illustrated in Fig. 7.10. The object recognition processor runs at a maximum frequency of 635MHz at 1.0V to achieve a high throughput of 10.16G pixel/s, dissipating 268mW. The processor achieves the best energy efficiency at 40MHz with a throughput of 640M pixel/s, as summarized in Table I. The results demonstrate 8.2× higher throughput and 6.7× better energy efficiency than the previous SAILnet IM [32, 33]. The spiking LCA IM outperforms the SAILnet IM with high target firing rate, as discussed in Section 4.3.2.3.

An example of recognizing an object is shown in Fig. 7.8. Tested with the MNIST

TABLE I: CHIP SUMMARY

Core Area	1.35mm × 1.35mm (Datapath : 0.97mm^2, Memory : 0.48mm^2, Learning : 0.21mm^2, Periphery: 0.16mm^2)	
Chip Area	1.73 × 1.73mm (2.99mm^2)	
Frequency (MHz)	40	635
Datapath (V)	0.45	1.00
Memory (V)	0.425	1.00
Throughput (Mpixel/s)	640	10160
Power (mW)	3.65	268.2
Energy Efficiency (pJ/pixel)	5.70	26.40

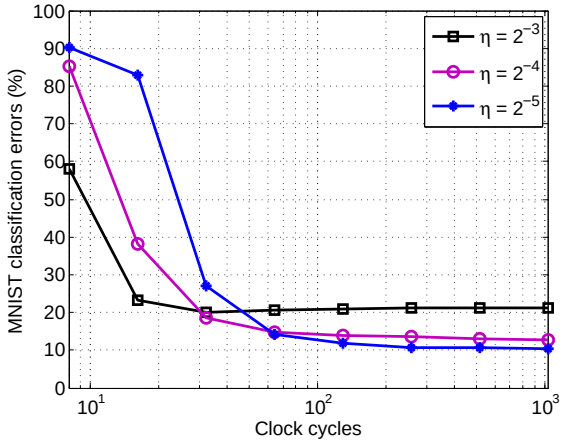
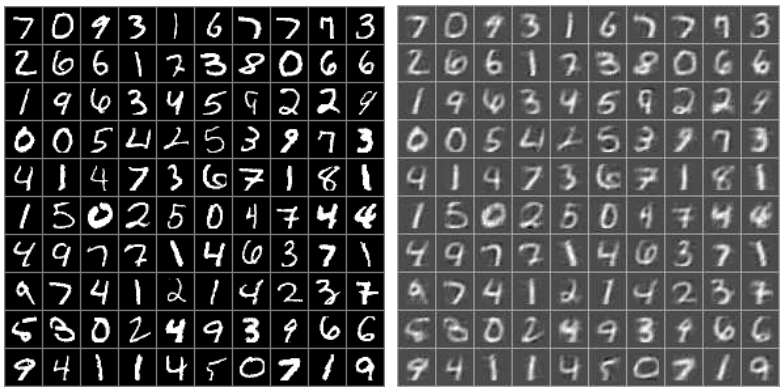
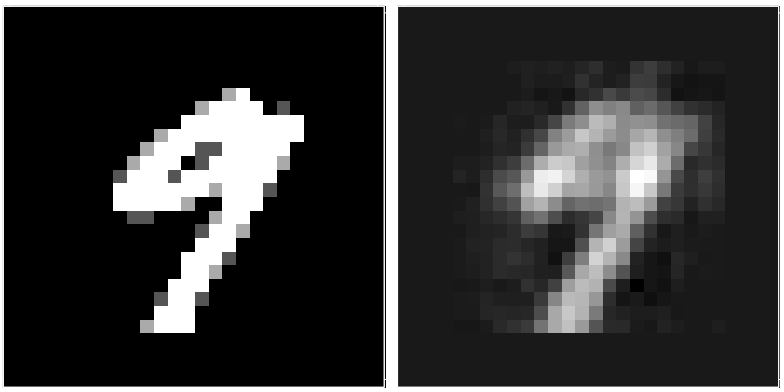


Figure 7.11: Classification error measured in different inference window.



(a) (b)

Figure 7.12: (a) 100 input images (each square in the grid is a 28×28 image), and (b) the reconstructed images using chip measurements.



(a) (b)

Figure 7.13: Misclassification in digit recognition. (a) Input image, '9', and (b) the reconstructed image that is classified as '7'.

database of 28×28 handwritten digits [92], the chip is capable of recognizing 9.9M objects/s at a classification accuracy of 84%. Increasing the inference period from 2τ to 12τ enhances the classification accuracy to 90% as illustrated in Fig. 7.11, but cuts the throughput by $6\times$. Fig. 7.12 shows 100 input images randomly selected from the MNIST dataset and the reconstructed images. Fig. 7.13 shows an example of misclassification in digit recognition. The classification accuracy of this single-layer IM and single-layer classifier is still lower than what is reported in state-of-the-art machine learning literature, but the scalable architecture allows multiple layers of IM and classifier to be integrated in future work to improve the results.

