ENERGY-EFFICIENT DIGITAL HARDWARE PLATFORM FOR LEARNING COMPLEX SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Jae Ha Kung

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2017

ENERGY-EFFICIENT DIGITAL HARDWARE PLATFORM FOR LEARNING COMPLEX SYSTEMS

Approved by:

Dr. Saibal Mukhopadhyay, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arijit Raychowdhury
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Dr. Sek Chai
Center for Vision Technologies
SRI International

Date Approved: March 16, 2017

ACKNOWLEDGEMENTS

I sincerely appreciate my advisor Professor Saibal Mukhopadhyay for the full support in every aspect of my doctoral studies. He has been a great academic mentor, and his research vision and guidance let me grow as an independent researcher. Having discussion with him significantly improved the quality of my research work and developed my ability to approach and solve fundamental problems that have great impact. Beside being the academic advisor, he helped me when I was having personal difficulties as if that was his own. It was always been a pleasure to personally talk with him and I will remain thankful.

I also want to thank my thesis committee members, Professor Sudhakar Yalamanchili, Professor Arijit Raychowdhury, Professor Hyesoon Kim and Dr. Sek Chai for their time, efforts and suggestions in improving the quality of the dissertation.

I thank the former and the current colleagues in GREEN Lab at Georgia Tech for providing me an enjoyable environment and going through the hard times together. I am thankful to work with and to know Dr. Minki Cho, Dr. Kwanyeob Chae, Dr. Denny Lie, Dr. Sergio Carlo, Dr. Wen Yueh, Dr. Amit Trivedi, Dr. Boris Alexandrov and Dr. Zakir Ahmed who had graduated before me. Also, it was great to spend time with the future PhDs, Monodeep Kar, Jong Hwan Ko, Taesik Na, Faisal Amir, Arvind Singh, Yun Long and Burhan Mudassar. Especially, I am thankful to Duckhwan Kim for being a great teammate for many research works and being a sincere friend. I will never forget the moments spent with my colleagues and hope everything goes well with their future studies and other plans.

Finally, I greatly appreciate my beloved wife and children, Hana Kang, Jiho and Jia Kung, for supporting my back in all aspects of my life. I also thank my family members in South Korea for all the sacrifices they have made for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

System learning is the most fundamental research area in the engineering domain. It is a modeling method to map external stimulations (inputs) to the corresponding measurements (outputs) with/without physically analyzing the system between them. Applications of the system learning method are electical circuit analysis (two-port network [1]), acoustic signal processing [2], thermal analysis (power-temperature relation [3]), particle physics [4], aircraft engine analysis [5], neural function characterization [6], to name a few. The system can be simple enough, such as a linear time-invariant system, to be easily identified by a simple mathematical model, i.e. a transfer function or a physical equation. However, it can be a quite complex system, such as a nonlinear dynamic system, which is highly difficult to understand with a mathematical representation. Even further, the presence of noise in the system or in input signals makes the learning process more challenging.

Historically, for some engineering or real-world problems, systems of interest were formulated by a set of mathematical equations; *model-based learning*. A variety of physical, biological, chemical behaviors around us are modeled by differential equations. For instance, heat propagates through a medium described by the well-known Poisson's equation [7]. Biological evolution of a species can be modeled by the reaction-diffusion equation [8]. In the engineering domain, joint movements of bipedal robots [9] or path planning of drones [10] can be modeled by coupled differential equations (DE). The model-based learning is efficient in that there is no need to process large volume of data. Yet, numerical simulation of physical models takes long time to run, e.g. Newtons method, finite element method (FEM), or Runge-Kutta method. Thus, many researchers have presented different reduction methods to simplify the learning of physical models [11, 12].

In many other applications, however, physical understanding of the system is not straightforward. For those applications, black box model is utilized for the system learning process. Black box modeling is the method assuming the system as a black box and only utilizes the input and output data from a set of experiments; *data-driven learning*. Neural network is a good substitute in representing that black box. The strength of the neural network comes from the nonlinear activation function (mostly, sigmoid type functions) associated with each neuron in the network. The nonlinear function allows the neural network to express or approximate nonlinear functions [13]. Depending on the type of connections, neural network is able to express a static nonlinear system (feedforward) or a dynamic nonlinear system (both feedforward and recurrent).

There connected portable devices through the network is ever increasing, called Internet of Things (IoT), and they communicate and exchage a bulk of data. The overflowing data are collected from sensors, embedded devices, and personal electronics and they can be processed on-line to understand some unknown systems. Thus, the advantage of neural networks in evaluating those unknown systems related to collected data can be utilized in mobile platforms. Although, application of a neural network as an embedded processing platform has received significant interest [14, 15], low operating power with feasibility of dynamic scaling of energy, performance, and quality-of-result is of critical importance to make it viable as an accelerator in mobile SoCs.

In this thesis, the primary objective is to present the energy-efficient operation of digital hardware to identify a physical system in real-world applications. This is done by first presenting the highly-efficient digital hardware to identify a thermal system of an integrated circuit (IC) where frequency-domain analysis is utilized for the learning process (Chapter 3). This is followed by the design of generic and energy-efficient solver for simulating wide classes of dynamical systems; equivalently, coupled differential equations as a physical model (Chapter 4). To deal with more complex systems, a neural network is selected as the data-driven learning method. The widely-accepted training algorithm of neural networks is being analyzed to develop an energy-efficient hardware platform during the inference of a target neural network, either feedforward or recurrent (Chapter 5, 6). At last, the algorithmic analysis of the impact of error (hardware induced error for achieving low-power operation) on performance for one class of neural networks, called Cellular Nonlinear Network (CeNN), is performed to achieve better energy-efficiency (Chapter 7). The detailed analysis on this computing model is essential in that it is capable of serving as either model-based or data-driven learning method.

CHAPTER 2

BACKGROUND ON SYSTEM LEARNING METHODS

2.1 Model-based Learning

2.1.1 Scientific Modeling: Formulation of Dynamical Systems

Scientific modeling is the process of understanding and predicting specific behaviors of objects or systems. The established model is used to develop fundamentals of science for a long time, from physics and chemistry to earth science and social science. As some systems in real life is extremely complex, scientists make many assumptions to simplify the model to ease the analysis. These assumptions may cause inaccurate estimation of the underlying object/system, for instance the classical Bohr's atomic model. Also, a single modeling may not be sufficient to fully describe the system (or the object) of interest. In that case, multiple models can be coupled together to represent the system with better accuracy.

The dynamical system, represented as coupled ordinary/partial differential (continuous time) or difference (discrete time) equations (ODE/PDE), is a scientific model for describing systems varying in time. The solution of dynamical systems is an important computing problem. The scientific simulations, a key driver for high-performance computing, are defined by solution of coupled ODEs/PDEs. There are many real-time control problems such as bipedal robotic walking, UAV path planning, or aircraft control that necessitates real-time ODE/PDE solution for fast actions [9, 16, 17, 18].

More recently, there is a growing interest in computing with dynamical systems, e.g. reaction-diffusion equations or oscillatory networks [19, 20, 21, 22, 23]. The concept of building a Turing complete machine using reaction-diffusion equations has been proposed [20]. The coupled oscillators based dynamical systems are being explored as a platform for solving complex problems [24, 25, 26, 27]. Computing with biophysical neuron with dynamics modeled using coupled PDEs is another well-known example of dynamical system based computing [22, 23]. The dynamical system based computing is showing promise in solving complex problems in computer vision, graph theory, optimization, to name a few [22, 23, 26, 27].