The on-chip learning co-processor runs at a maximum frequency of 650MHz at 1.0V, dissipating 258mW. The power breakdown of learning is shown in Fig. 7.14. We measured the power consumption of datapath, IM memory, and learning memory at the minimum supply voltages at each clock frequency. Since the co-processor needs to execute three instructions to update RFs and feedback weights, we set the supply voltages and frequency where the three instructions are fully functional, and report the average power consumption of the three for each clock frequency. To compute vector-matrix product, matrix scaling, and matrix-matrix product, 3057, 6144, and

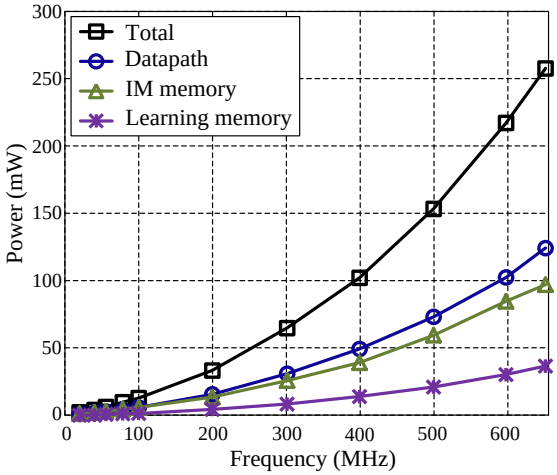


Figure 7.14: Measured power consumption of learning co-processor.

22464 clock cycles per 16×16 image patch are required, respectively. A rigorous training using 4M image patches can be completed within 200 seconds. Note that the learning rule (4.21) is a global computation, so it takes longer than the SAILnet local learning. After learning converges, the co-processor is powered off.

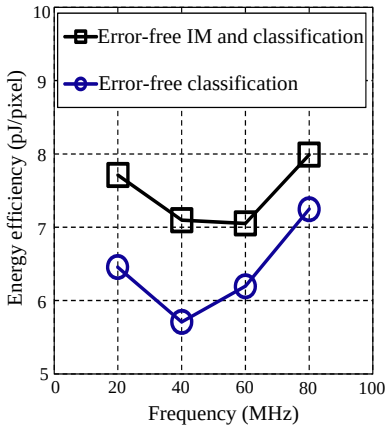


Figure 7.15: Measured energy efficiency of the object recognition processor by exploiting error tolerance.

The neuromorphic IM is error tolerant [33], and integrating IM and classifier provides additional error tolerance as the soft classifier accommodates more errors in feature extraction. The classifier calculates the score of each object class, chooses the maximum score, and generates the corresponding object ID as output. Errors

propagated from the IM can be tolerated in the classifier. Error-free classification even with errors in the output of the IM can be achieved. With the error tolerance, the supply voltage for the datapath and the memory can be reduced to 450mW and 425mW, respectively to improve the energy efficiency to 5.7pJ/pixel at 40MHz, as shown in Fig. 7.15.

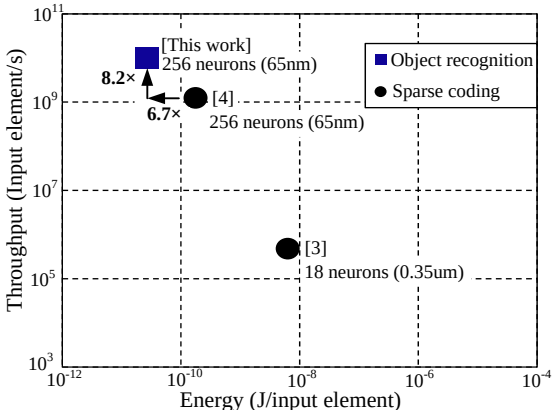


Figure 7.16: Throughput and energy comparison with state-of-the-art neuromorphic ASICs for sparse coding.

Table 7.1: Comparison with prior works

Reference	Seo [43]	Merolla [42]	Shapero [53]	Kim [32]	This work
# Neurons	256	256	18	256	256
# Synapses	64K	256K	0.53K	128K	83K
Bitwidth of a Synapse	4 bit	1 bits	7 bit	8 and 13 bits	4, 5, and 14 bits
Memory size	256Kbits	256Kbits	3.7Kbits	1.31Mbits	301Kbits
Architecture	Crossbar	Crossbar	Crossbar	2-layer bus and ring	2-layer bus and ring
Algorithm	STDP	RBM	Spiking LCA	SAILnet	Spiking LCA with classification
Learning	On-chip	Off-chip	Off-chip	On-chip	On-chip
Technology	45nm	45nm	0.35um	65nm	65nm
Core Area	4.2mm^2	4.2mm^2	-	3.1mm^2	1.8mm^2
Energy metric	-	45pJ/spike	6.3nJ/input	48pJ/pixel	5.7pJ/pixel

7.5 Summary

This chapter presents a 65nm CMOS end-to-end neuromorphic object recognition processor. The size of on-chip memory is vastly reduced by implementing the

convolutional spiking LCA IM that limits the size of receptive fields. The classifier is tightly integrated with the IM, and snoops the ring of the IM network to reduce area and power. The light-weight learning co-processor executes three instructions to update RFs and feedback weights, and its power is reduced by processing non-zero neuron output. To enhance the throughput of the object recognition processor, we organize 256 neurons into four IM networks to unroll the convolution operation.

Compared to state-of-the-art neuromorphic ASICs for sparse coding [53, 32], this design demonstrates new capabilities including object recognition. Using algorithm and architecture techniques, the energy efficiency of the neuromorphic ASIC chip is enhanced to 5.7pJ/pixel, which is an 8.2× improvement over the previously designed SAILnet sparse coding IM and a 1105× improvement over spiking LCA IM implemented in 0.35um CMOS, as illustrated in Fig. 7.16. In comparison with other neuromorphic ASICs [43, 42], this work implements a 2-layer bus and ring architecture in the IM, and demonstrates object recognition supported by on-chip learning. The test chip achieves a classification accuracy of 90%. Table 7.1 summarizes the comparison.

CHAPTER VIII

Conclusion

The co-design techniques are employed to advance the practical implementations of CT image reconstruction, neuromorphic sparse coding, and object recognition. We demonstrate a CT forward-projection architecture for CT image reconstruction to achieve both acceleration and scalability. We also demonstrate a 2-layer bus-ring architecture for the efficient mapping of sparse spiking neural networks to achieve 1.24 G pixel/s feature extraction throughput. This work leads to an event-driven classifier design integrated with a spiking LCA IM to demonstrate 640 M pixel/s and 3.67mW object recognition.

8.1 Advances

Custom forward-projection architecture is implemented for fast iterative image reconstruction in X-ray CT. In comparison with conventional simulators for medical imaging, fixed-point quantization is applied in the implementation to reduce hardware costs. Errors caused by fixed-point quantization are tolerated in the iterative method, and the degradation of the quality of practical CT images was negligible, motivating the use of fixed-point datapath in implementation. The proposed water-filling buffer resolves the inherent hardware inefficiency due to a mismatch between the 3D object grid and the 2D projection grid. The proposed out-of-order sectored

processing takes advantage of temporal and spatial locality of references to reduce off-chip memory bandwidth by three orders of magnitude, providing scalability for high-level parallelism in advanced hardware platforms.

The impacts of fixed-point quantization are further analyzed in the iterative method using a perturbation-based model. The effect of fixed-point quantization is modeled as a perturbation of floating-point arithmetic by injecting uniform white noise. Particularly, the effects of the fixed-point quantization error in forward-projection, back-projection, and image update are quantified using the open loop and the closed loop gain of a diagonally preconditioned gradient descent algorithm with a quadratic regularizer. An upper bound on the quantization error variance is derived, and the result shows that the quantization step size can be chosen to meet a given upper bound.

The co-design techniques are employed to implement custom hardware architectures for neuromorphic sparse coding. A tuning strategy of the neural network size, firing rate and update step size is proposed to achieve sparse and random spiking. The resulted sparsity allows for implementation of spiking neural networks by addressing routing complexity and scalability. The proposed arbitration-free bus tolerates spike collision by leveraging sparsity, removing costly bus arbitor and enabling efficient communication. A scalable latent ring mitigates neuron misfires by damping neuron spiking to be sparse and random. Ultimately, a 2-layer bus-ring architecture is proposed to achieve both high throughput and scalability. In this 2-layer architecture, neurons are grouped into multiple clusters, and neurons in a cluster are connected with a 2D bus to reduce wire loading. Multiple 2D buses are linked in a systolic ring to reduce communication latency. The cluster size and the ring size are optimized by considering the tradeoff between spike collision rate and throughput.

The optimized bus-ring architecture allows for the efficient mapping of sparse spiking networks onto hardware using a 65nm CMOS technology, achieving a sparse

feature extraction throughput of 1.24G pixel/s at 310MHz and 1.0V. The proposed memory power gating improves energy efficiency of inference by separating weight memory into core and auxiliary memory, and powering off the auxiliary memory in inference. The proposed approximate learning scheme tracks only significant neuron activities, thereby enabling on-chip learning in seconds. The sparse coding ASIC exploits error tolerance in sparse feature extraction, lowering the core memory supply voltage for an enhanced energy efficiency.

The inherent sparse spiking enables low-power energy-efficient event-driven classification, demonstrating 640M pixel/s 3.65mW object recognition using a 65nm CMOS technology. The convolutional spiking LCA inference module (IM) limits the size of receptive fields by dividing an input image into patches, saving area and power. The proposed classifier is tightly integrated with the IM, and activated by sparse spike-events, simplifying its implementation by removing all multiplications. Multiple IM networks and sub-classifiers boost the throughput up to 10.16G pixel/s at 635MHz by processing multiple image patches in parallel. The light-weight learning co-processor leverages sparsity to efficiently implement the stochastic gradient descent updates for on-line learning. The integrated IM with classifier enhances throughput and energy efficiency by exploiting error tolerance.

8.2 Future work

Implementation of custom neural network architectures through co-design techniques enables the development of unconventional neuromorphic computing hardware. Analog designs are efficient in computing, but component mismatch requires calibration, and scaling to deep submicron CMOS devices further increases mismatch in analog designs [94]. Therefore, analog computing may not be the perfect solutions for a conventional von Neumann computing. However, it could be ideal candidates for building neuromorphic computing, since neuromorphic computing is tolerant to

errors caused by noise in the signals and variations of the components [95, 96, 97].

Based on this observation, energy-efficient neuromorphic systems can be realized by the co-design of analog components and error-tolerant spiking neural networks. Future plans about an implementation of analog-digital neural networks include 1) integration of analog CMOS neurons with digital neural network architectures to demonstrate hybrid neuromorphic computing platforms, 2) investigation of content addressable memory enabled by local connectivity in sparse spiking neural networks in order to reduce cost and power, and 3) analysis of the impact of analog neurons and local memory on neuromorphic algorithms through the co-design techniques.