2.1.2 Hardware Platform for Model-based Learning

There are three major trends in designing hardware platforms for accelerating solution of coupled ODE/PDE to support dynamical system analysis. First, the Graphical Processor Unit (GPU) helps accelerate ODE/PDE solution for scientific computation [28, 29]. The GPU-based platforms provide the advantage of programmability, but in general are more power hungry and less energy-efficient. Second, the FPGA-based coprocessors have also been developed to speed up the computation of conventional numerical analysis methods [30, 31, 32]. However, FPGAs only accelerate the numerical algorithms, do not provide a different computing model. Finally, there is significant effort in exploring ASICs and non-CMOS devices to accelerate specific dynamical systems, for example, simple first/second order equations, oscillatory networks, and spiking systems [22, 23, 25, 26, 27, 33, 34]. The ASIC-based approach provides high energy-efficiency, but lack the capability of solving different types of dynamical systems in a single platform. Likewise, directly using the internal dynamics of a device to build a computing platform does not provide the ability to 'program' a required dynamical behavior.

2.1.3 Computing Model for Simulating Dynamical Systems

A fundamentally different computing model for dynamical system analysis is Cellular Nonlinear Network (CeNN) [35]. The CeNN is composed of an array of cells where each cell follows an ODE based dynamics [35]. Each cell in CeNN is connected to (local) neighboring cells resulting in a system of coupled ODEs. The weight of local connections, referred to as the templates, defines the coupling and hence, the nature of the system of the equations. A multilayer CeNN can realize a system defined by multiple coupled PDEs, where each layer represents the ‘first-order’ equation. The advantage of CeNN based computation originates from the inherent ODE-based cell dynamics, the high-degree of parallelism, local connectivity, and programmability to solve wide classes of ‘system of equations’.

Although the CeNN platform is most widely known for image processing, several past efforts have shown that a multilayer CeNN with linear and nonlinear templates can be used to solve different types of coupled differential equations [36, 37, 38, 39, 40, 41]. There are many studies on how to map a scientific model to CeNN algorithm [38, 39, 41, 42, 43]. Significant research efforts have also been directed to design digital and analog CeNN chips for image processing applications [44, 45, 46, 47]. However, the mapping algorithms developed in prior studies mostly focused on *specific equations* and/or *linear templates*. Likewise, the hardware acclerators were designed for image processing applications with *spatially* and/or *temporally invariant templates*. Therefore, the prior efforts are not sufficient to develop a generic dynamical system simulator with *nonlinear interactions* between equations leading to *space and time variant templates*.

2.2 Data-driven Learning

2.2.1 Fundamentals of Neural Networks

A neural network (NN) is an algorithmic approach for statistical learning of unknown complex functions or systems with large dataset. Various types of NN have been developed over last few decades and utilized in a wide range of applications [48, 49, 50, 51, 52]. NNs have high-degree of parallelism providing opportunities for high system performance with large number of simple processing engines (PEs). The hardware engines for the neural network have received attention for embedding in mobile platforms to perform various applications, for example, medical diagnosis [53], mobile robot motion control [54] and vision systems [55].

In general, NN is composed of nodes (i.e. neurons) and directed edges (i.e. synaptic weights). Each node has activation functions which compute the output of the node with respect to the state of a neuron. This activation function is typically a sigmoid type function or Rectified Linear Unit (ReLU). Depending on the connectivity of neurons, the behavior of the neural network changes significantly (either static or dynamic). In the following subsections, different types of neural networks will be discussed along with their learning algorithms and applications.

A) Feedforward Neural Network

Having unidirectional connections between two nodes at different layers, the network is called a feedforward neural network. The most simplest structure of the feedforward NN is a perceptron proposed by Rosenblatt [56]. In Rosenblatt's perceptron, the output of a neuron is computed by the linear sum of inputs followed by the hard limiter (activation function). In general, a feedforward NN is multiple layers of these perceptrons called multilayer perceptron (MLP) [48]. Due to the nonlinearity of each neuron, MLP is suitable for estimating or identifying any *static* nonlinear system [57].

The learning (or training) of a feedforward NN is a challenging optimization problem (parameter and/or structure optimization). There are two methods on training the feedforward NN: 1) iterative gradient descent (backpropagation algorithm) [58, 59, 60, 61] and 2) evolutionary algorithm (e.g. genetic algorithm) [62, 63, 64]. Mostly, the backpropagation algorithm is used for training due to its lower computational complexity compared to the genetic algorithm. However, it normally tends to settle at a local minima while the genetic algorithm is better at finding the global minimum. Since each optimization method has its pros and cons, hybrid training algorithms are proposed as well for better training performance in terms of the accuracy and the complexity [65, 66].

The applications of interest for the feedforward NN are image recognition (or classification) [60, 67, 68] and function approximation [69, 70, 71]. As explained, the feedforward NN is a good candidate for approximating static nonlinear functions (or systems). That is why it is suitable for classifying images. Basically, the classification problem is equivalent to approximating a nonlinear function that separates the state space into desired number of regions. Thus, for the system learning problem, if a system that we are trying to approximate is desribed by a static nonlinear equation can be identified by utilizing the feedforward NN.

B) Recurrent Neural Network

Given a MLP, if there are additional recurrent connections (forming any cycles in the network), the neural network becomes a recurrent neural network (RNN) [51]. Over several decades, various types of recurrent NNs have been proposed depending on where recurrent connections are established. Nonlinear AutoRegressive with eXogenous inputs (NARX) model has only recurrent connections (delayed by arbitrary time steps) from the output to the input layer (no recurrent connections within the network) [13]. If there are recurrent connections from a single hidden layer to the input layer delayed by unit time step, the network is called Simple Recurrent Network (SRN) [72]. The network is known as a Recurrent MultiLayer Perceptron (RMLP) when recurrent connections are from multiple hidden layers to their previous layers [73]. Owing to the recurrent connections in the network, RNN is capable of approximating any *dynamic* nonlinear systems in theory [74].

As can be expected, training RNN is more complicated than that of the feedforward NN due to its learning capaility of temporal information. There are two conventional training algorithms: 1) BackPropagation Through Time (BPTT) [75] and 2) Real-Time Recurrent Learning (RTRL) [76]. The BPTT algorithm is simply the extension of the (standard) backpropagation algorithm in that it unfolds the temporal behavior of RNN to form a layered feedforward NN. The number of layers of the constructed multilayer feedforward NN linearly increases by an additional time step involved in the training. Also, the memory space required to perform BPTT algorithm increases as the length of a training sequence increases [48].

To overcome these limitations, truncated BPTT algorithm has been proposed by Williams and Peng [75]. Another critical problem that BPTT-type algorithms have is vanishing (or exploding) gradient problem pointed out by Bengio [77]. To deal with this concern, Long Short-Term Memory (LSTM) architecture [52] and Hessian-free training method [78] are proposed. These two novel idea shed light on RNN as a promising NN architecture again.