In addition, the intuition of error tolerance in the end-to-end object recognition system allows for the investigation of low-cost medical imaging hardware through the co-design techniques. Despite the opportunities of speeding up image processing using general-purpose hardware, the ultimate goal of medical imaging is to help doctors with disease diagnosis [98]. Study on dictionary learning shows that an exact signal reconstruction is not necessarily required for a good detection/prediction performance [90]. Motivated by these observations, efficient medical imaging systems can be investigated by the following steps: 1) non-perfect image reconstruction and image downsampling, 2) feature extraction in the region of interest of the low quality images, and 3) analysis of the impacts of the image quality on the classification results.

Furthermore, the FPGA-based CT forward projection accelerator paves a way to implement a complete CT image reconstruction system by the following steps: 1) implementation of an FPGA-based CT back-projection accelerator, 2) implementation of an edge preserving regularizer to improve the quality of reconstructed images, 3) integration of the forward-projection, the back-projection, and the regularizer designs, and 4) enhancement of the throughput and the accuracy of image reconstruction using multiple FPGAs.

APPENDIX

APPENDIX A

Quantization error bounds of iterative image reconstruction in X-ray CT

A.1 Derivation of perturbation-based error bounds

(Derivation of (3.7))

$$
\begin{aligned}
\rho\left(\mathrm{cov}\left(e^{(n)}, e^{(n)}\right)\right) &= \max_{x:\|x\|=1}\left(x' cov\left(e^{(n)}, e^{(n)}\right) x\right) \\
&= \frac{\Delta_{\mathrm{fp}}^2}{12} \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} x'\left(D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} \left(VFV'\right) D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}}\right) x\right) \\
&= \frac{\Delta_{\mathrm{fp}}^2}{12} \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} \parallel F^{\frac{1}{2}} V' D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} x \parallel^2\right) \\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12} \rho(F) \cdot \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} \parallel V D^{\frac{1}{2}} U \Sigma^k U D^{\frac{1}{2}} x \parallel^2\right) \\
&= \frac{\Delta_{\mathrm{fp}}^2}{12} \rho(F) \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} \parallel D^{\frac{1}{2}} U \Sigma^k U' D^{\frac{1}{2}} x \parallel^2\right) \\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12} \rho(F)\rho(D) \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} \parallel U \Sigma^k U' D^{\frac{1}{2}} x \parallel^2\right) \\
&= \frac{\Delta_{\mathrm{fp}}^2}{12} \rho(F)\rho(D) \max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1} \parallel \Sigma^k U' D^{\frac{1}{2}} x \parallel^2\right)
\end{aligned}
$$

$$
\begin{aligned}
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\rho(F)\rho(D)\max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1}\rho(\Sigma)^{2k}\parallel U'D^{\frac{1}{2}}x\parallel^2\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\rho(F)\rho(D)\max_{x:\|x\|=1}\left(\sum_{k=0}^{n-1}\rho(\Sigma)^{2k}\parallel D^{\frac{1}{2}}x\parallel^2\right)\\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\rho(F)\rho(D)\rho(D)\sum_{k=0}^{n-1}\rho(\Sigma)^{2k}\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\rho(F)\rho(D^2)\frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2}. \qquad \text{(A.1)}
\end{aligned}
$$

The second equality in (A.1) holds using the definition of matrix 2-norm. The first, second, third, and forth inequalities hold from the matrix norm property that $\parallel AB \parallel \leq \parallel A \parallel \cdot \parallel B \parallel$. The third, forth, and fifth equalities hold since a unitary matrix conserves a matrix norm.

(Derivation of (3.8))

$$
\begin{aligned}
\mathrm{tr}\left(\mathrm{cov}\left(e^{(n)},e^{(n)}\right)\right) &= \frac{\Delta_{\mathrm{fp}}^2}{12}\mathrm{tr}\left(\sum_{k=0}^{n-1}\left(D^{\frac{1}{2}}U\Sigma^kU'D^{\frac{1}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\Sigma^kU'D^{\frac{1}{2}}\right)\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\mathrm{tr}\left(D^{\frac{1}{2}}U\Sigma^kU'D^{\frac{1}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\Sigma^kU'D^{\frac{1}{2}}\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\mathrm{tr}\left(\Sigma^kU'D^{\frac{1}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\Sigma^kU'DU\right)\\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^k)\mathrm{tr}\left(U'D^{\frac{1}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\Sigma^kU'DU\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^k)\mathrm{tr}\left(U'D^{\frac{3}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\Sigma^k\right)\\
&\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^{2k})\mathrm{tr}\left(U'D^{\frac{3}{2}}\left(VFV'\right)D^{\frac{1}{2}}U\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\sum_{k=0}^{n-1}\rho\left(\Sigma^{2k}\right)\mathrm{tr}\left(D\left(VFV'\right)D\right)\\
&= \frac{\Delta_{\mathrm{fp}}^2}{12}\mathrm{tr}\left(D\left(VFV'\right)D\right)\frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2}. \qquad \text{(A.2)}
\end{aligned}
$$

The second equality in (A.2) holds from the property that $\mathrm{tr}(A+B) = \mathrm{tr}(A)+\mathrm{tr}(B)$. The third, fourth, and fifth equalities hold from the property that $\mathrm{tr}(ABC) = \mathrm{tr}(BCA)$. The first and second inequalities hold from the property that $\mathrm{tr}(AD) \leq max_i(d_i) \cdot \mathrm{tr}(A)$ since D has positive diagonal entries.

(Derivation of (3.11)) Continued from (3.10), using $K_{\mathrm{bp}} = D$ and (3.5), the upper bound on the spectral radius of the second term of (3.10) is

$$\begin{aligned}
&\frac{\Delta_{\mathrm{bp}}^2}{12} \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} x' \left(D^{\frac{1}{2}} U \Sigma^k U' D^2 U \Sigma^k U' D^{\frac{1}{2}} \right) x \right) \\
&\leq \frac{\Delta_{\mathrm{bp}}^2}{12} \rho(D) \cdot \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} \parallel U \Sigma^k U' D^{\frac{1}{2}} x \parallel^2 \right) \\
&\leq \frac{\Delta_{\mathrm{bp}}^2}{12} \rho(D) \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} \rho(\Sigma)^{2k} \parallel U' D^{\frac{1}{2}} x \parallel^2 \right) \\
&\leq \frac{\Delta_{\mathrm{bp}}^2}{12} \rho(D) \rho(D) \sum_{k=0}^{n-1} \rho(\Sigma)^{2k} \\
&= \frac{\Delta_{\mathrm{bp}}^2}{12} \rho(D^2) \frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2},
\end{aligned} \tag{A.3}$$

and the upper bound of the third term of (3.10) is

$$\begin{aligned}
&\frac{\Delta_{\mathrm{im}}^2}{12} \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} x' \left(D^{\frac{1}{2}} U \Sigma^{k+1} U' D^{-1} U \Sigma^{k+1} U' D^{\frac{1}{2}} \right) x \right) \\
&\leq \frac{\Delta_{\mathrm{im}}^2}{12} \rho(D) \cdot \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} \parallel U \Sigma^{k+1} U' D^{-\frac{1}{2}} x \parallel^2 \right) \\
&\leq \frac{\Delta_{\mathrm{im}}^2}{12} \rho(D) \max_{x:\|x\|=1} \left(\sum_{k=0}^{n-1} \rho(\Sigma)^{2k+2} \parallel U' D^{-\frac{1}{2}} x \parallel^2 \right) \\
&\leq \frac{\Delta_{\mathrm{im}}^2}{12} \rho(D) \rho(D^{-1}) \sum_{k=0}^{n-1} \rho(\Sigma)^{2k+2} \\
&= \frac{\Delta_{\mathrm{im}}^2}{12} \rho(D) \rho(D^{-1}) \rho((\Sigma)^2) \frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2}.
\end{aligned} \tag{A.4}$$

Using (A.1), (A.3), and (A.4), we have

$$\rho\left(\mathrm{cov}\left(e^{(n)},e^{(n)}\right)\right)$$
$$\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\rho(F)\rho(D^2)\frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2}+\frac{\Delta_{\mathrm{bp}}^2}{12}\rho(D^2)\frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2}+\frac{\Delta_{\mathrm{im}}^2}{12}\rho(D)\rho(D^{-1})\rho((\Sigma)^2)\frac{1-\rho(\Sigma)^{2n}}{1-\rho(\Sigma)^2}. \tag{A.5}$$

Therefore, as $n\to\infty$, the upper bound on the spectral radius of (3.10) is

$$\rho(\mathrm{cov}(e^{(\infty)},e^{(\infty)}))\leq \frac{\Delta_{\mathrm{fp}}^2}{12}\frac{\rho(D^2)\rho(F)}{1-\rho(\Sigma^2)}+\frac{\Delta_{\mathrm{bp}}^2}{12}\frac{\rho(D^2)}{1-\rho(\Sigma^2)}+\frac{\Delta_{\mathrm{im}}^2}{12}\frac{\rho(\Sigma^2)\rho(D)\rho(D^{-1})}{1-\rho(\Sigma^2)}. \tag{A.6}$$

(Derivation of (3.12)) Continued from (3.10), using $K_{\mathrm{bp}}=D$ and (3.5), the upper bound on the trace of the second term of (3.10) is

$$\begin{aligned}
&\frac{\Delta_{\mathrm{bp}}^2}{12}\mathrm{tr}\left(\sum_{k=0}^{n-1}\left(D^{\frac{1}{2}}U\Sigma^kU'DU\Sigma^kU'D^{\frac{1}{2}}\right)\right)\\
&\leq \frac{\Delta_{\mathrm{bp}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^k)\mathrm{tr}\left(U'DU\Sigma^kU'DU\right)\\
&\leq \frac{\Delta_{\mathrm{bp}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^{2k})\mathrm{tr}\left(U'D^2U\right)\\
&= \frac{\Delta_{\mathrm{bp}}^2}{12}\mathrm{tr}\left(D^2\right)\frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2},
\end{aligned} \tag{A.7}$$

and the upper bound of the trace of the third term of (3.10) is

$$\begin{aligned}
&\frac{\Delta_{\mathrm{im}}^2}{12}\mathrm{tr}\left(\sum_{k=0}^{n-1}\left(D^{\frac{1}{2}}U\Sigma^{k+1}U'D^{-1}U\Sigma^{k+1}U'D^{\frac{1}{2}}\right)\right)\\
&\leq \frac{\Delta_{\mathrm{im}}^2}{12}\sum_{k=0}^{n-1}\rho(\Sigma^{k+1})\mathrm{tr}\left(U'DU\Sigma^{k+1}U'D^{-1}U\right)
\end{aligned}$$

$$\leq \frac{\Delta_{\text{im}}^2}{12} \sum_{k=0}^{n-1} \rho(\Sigma^{2k+2}) \text{tr}\,(I)$$
$$= \frac{\Delta_{\text{im}}^2}{12} \rho(\Sigma)^2 \text{tr}\,(I)\, \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2}, \tag{A.8}$$