Because of the limited memory space in hardware, BPTT-type algorithms are suitable for off-line training of RNN. For on-line training, RTRL algorithm can be performed which updates the synaptic weights while operating the network [76]. To train RNN in real-time, the RTRL algorithm approximates the gradient of total error with respect to weight changes by an instantaneous estimate of the gradient for each time step [48]. This deviation from the actual gradient can be minimized by reducing the learning rate η, which determines how fast the gradient descent algorithm evolves. However, the computational complexity of the RTRL algorithm is high since it requires the weight update at every time steps.

As explained, RNN is capable of dealing with systems having nonlinear temporal behaviors which depend on external input sequences. There have been extensive studies on applications of the RNN owing to its ability to utilize the history of past inputs or states. They include appproximation of dynamic systems [79, 80, 81], mapping finite state machines [82, 83], language modeling [84, 85], and associative memory [86, 87], to name a few. Since RNN is more general definition of neural network compared to feedforward NN, detailed understanding of its behavior is required for further development. Finding a good application using RNN is still an open problem by many scientists and RNN has a great amount of potential due to the ability to approximate nonlinear systems.

2.2.2 Impact of Noise on Neural Networks

Understanding the impact of noise on training and/or operation of feedforward NNs is an important research topic to implement them in hardware [88]. When the given network is prone to error from either connection weights or neuron states, the output accuracy of the network will degrade significantly by small perturbations. Thus, error tolerant training methods or guidelines to allow well-defined level of noise (not sacrificing accuracy much) have to be studied. Towards this direction, fault-tolerant training methods [89, 90, 91] have been proposed to realize feedforward NN tolerant to errors to some extent. The effects of noise from the synaptic arithmetic unit during neural network training are studied and noise injection training scheme is proposed [89]. Limiting the absolute value of each connection weight during training, when the value exceeds certain threshold, to flatten the distribution of the magitude of weights enhanced the error tolerance of a feedforward NN [90]. In [91], the convergence of fault injection-based traininng algorithm has been provided.

Apart from the feedforward NN, neural networks with recurrent connections may have stability (or convergence) issues under noise even at the operation of the network. There have been a number of works on providing sufficient conditions for the global stability of various classes of RNNs [92, 93, 94, 95, 96, 97, 98, 99]. It has been shown that a controlled amount of additive errors can improve the convergence and regularization in training RNNs [92]. Recently, the stability condition of RNNs considering transmission delays has been studied [93]. Moreover, the global exponential stability of RNNs considering random noise as well as time delays has been explored [94]. In [94], upper bounds of delays and additive noise for maintaining global exponential stability are mathematically shown. Similar stability criteria studies have been done for other classes of (recurrent) neural networks as well [95, 96, 97, 98, 99]. Many researches have provided the mathematical estimation of the upper limit of connection weight matrices in delayed neural networks (DNNs), which guarantees global stability of DNNs [95, 96, 97]. Most recently, a new criteria for global stability of DNN is shown considering parameter uncertainties on the delayed connection weight matrix [98]. In addition, the stability analysis on Hopfield neural networks (HNNs) with time delays has been done [99].

2.2.3 Neuromorphic Hardware

Neuro-inspired architecture is increasingly recognized as a new paradigm since it is capable of dealing with applications that a conventional von Neumann architecture could not solve efficiently. To accelerate NN computations, many research scientists focused on designing neuromorphic hardware accelerators [100, 101, 102, 103, 104, 105, 106, 107]. Specifically, digital implementation of NNs provide the advantage of easier integration, higher scalability (can handle variable input size), and better programmability.

A) Digital Implementation of Neuromorphic Hardware

The most efficient feedforward NN in image classification and object recognition is a convolutional neural network (ConvNet) [108, 109]. Naturally, its promising performance in those applications led research on hardware implementation of ConvNet accelerators. A group led by Yann LeCun, a ConvNet pioneer, first demonstrated FPGA-based ConvNet based on parallel filter banks to detect objects in real-time [15, 100]. FPGA demonstration was followed by ASIC design of a ConvNet processor, named NeuFlow, developed by a hardware group at Yale University [101]. The computation in NeuFlow is done by parallel processing units where required data stream is provided by Smart Direct Memory Access (Smart DMA) module.

The dataflow model used in NeuFlow limits its energy efficiency since it requires explicit data transfer between processing units. To lower energy consumption in designing ConvNet, a standalone neuromorphic hardware (DaDianNao) with localized memory banks (eDRAM) for each computing unit is proposed [102]. To compensate memory latency and wire congestion, the proposed architecture has a mesh structure with distributed on-chip memory. Recently, another type of energy-efficient ConvNet engines (Hardware Convolution Engine (HWCE) [103] and Origami [104]) was presented. Both accelerators are streaming-based engines integrated with a shared-memory (shared-L1) cluster of RISC processors. They reduce memory access pattern by making only one pixel value to be changed during 2D convolution.

Apart from the ConvNet, Researchers at IBM initiated SyNAPSE project to build a brain-like chip which realizes a spiking neural network (SNN) which is known to have the most similar behavior with human brain. Along the way, *neurosynaptic core*, having 256 digital spiking neurons, has been fabricated in 45nm process; here, spiking neurons can have either 0 or 1 as a state value [105]. This neuromorphic chip locally stores and computes the state of each neuron (utilizing crossbar synapses) and each neuron consumes only 45pJ. Most recently, an advanced version of such low-power spiking neural network engine, named *TrueNorth*, was presented [106]. This brain-like chip is composed of 4,096 parallel processors which represents one million spiking neurons [107].

B) Approximate Computation in Neuromorphic Hardware

Approximate and accuracy-aware computing has emerged as an attractive approach for low-power neuromorphic hardware by trading-off energy and quality-of-result under performance constraints [110, 111, 112, 113]. The most common way to approximate computation is reducing the bit precision of operands [110, 111, 112]. By carefully forcing some LSBs to zero, effective power saving can be achieved with slight degradation in output quality. Most recently, stochastic rounding during training of ConvNet to effectively reduce the bit precision is proposed [112]. Another method to approximate NN hardware is by implementing approximate multipliers as processing engines [113]. An approximate multiplier is designed with iterative logarithmic multiplier having computation error less than 1%. However, having only approximate PEs is limited in saving power while maintaining the output quality.

Even though there have been some level of research on the impact of approximation in neural computation, most research works only deal with feedforward NNs. Having recurrent connections in the network will show completely different error behavior due to the error propagation in temporal direction. Also, having both reduced precision and approximate processing units is not evaluated yet. Thus, the impact of hardware-induced error in recurrent-type networks and the approximation methods utilizing both precision control and approximate hardware will be addressed in the thesis.

CHAPTER 3

HIGHLY-EFFICIENT MODEL-BASED SYSTEM LEARNING ON INTEGRATED CIRCUITS

For an integrated circuit (IC), thermal behavior of the chip is a critical design factor as scaling of device technology significantly increases the dynamic and leakage power of silicon ICs [114, 115]. Higher power density elevates junction temperature and triggers higher leakage current which leads to thermal runaway. Higher temperature also degrades circuit reliability and performance and increases cooling and packaging cost [116]. The thermal problem becomes more intriguing in multi-core processors due to the thermal coupling between neighboring cores and components [117]. Even if a specific core/component is in a low power mode, its temperature can increase when a neighboring core is in the high-power mode due to thermal coupling. For example, measurements on an AMD Fusion processor executing a compute intensive CPU workload demonstrated up to a 13°C rise in temperature in the adjacent idle CPU and GPU cores [118]. As the workload/power pattern of a processor dynamically changes, so does the effect of the thermal coupling and hence, the junction temperature.

The effective thermal management of multicore processors requires on-line estimation of the full-chip thermal patterns. The design-time thermal modeling methods ([119, 120, 121, 122]), although highly accurate for pre-fabrication analysis, are less effective for on-line analysis. It is in principle difficult for design-time modeling to account for post-fabrication (chip-to-chip and potentially time dependent) variations in leakage power, thermal properties of the package, and workload dependent power patterns [123, 124, 125, 126]. On-chip temperature sensors are used in recent processors to sense runtime temperature changes, but they only provide current temperature at specific locations. Several spatial reconstruction methods on temperature distribution have been proposed [127, 128, 129, 130] that can predict full-chip (spatial) temperature distribution including at locations without sensors but only at a given time. These methods do not estimate transient temperature variations at locations without sensors; therefore, has limited ability to estimate spatiotemporal temperature variations.

3.1 Background: Thermal Analysis of ICs

3.1.1 Need for Post-silicon Thermal Analysis

The design-time thermal analysis methods do not account for the post-fabrication variations in electrical or thermal properties. For example, leakage power can experience significant variations. Specifically, the process variations affect the leakage power considerably; leakage current varies 23% to 30% on average [123]. This amount of variation is noticeable since the leakage power is more than half of total power consumption in recent technology nodes. Likewise thermal properties of the package components, e.g. heat spreader, heat sink, and thermal interface materials (TIM), are difficult to extract. The extraction of such properties requires complex measurements and characterizations [124]. These properties can also have chip-to-chip or time dependent variations; for example, the properties of interface materials can vary due to the delamination [125] or the quality of interface adhesion [126]. Fig. 3.1 illustrates how the variations in the thermal properties of a chip can change temperature variations including thermal coupling showing the need for post-silicon thermal analysis.

3.1.2 Related Work on Full-chip Temperature Estimation with Limited Number of Sensors

On-chip temperature sensors are placed in developing recent processors to relieve the performance degradation or reliability hazard caused by elevated chip junction temperature [131, 132]. Temperature sensors, however, are limited to providing current temperature at specific locations where the sensors are placed. Moreover, if the sensors were placed at distant locations from actual hotspot due to placement or routing problems, temperature

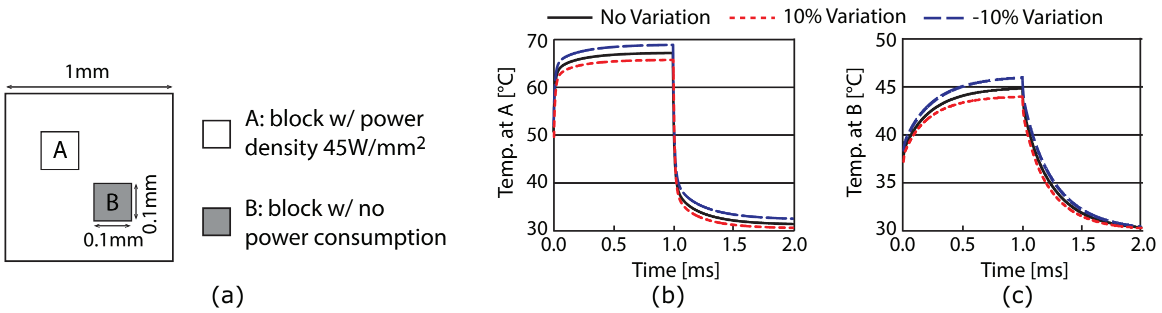


Figure 3.1: An illustrative simulation result using RC-based thermal simulator [119] of the thermal coupling with variations on thermal conductivity of TIM; (a) an example floorplan used for the simulation, (b) transient temperature of power consuming block A, and (c) that of non-power consuming block B.

readings may be lower than the actual maximum temperature (optimistic).

Several thermal analysis methods have been proposed to reconstruct spatial temperature variations of an IC from limited sensor data [127, 128, 129, 130] . Spatial reconstruction methods are proposed to establish full-chip temperature distribution based on reasonable number of on-chip sensors. Interpolation scheme is used to estimate temperature at locations without sensors [127]. To achieve energy efficiency, only a subset of sensors is activated using a dynamic sensor selection mechanism. For more accurate hotspot temperature estimation, thermal characterization using signal reconstruction techniques has been proposed [128]. This technique uses space-domain filters to characterize spatial distribution of temperature. To deal with uncertainties in real-time power consumption, temperature estimation using a linear estimator, approximating power density as a Gaussian random variable, has been proposed [129]. Also, a runtime temperature estimation method using Kalman filter has been proposed [130]. This work provides methodology dealing with noisy sensor readings and process variations. However, these methods only provide full-chip temperature distribution at a given time [127, 128, 129] or next time step [130] (limited to *reactive* thermal management).

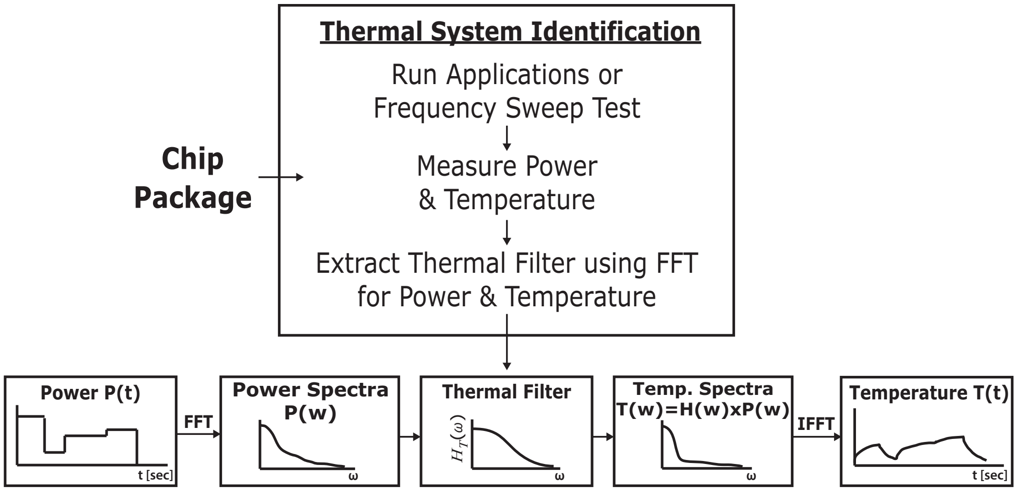

Thermal System Identification
Run Applications or Frequency Sweep Test
Measure Power & Temperature
Extract Thermal Filter using FFT for Power & Temperature
Chip Package
Power P(t)
t [sec]
FFT
Power Spectra P(w)
ω
Thermal Filter
$H_T(\omega)$
ω
Temp. Spectra T(w)=H(w)xP(w)
ω
IFFT
Temperature T(t)
t [sec]

Figure 3.2: The basic concept of Thermal System Identification (TSI) method [3]. The method requires FFT/IFFT computation.