Using (A.2), (A.7), and (A.8), we have

$$\text{tr}\left(\text{cov}\left(e^{(n)}, e^{(n)}\right)\right)$$
$$\leq \frac{\Delta_{\text{fp}}^2}{12} \text{tr}\,(D\,(VFV')\,D)\, \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2} + \frac{\Delta_{\text{bp}}^2}{12} \text{tr}\left(D^2\right) \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2} + \frac{\Delta_{\text{im}}^2}{12} \rho(\Sigma)^2 \text{tr}\,(I)\, \frac{1-\rho(\Sigma^2)^n}{1-\rho(\Sigma)^2}. \tag{A.9}$$

Therefore, as $n \to \infty$, the upper bound on the trace of (3.10) is

$$\text{tr}(\text{cov}(e^{(\infty)}, e^{(\infty)})) \leq \frac{\Delta_{\text{fp}}^2}{12} \frac{\text{tr}(VFV'D^2)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{bp}}^2}{12} \frac{\text{tr}(D^2)}{1-\rho(\Sigma^2)} + \frac{\Delta_{\text{im}}^2}{12} \frac{\rho(\Sigma^2)\text{tr}(I)}{1-\rho(\Sigma^2)}. \tag{A.10}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. A. Fessler, "Statistical image reconstruction methods for transmission tomography," *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis*, pp. 1–70, 2000.

[2] T. M. Buzug, *Computed tomography: from photon statistics to modern cone-beam CT.* New York: Springer-Verlag, 2009.

[3] S. Kawata and O. Nalcioglu, "Constrained iterative reconstruction by the Conjugate Gradient method," *IEEE Trans. Med. Imag.*, vol. 4, pp. 65–71, 1985.

[4] Z. Q. Luo and P. Tseng, "On the convergence of the coordinate descent method for convex differentiable minimization," *J. Optim. Theory Appl.*, vol. 72, no. 1, pp. 7–35, 1992.

[5] H. Erdogen and J. A. Fessler, "Ordered subsets algorithms for transmission tomography," *Phys. Med. Biol.*, vol. 44, pp. 2835–51, 1999.

[6] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic Reconance in Medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[7] K. T. Block, M. Uecker, and J. Frahm, "Undersampled radial MRI with multiple coils. Iterative image reconstruction using a total variation constant," *Magnetic Reconance in Medicine*, vol. 57, no. 6, pp. 1086–1098, 2007.

[8] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger, "SENSE: sensitivity encoding for fast MRI," *Magnetic Reconance in Medicine*, vol. 42, pp. 952–962, 1999.

[9] D. J. Field, "What is the goal of sensory coding?" *Neural Computation*, vol. 6, no. 4, pp. 559–601, 1994.

[10] B. A. Olshausen, "Principles of image representation in in visual cortex," *The Visual Neurosciences*, pp. 1603–1615, 2003.

[11] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, pp. 607–609, 1996.

[12] P. Foldiak, "Forming sparse representations by local anti-Hebbian learning," *Biological Cybernetics*, vol. 64, no. 2, pp. 165–170, Dec. 1990.

[13] M. S. Falconbridge, R. L. Stamps, and D. R. Badcock, "A simple Hebbian/anti-Hebbian network learns the sparse, independent components of natural images," *Neural Computuation*, vol. 18, no. 2, pp. 415–429, Feb. 2006.

[14] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief neural networks for scalable unsupervised learning of hierarchical representation," in *26th Annual Int. Conf. Mahine Learning*, 2009, pp. 609–616.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1–9.

[16] G. Hinton, Y. Dong, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senio, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[17] G. E. Dahl, Y. Dong, L. Deng, and A. Acero, "Contex-depedent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, and Language Proc.*, vol. 20, no. 1, pp. 30–42, 2011.

[18] J. Kim, J. A. Fessler, and Z. Zhang, "Forward-projection architecture for fast iterative image reconstruction in X-ray CT," *IEEE Trans. Signal Processing*, vol. 60, no. 10, pp. 5508–5518, 2012.

[19] J. Kim, Z. Zhang, and J. A. Fessler, "Hardware acceleration of iterative image reconstruction for X-ray computed tomography," in *IEEE Conf. Acoust. Speech Sig. Proc.*, May 2011, pp. 1697–1700.

[20] J. Kim, J. A. Fessler, and Z. Zhang, "Perturbation-based error analysis of iterative image reconstruction algorithm for X-ray computed tomography," in *Int. Conf. Image Formation in X-ray Computed Tomography*, 2012, pp. 194–197.

[21] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Phy. Med. Biol.*, vol. 52, pp. 3405–19, 2007.

[22] ——, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 3, pp. 654–63, 2005.

[23] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[24] I. Goddard and M. Trepanier, "High-speed cone-beam reconstruction: an embedded systems approach," in *SPIE*, vol. 4681, Feb. 2002, pp. 483–91.

[25] J. Xu, N. Subramanian, A. Alessio, and S. Hauck, "Impulse C vs. VHDL for Accelerating Tomographic Reconstruction," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 171–174.

[26] F. Attneave, "Some informational aspects of visual perception," *Psychological Review*, vol. 61, no. 3, pp. 183–193, May 1954.