3.1.3 Thermal System Identification (TSI)

A post-silicon thermal analysis methodology - *Thermal System Identification (TSI)* - has been proposed to estimate temperature considering inherent variations after fabrication of the chip [3]. Fig. 3.2 briefly describes the entire process of the TSI method. TSI characterizes the relation between power and temperature of a packaged IC in the frequency domain using on-chip power and temperature measurements. Then, this relation is being used to predict temperature variations in time at one location with a given power pattern using transformation between time and frequency, such as Fast Fourier Transform (FFT). This method enables *proactive* thermal management; future temperature estimates can be utilized in dynamic thermal management (DTM) techniques such as workload allocation in advance.

TSI exploits the well-known analogy between electrical current and heat conduction. By using this analogy, temperature is analogous to voltage, heat flow is analogous to current, thermal conductivity is analogous to electrical conductivity ($\propto 1/R$), and heat capacity is analogous to capacitance (C). This analogy forms the basis of thermal analysis using distributed RC networks [119]. The first-order RC circuit is a simple low-pass filter which makes it possible to think of a thermal system as a complex low-pass filter. Further, an RC network is a linear system which enables the application of superposition principle.

For a simple understanding of TSI, assume a single point (x, y) on a given chip. Power consumption, $P(t)$, at location (x, y) is also assumed to be given. Temperature increase from ambient temperature, $\triangle T(t)$, is then obtained by performing a convolution between power and the impulse response of a thermal system, $h_T(t)$. It can be written as

$$\triangle T(t) = h_T(t) * P(t). \tag{3.1}$$

The convolution operation is simply multiplication in the frequency domain. Hence, Fourier Transform of (3.1) becomes

$$\triangle T(\omega) = H_T(\omega) * P(\omega), \tag{3.2}$$

where $H_T(\omega)$ is the thermal filter response between the power generation and the temperature observation point (one-to-one). Here, TSI can be defined as a method to learn $H_T(\omega)$ in the relation of (3.2). TSI exploits the presence of on-chip temperature sensors at various locations of a chip. Essentially one can apply a periodic power pattern with varying frequency and estimate the associated temperature variation from the sensors at different locations. The measured results can be used to construct the amplitude and phase response of thermal filters between the locations of power dissipation to the locations of temperature sensors.

The method presented in this thesis advances the state-of-the-art of post-silicon thermal analysis over the prior work. First, unlike the prior work, in this thesis, the MIMO thermal filters that consider multiple power sources and multiple temperature sensors were considered and experimentally verified. The experimental verification of the MIMO filter makes TSI more applicable for multi-core processors with thermal coupling between different components (power sources, e.g. cores). The experimental verifications are performed using a 130nm test chip that emulates arbitrary power (and hence, thermal) patterns. Second, the prior work only presented the ability to estimate transient temperature variations at the specific sensor locations. The work reported here overcomes this limitation and developed methods to predict temporal variation in temperature even at locations without sensors.

3.2 MIMO Thermal Filter

3.2.1 Superposition Principle

The power consumption of each functional block generates a certain amount of heat on a silicon die. The heat is naturally being conducted via silicon substrate. This physical phenomenon of generated heat is well described by the Fourier's heat conduction equation, which is

$$\rho C_p \frac{dT(\vec{r},t)}{dt} = \kappa \nabla^2 T(\vec{r},t) + p(\vec{r},t). \tag{3.3}$$

where ρ, C_p, and κ are material properties, $\vec{r}$ is the location of interest, and t is time. $\vec{r}$ could be (x,y) or (x,y,z) depending on the dimension of heat conduction. Here, $T(\vec{r},t)$ is temperature and $p(\vec{r},t)$ is power density at $\vec{r}$.

Among several approaches solving (3.3), the Green's function approach well describes the temperature change at $\vec{r}$ with respect to power sources at different locations, $\vec{r_j} = (\vec{r} - \vec{r_j'})$ [133]. Using the Green's function, temperature at $\vec{r}$ can be computed by

$$T(\vec{r},t) = \iint g(\|\vec{r'}\|, f_0) \cdot p(\vec{r} - \vec{r'}, t) \cdot \vec{r'} dr' d\theta' \tag{3.4}$$

where $g(\|\vec{r'}\|, f_0)$ is the Green's function which is a function of distance $\|\vec{r'}\|$ and the frequency f_0 of the power source. Consider two power sources $p_1(\vec{r} - \vec{r_1'}, t)$ and $p_2(\vec{r} - \vec{r_2'}, t)$ on the chip (Fig. 3.3(a)). Computation of (3.4) assuming there exists one power source gives us the temperature increase due to that heat energy as depicted in Fig. 3.3(b) and (c). Since the integral is a linear operator in (3.4), actual $T(\vec{r},t)$ can be computed by adding

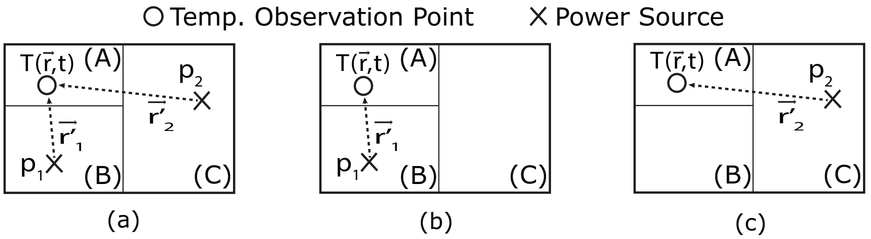


Figure 3.3: Superposition of heat energy from two power sources at a certain temperature observation point.

Table 3.1: Verification of the superposition principle by using the RC-based thermal simulator [119]

On	T↑ at A	On	T↑ at B	On	T↑ at C
B	8.63°C	A	2.84°C	A	1.31°C
C	1.10°C	C	1.07°C	B	3.89°C
B,C	9.77°C	A,C	3.94°C	A,B	5.23°C

each temperature result; $T_1(\vec{r}, t)$ and $T_2(\vec{r}, t)$. This simple example gives us the insight of the superposition principle of the heat conduction.

The superposition principle is being verified by using a RC-based thermal simulator with a simple example [119]. The same floorplan is used as shown in Fig. 3.3(a). Assume each block consumes 2.44W (A), 7.37W (B), and 3.07W (C). These numbers are merely random power numbers. Two functional blocks (B and C) are turned on and temperature is observed at the remaining block (A). It is compared with the temperature estimate, which is the addition of temperature increase when each functional block (B or C) is turned on in isolation. Table 3.1 shows the simulation results and it verifies the superposition principle very well. The first two rows represent the temperature increase at a certain location when one functional block is operating. The last row represents the temperature rise due to the operation of two functional blocks. Additive increase of the first two temperatures is about the same as the temperature increase in the last row.

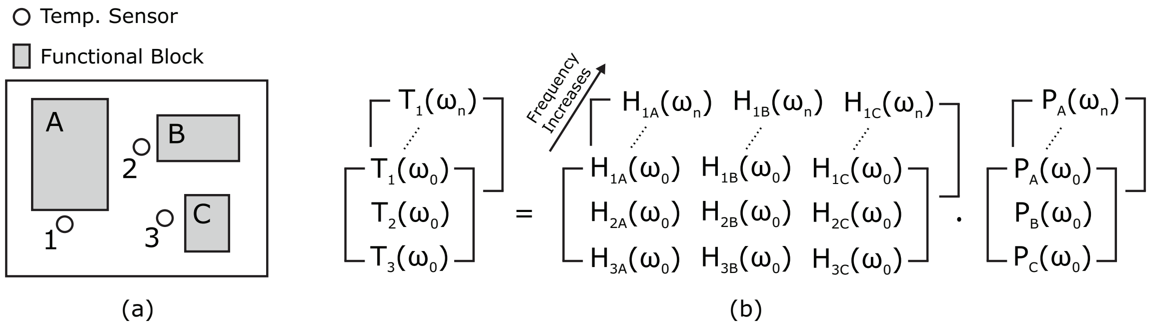

Temp. Sensor
Functional Block
A
B
C
1
2
3
(a)
Frequency Increases
$T_1(\omega_n)$
$T_1(\omega_0)$
$T_2(\omega_0)$
$T_3(\omega_0)$
$H_{1A}(\omega_n)$ $H_{1B}(\omega_n)$ $H_{1C}(\omega_n)$
$H_{1A}(\omega_0)$ $H_{1B}(\omega_0)$ $H_{1C}(\omega_0)$
$H_{2A}(\omega_0)$ $H_{2B}(\omega_0)$ $H_{2C}(\omega_0)$
$H_{3A}(\omega_0)$ $H_{3B}(\omega_0)$ $H_{3C}(\omega_0)$
$P_A(\omega_n)$
$P_A(\omega_0)$
$P_B(\omega_0)$
$P_C(\omega_0)$
(b)

Figure 3.4: (a) A simple floorplan example with multiple power sources and temperature sensors and (b) the corresponding matrix equation of the MIMO thermal filter.

3.2.2 Definition of MIMO Thermal Filter

On the basis of the superposition principle, a MIMO thermal filter can be extracted and utilized in estimating spatiotemporal temperature variations. Each thermal filter response ($H_{ij}(\omega)$ equivalent to $H_T(\omega)$ in (3.2)) between one power source and a sensor location at a given frequency ω_k can be extracted by

$$H_{ij}(\omega_k) = \triangle T_i(\omega_k)/P_j(\omega_k), \tag{3.5}$$

where *i* is the sensor index, *j* is the power source index, and *k* is the frequency index. Recall that a sine- or square-wave power pattern with a fixed frequency ω_k is applied during the thermal filter characterization step. Assume there are three power sources (A, B, and C) and three temperature sensors (1, 2, and 3) on the chip (Fig. 3.4(a)). In this case, nine different thermal filter responses should be extracted to establish MIMO thermal filter.

By the superposition principle, temperature at one location can be estimated by summing temperature changes due to multiple power sources. Thus, temperature increase at sensor *i* when the fundamental frequency of a power source is ω_k becomes

$$\triangle T_i(\omega_k) = \sum_{j \in \{A,B,C\}} H_{ij}(\omega_k) \cdot P_j(\omega_k). \tag{3.6}$$

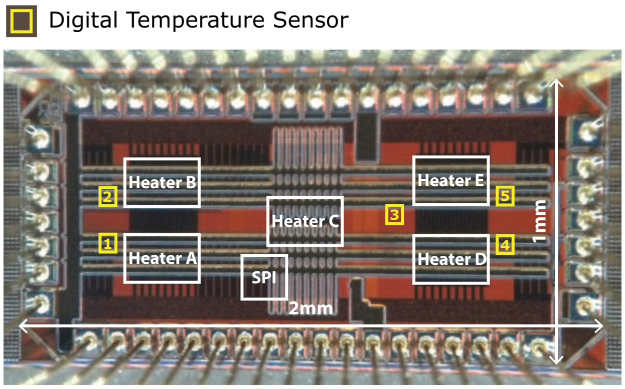


Figure 3.5: The die photo of the test chip; five digitally controllable heaters, five digital sensors, and SPI registers [134].

Then, the relation between power and temperature can be written in a 3-dimensional matrix equation:

$$\triangle\mathbf{T}(\omega) = \mathbf{H_T}(\omega) \cdot \mathbf{P}(\omega), \tag{3.7}$$

where $\mathbf{H_T}(\omega)$ is defined as the *MIMO thermal filter*. Using the relation given by (3.7), the estimation of temporal variations in temperature at sensor locations is possible in the presence of multiple power sources. The estimation of spatiotemporal temperature variations at any locations of interest using the extracted MIMO thermal filter will be discussed in Section 3.5.

3.3 Test Chip: Thermal Emulator

3.3.1 Extraction of MIMO Thermal Filter

The proposed MIMO thermal filter is verified using a test chip, referred to as the *field programmable thermal emulator*, designed and fabricated in 130nm CMOS technology [134]. The test chip has three major components, digital heaters, delay-based temperature sensors, and SPI registers. These components are highlighted and indexed in the die photo of the test chip (Fig. 3.5). An external microcontroller is connected to the SPI interface of the test chip to program desired input power patterns or read digital sensor outputs (Fig. 3.6(a)).

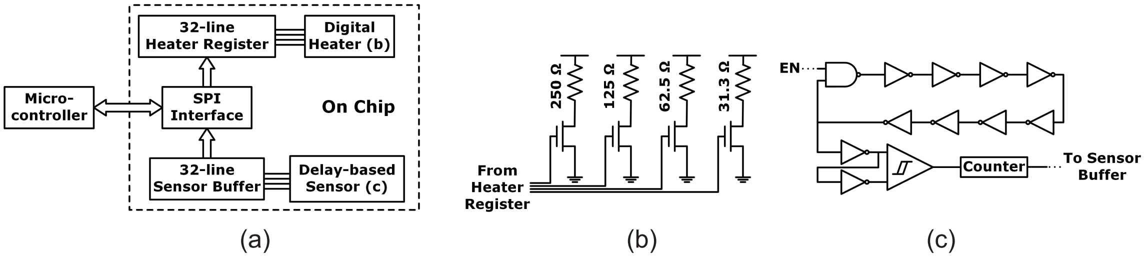


Figure 3.6: (a) A block diagram of the test chip and the microcontroller, (b) a schematic of resistor banks to control power (heat) generation, and (c) a schematic of a delay-based digital temperature sensor.

The programmable CMOS heater is based on n-well resistor to generate heat directly onto the junction. The same sized resistors are grouped together to make 250Ω, 125Ω, 62.5Ω, and 31.25Ω. Each resistor bank has an NMOS footer to digitally control the current conducted across the heater as shown in Fig. 3.6(b). Four different resistor banks allow us to program 16 distinctive levels of quantized power through the SPI interface. Since using the SPI interface, it is possible to turn on/off each heater one-by-one at a time.

The delay-based digital sensor is a nine stage ring oscillator (RO), which is designed to update the counter value every 2ns; operate at 500MHz (Fig. 3.6(c)). By using BJT-based analog temperature sensor, the sensitivity (temperature increase per count) is pre-calculated. Thus, temperature increase is computed by

$$\Delta T = (Count_{amb} - Count_{curr}) \times sensitivity, \tag{3.8}$$

where $Count_{amb}$ and $Count_{curr}$ represent the count value of the RO at the ambient and current temperature, respectively. The count value is inversely proportional to the temperature. The sensitivity of the test chip used throughout the experiment was 0.303°C/count. If $(Count_{amb} - Count_{curr})$ = 100, then the temperature rise becomes 30.3°C. To increase the accuracy of the counter value, the toggle in the digital sensor has been counted for 64 cycles with same power pattern applied at each cycle.