[27] J. J. Atick and A. N. Redlich, "What does the retina know about natural scenes?" *Neural Computation*, vol. 4, no. 2, pp. 196–210, Mar. 1992.

[28] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *J. Physiol.*, vol. 195, no. 1, pp. 215–243, 1968.

[29] R. L. D. Valois, D. G. Albrecht, and L. G. Thorell, "Spatial frequency selectivity of cells in macaque visual cortex," *Vision Research*, vol. 22, no. 5, pp. 545–559, 1982.

[30] J. P. Jones and L. A. Palmer, "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex," *J. Physiol.*, vol. 58, no. 6, pp. 1233–1258, 1987.

[31] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "Efficient Hardware Architecture for Sparse Coding," *IEEE Trans. Signal Processing*, vol. 62, no. 16, pp. 4173–4186, 2014.

[32] ——, "A 6.67mW sparse coding ASIC enabling on-chip learning and inference," in *Symp. VLSI Circuits*, 2014, pp. 61–62.

[33] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, 2015.

[34] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640m pixel/s xmW neuromorphic object recognition processor with on-chip learning," in *Symp. VLSI Circuits*, 2015.

[35] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 floating gate synapses," in *Int. Joint Conf. Neural Networks*, 1989, pp. 191–196.

[36] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, M. Yamada, and A. Masaki, "Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed on 576 digital neurons," in *Int. Joint Conf. Neural Networks*, 1990, pp. 527–535.

[37] U. Ramacher, "SYNAPSE–A neurocomputer that synthesizes neural algorithms on a parallel systolic engine," *J. Parallel and Distributed Computing*, vol. 14, no. 3, pp. 306–318, 1992.

[38] P. Ienne and M. A. Viredaz, "GENES IV: A bit-serial processing element for a multi-model neural-network accelerator," *J. VLSI Signal Process.*, vol. 9, no. 3, pp. 257–273, 1995.

[39] M. Mahowald, *An analog VLSI system for stereoscopic vision.* Springer, 1994.

[40] M. A. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," Ph.D. dissertation, California Inst. Techology, 1991.

[41] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses," *IEEE Trans. Neural Networks*, vol. 18, no. 1, pp. 253–265, Jan. 2007.

[42] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *IEEE Custom Integrated Circuits Conf.*, 2011, pp. 1–4.

[43] J. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Fridman, "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *IEEE Custom Integrated Circuits Conf.*, 2011, pp. 1–4.

[44] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Trans. Neural Netowrks*, vol. 17, no. 1, pp. 211–221, 2006.

[45] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *Artificial Neural Networks and Machine Learning–ICANN*, 2012, pp. 121–128.

[46] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, L. Camunas-Mesa, R. Berner, M. Rivas-Perez, T. Delbruck, S. C. Liu, R. Douglas, P. Hafliger, G. Jimenez-Moreno, A. C. Ballcels, T. Serrano-Gotarredona, A. J. Acosta-Jimenez, and B. Linares-Barranco, "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory–processing–learning–actuating system for high-speed visual object recognition and traking," *IEEE Trans. Neural Networks*, vol. 20, no. 9, pp. 1417–1438, 2009.

[47] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "SpiNNaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1–11, 2013.

[48] C. Zamarreno-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multicasting Mesh AER: A Scalable Assembly Approach for Reconfigurable Neuromorphic Structured AER Systems. Application to ConvNets," *IEEE Trans. Biomed. Circuits and Syst.*, vol. 7, no. 1, pp. 82–102, 2013.

[49] K. Kim, S. Lee, J.-Y. Kim, M. Kim, and H.-J. Yoo, "A 125 GOPS 583 mW network-on-chip based parallel processor with bio-inspired visual attention engin," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 136–147, 2009.

[50] S. Lee, J. Oh, J. Park, J. Kwon, M. Kim, and H.-J. Yoo, "A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 42–51, 2011.

[51] J.-Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, "A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, 2010.

[52] J. Oh, G. Kim, B.-G. Nam, and H.-J. Yoo, "A 57 mW 12.5 uJ/epoch embedded mixed-mode neuro-fuzzy processor for mobile real-time object recognition," *IEEE J. Solid-State Circuits*, vol. 48, no. 11, pp. 2894–2907, 2013.

[53] S. Shapero, C. Rozell, and P. Hasler, "Configurable hardware integrate and fire neurons for sparse approximation," *Neural Networks*, vol. 45, pp. 134–143, 2013.

[54] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with Tera-parallel MIMD architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf.*, 2015, pp. 1–3.

[55] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci.*, vol. 3, pp. 919–926, 2000.

[56] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[57] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *J. of Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[58] W. B. Levy and R. A. Baxter, "Energy efficient neural codes," *Neural Computation*, vol. 8, no. 3, pp. 531–543, 1996.

[59] E. Seeram, *Computed tomography: Physical principles, clinical applications, and quality control.* Saunders Elsverier, 2009.

[60] J. H. Siewerdsen and D. A. Jaffray, "Cone-beam computed tomography with a flat-panel imager: Effects of image lag," *Med. Phys.*, vol. 26, pp. 1624–41, 1999.

[61] D. A. Jaffray, J. H. Siewerdsen, J. W. Wong, and A. A. Martinez, "Flat-panel cone-beam computed tomography for image-guided radiation therapy," *Int. J. Radiat. Oncol. Biiol. Phys.*, vol. 53, pp. 1337–49, 2002.