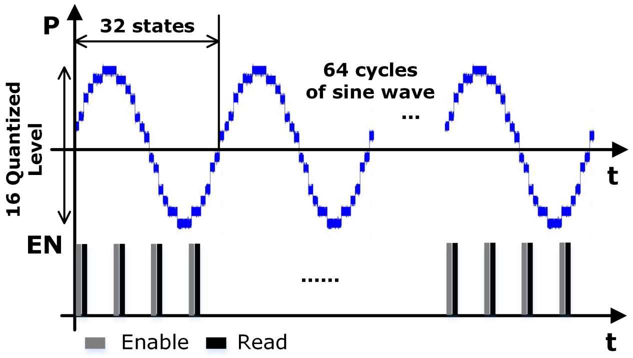


Figure 3.7: The 'EN' signal for the digital sensor reading with 64 cycles of sinusoidal power pattern applied at the heater.

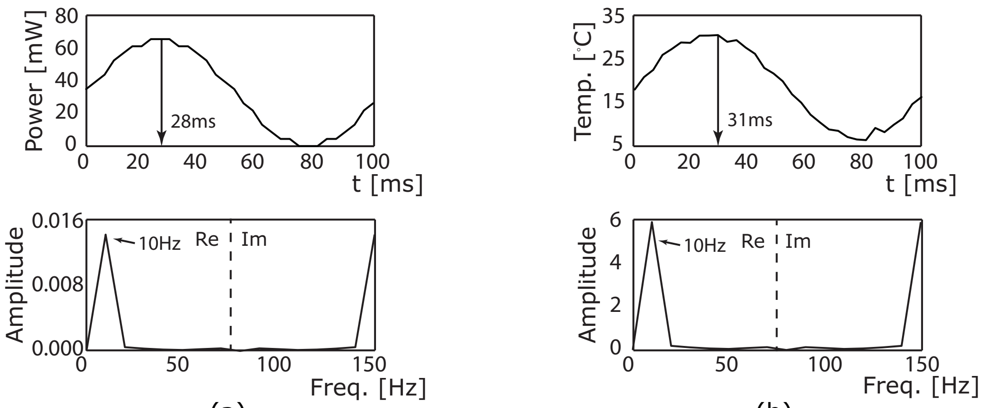
Throughout the experiment, an external microcontroller is used to program the power pattern of heaters and to store the temperature readings via SPI interface. There are 32 4-bit (one footer at each resistor bank) registers to program up to 32 states and these states are used to generate time-varying power pattern. Also, SPI is connected to sensor buffers (16 8-bit registers) in the test chip to read out the digital sensor measurement.

3.4 Experimental Results

3.4.1 Extraction of MIMO Thermal Filter

Using the test chip described in Section 3.2, the sinusoidal input power pattern at 12 different frequencies is applied to each heater to extract MIMO thermal filter. As mentioned earlier, 64 cycles of sinusoidal waves are programmed to the heater to increase the accuracy of the digital sensor readings (Fig. 3.7). Each cycle has 32 states because there are 32 registers for programming the input power pattern. These register values are circulated to repeat 64 times. Further, the sine wave consists of 16 quantized levels controlled by four resistor banks (Fig. 3.6(b)).

The applied power pattern at one heater, e.g. assume heater 'B' in Fig. 3.5, naturally heats up all digital sensors to some extent. The counter at each sensor starts counting when EN signal is 'high (Enable)' and stops counting at the following 'high (Read)' EN signal as



(a) (b)

Figure 3.8: (a) The heater power and (b) the measured temperature both in time- and frequency-domain at 10Hz. Note that the temperature has some delay to follow power transition ($\sim$3ms).

in Fig. 3.7. To read out the temperature at each sensor location through the SPI interface, the count value is stored into a sensor buffer, which has 16 8-bit registers (Fig. 3.6(a)). Here, the count value is represented by 32-bit, which requires four registers to store the full range of its value. It makes the sensor buffer to store 4 sensor outputs at a time; four pairs of 'EN' signals are activated in a single sine wave (Fig. 3.7). To read counter values at all 32 states, the same process (reading four sensor outputs) is done by shifting 'EN' signals by one state to the right (thus, repeated eight times). Each count value stored in a sensor buffer is divided by 64 to obtain an average count value for the corresponding power. Finally, SPI interface allows us to externally obtain the results in sensor buffers. Using (3.8), temperature increase due to the applied sinusoidal power pattern is computed.

The j^{th} column of the MIMO thermal filter matrix $\mathbf{H_T}$ in (3.7) can be obtained by applying a power pattern at heater j and measure temperature at all five sensors. Temperature is obtained by using the test method described previously. The upper figure in Fig. 3.8(b) shows the temperature measurement according to the applied power pattern at 10Hz (Fig. 3.8(a)). Note that there is some delay in temperature tracking the power transition ($\sim$3ms at 10Hz); this results in the phase difference. The power and temperature are then transformed to the frequency domain (using FFT) to compute each element in H_T by

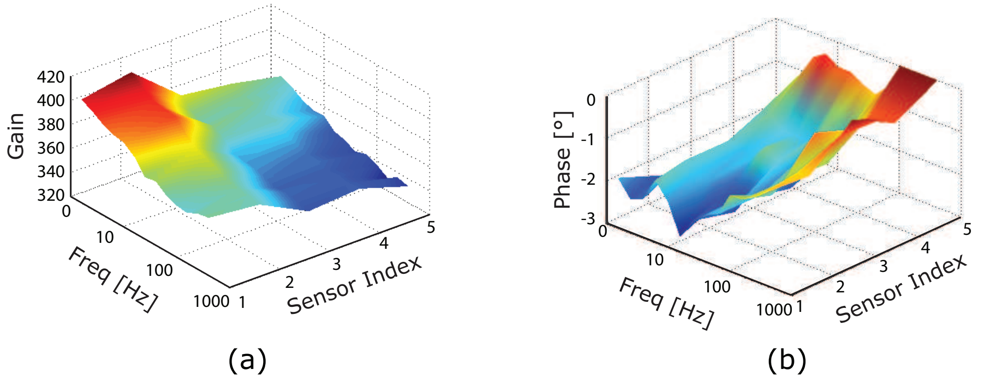


Figure 3.9: (a) The amplitude response and (b) the phase response of the MIMO thermal filter extracted from our test chip when heater 'B' is turned on.

using (3.5). Power and temperature in frequency domain are also presented at the bottom of Fig. 3.8(a) and (b) as a reference. The amplitude of the power or temperature in frequency domain dominates at the fundamental frequency, which is 10Hz in this example. Fundamental frequencies ($\omega_k/2\pi$) from 1Hz to 1kHz were tested during the MIMO thermal filter characterization.

Similarly, all elements in $\mathbf{H_T}$ can be computed. As a result, the amplitude and the phase response are extracted when heater 'B' is turned on. As shown in Fig. 3.9, the amplitude response of the MIMO thermal filter is similar to a low pass filter. The gain at sensor #2 is the largest since it is at the closest location to the heat source. The phase response also has to show low-pass like response [135]. The extracted phase response, however, is somewhat different due to the responsiveness of our delay-based digital sensor. The negative phase means temperature change lags the transition in power, which is expected.