[62] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *J. Opt. Soc. Am. A*, vol. 1, no. 6, pp. 612–619, 1984.

[63] J. A. Fessler, *Image reconstruction: Algorithms and analysis.* Book in preparation.

[64] K. Sauer and C. Bouman, "A local update strategy for iterative reconstruction from projections," *IEEE Trans. Signal Processing*, vol. 41, no. 2, pp. 534–548, 1993.

[65] J.-B. Thibault, K. D. Sauer, C. A. Bouman, and J. Hsieh, "A three-dimensional statistical approach to improved image quality for multislice helical CT," *Med. Phys.*, vol. 34, pp. 4526–44, 2007.

[66] Y. Long, J. A. Fessler, and J. M. Balter, "3D forward and back-projection for X-ray CT using separable footprints," *IEEE Trans. Med. Imag.*, vol. 29, no. 11, pp. 1839–50, 2010.

[67] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time signal processing.* New Jersey: Prentice Hall, 1997.

[68] J. A. Gubner, *Probability and random processes for electrical and computer engineers.* Cambridge University Press, 2006.

[69] D. C. Lay, *Linear algebra and its applications.* Pearson Education, Inc, 2003.

[70] K. Hoffman and R. Kunze, *Linear Algebra.* New Jersey: Prentice Hall, 2003.

[71] L. N. Trefethen and D. Bau, *Numerical linear algebra.* SIAM, 1997.

[72] Virtex-5 FPGA family. Xilinx Corporation. [Online]. Available: http://www.xilinx.com/products/virtex5/index.htm

[73] Virtex-7 FPGA family. Xilinx Corporation. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm

[74] M. Rehn and F. T. Sommer, "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields," *J. Computational Neuroscience*, vol. 22, no. 2, pp. 135–146, Apr. 2007.

[75] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, no. 10, pp. 2526–2563, Oct. 2008.

[76] J. Zylberberg, J. T. Murphy, and M. R. DeWeese, "A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields," *PLoS Computational Biology*, vol. 7, no. 10, pp. 1–12, Oct. 2011.

[77] D. O. Hebb, *The Organization of Behavior: A neuropsychological theory.* John Wiley & Sons, 1949.

[78] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural System*. Taylor & Francis, 2001.

[79] C. Mead and M. Ismail, *Analog VLSI Implementation of Neural Systems*. Springer, 1989.

[80] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

[81] W. E. Vinje and J. L. Gallant, "Sparse coding and decorrelation in primary visual cortex during natural vision," *Science*, vol. 287, no. 5456, pp. 1273–1276, 2000.

[82] A. S. Ecker, P. Berens, G. A. Keliris, M. Bethge, N. K. Logothetis, and A. S. Tolias, "Decorrelated neuronal firing in cortical microcircuits," *Science*, vol. 327, pp. 584–587, Jan. 2010.

[83] A. Balavoine, J. Romberg, and C. Rozell, "Convergence and rate analysis of neural networks for sparse approximation," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 23, no. 9, pp. 1377–1389, Sep. 2012.

[84] D. Ringach, "Spatial structure and asymmetry of simple-cell receptive fields in macaque primary visual cortex," *J. Neurophysiol*, vol. 88, pp. 455–463.

[85] Advanced Topics in Systems Neuroscience. Redwood center for theoretical neuroscience. [Online]. Available: http://redwood.berkeley.edu/wiki/

[86] E. P. Simoncelli and B. A. Olshausen, "Natural Image Statistics and Neural Representation," *Annu. Rev. Neurosci.*, vol. 24, pp. 1193–1216, May 2001.

[87] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Int. Joint Conf. Neural Networks*, 1990, pp. 537–543.

[88] L. M. Chalupa and J. S. Werner, *The visual neuroscience*. Cambridge, MA, London, England: The MIT Press, 2003.

[89] P. A. Merolla, J. V. Arthur, B. E. Shi, and K. A. Boahen, "Expandable Networks for Neuromorphic Chips," *IEEE Trans. Circuits Syst.-I*, vol. 54, no. 2, pp. 301–311, 2007.

[90] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE Trans. Patten Anal. Mach. Intell.*, vol. 34, no. 4, pp. 791–804, 2012.

[91] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *IEEE Proc.*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[92] The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[93] P. F. Schultz, D. M. Paiton, W. Lu, and G. T. Kenyon. (2014) Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels. [Online]. Available: http://arxiv.org/abs/1406.4205v1

[94] A.-J. Annema, B. Nauta, R. van Langevelde, and H. Tuinhout, "Analog circuits in ultra-deep-submicron CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 132–143, 2005.

[95] C. F. Stevens and Y. Wang, "Changes in reliability of synaptic function as a mechanism for plasticity," *Nature*, vol. 371, pp. 704–707, 1994.

[96] W. Deng, J. B. Aimone, and F. H. Gage, "New neurons and new memories: how does adult hippocampal neurogenesis affect learning and memory?" *Nature Rev. Neurosci.*, vol. 11, pp. 339–350, 2010.

[97] J. Lengler, F. Jug, and A. Steger, "Reliable neuronal systems: the importance of heterogeneity," *PLoS Computational Biology*, vol. 8, no. 12, 2013.

[98] K. Doi, "Current status and future potential of computer-aided diagnosis in medical imaging," *Br. J. Rad. Special Issue*, vol. 78, 2005.