3.4.2 Accuracy of the Temperature Prediction Using MIMO Thermal Filter

Using the extracted MIMO thermal filter, temperature variation at each sensor location is estimated. Temperature estimates are compared to measured temperature using on-chip digital temperature sensors. Two different random power pattern sets are tested to verify the accuracy of the MIMO thermal filter. The set 1 uses two heaters ('A' and 'D') while set 2 turns on all five heaters. The input power set 1 is represented by gray solid line

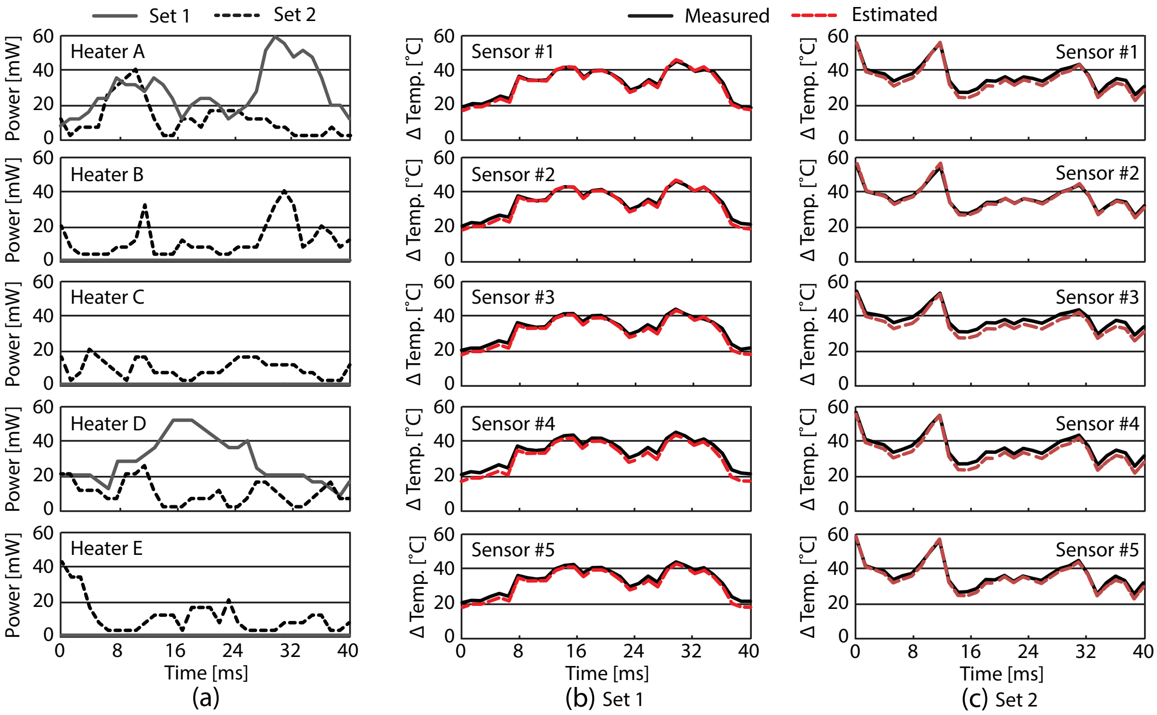


Figure 3.10: (a) Two sets of tested input power pattern at each heater and (b-c) the comparison between the measured temperature and the estimated one when input power pattern is set 1 (b) and set 2 (c).

and set 2 is shown with black dotted line in Fig. 3.10(a). Fig. 3.10(b) and (c) show the variations in temperature when power pattern set 1 and set 2 are used, respectively. For both cases, measured temperature by using on-chip delay-based temperature sensors is shown with black solid line and the estimated temperature with red dotted line. As shown in the figure, it is possible to estimate on-chip temperature with high accuracy using the proposed MIMO thermal filter. To analyze the accuracy, estimation error is computed by $(T_{measured} - T_{estimated})$. As a result, average estimation errors were 0.62°C for set 1 and 1.98°C for set 2, where the standard deviation (σ) of error for each case was 0.21°C and 0.11°C.

However, there can be noise in temperature sensors, for example, due to variations in process parameters of the sensor or thermal noise, which affects the extraction process of thermal filter response. Note the measured sensor responses already capture the effect of

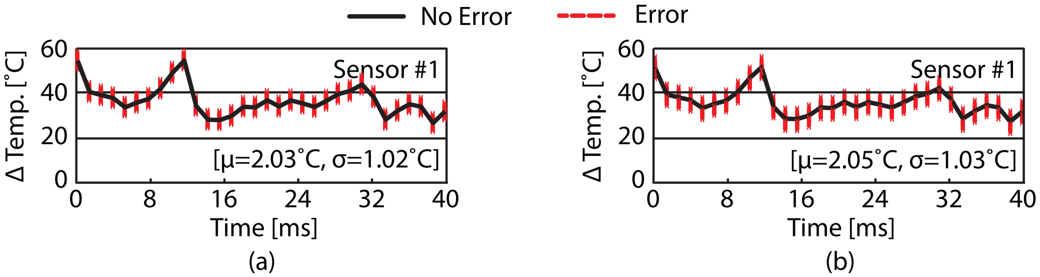


Figure 3.11: The comparison of the estimated temperature in sensor #1 (only showing #1 for brevity) without sensor error and with sensor error (100 simulations) for (a) normal power and (b) low power temperature sensor case.

sensor noise. However, to further evaluate the effects of sensor noise and hence, estimation errors introduced in the extracted thermal filters, additional noise has been applied to temperature sensor readings and extracted the thermal filter response. Two different cases are simulated: 1) normal-power temperature sensors ($>1\mu$W) [136, 137] and 2) low-power temperature sensors ($<1\mu$W) [138]. For normal-power sensors, they have sensing error range about [-0.8°C:0.8°C] while low-power sensors have error range about [-1.5°C:1.5°C]. Thus, the standard deviation of 0.3°C and 0.5°C for each case is selected when generating random errors on sensor readings.

By applying the same power patterns, the set 2 in Fig. 3.10(a), the temperature using re-extracted (erroneous) thermal filter is estimated. As a reference, the average estimation error was 1.98°C with standard deviation of 0.11°C when there is no sensor error. A hundred different sets are simulated by applying random noise with *Normal* distribution to sensor readings during the filter extraction for each case (normal- and low-power). As a result, each average estimation error compared with measurement data was 2.03°C and 2.05°C with standard deviation of 0.12°C and 0.13°C. The comparison plot for sensor #1 between temperature estimation without sensor error and with error (100 simulations) is also shown in Fig. 3.11. This indicates that the MIMO thermal filter can predict the temperature variation with reasonable accuracy even under reasonable sensor noise.

3.5 Temperature Estimation at Locations Without Temperature Sensors

3.5.1 Interpolated Thermal Filter

In the previous section, the accuracy of the estimated temperature at sensor locations using the MIMO thermal filter is validated. Temperature estimation at locations without temperature sensors is also important to reconstruct the full-chip thermal map using limited number of temperature sensors. Especially, estimating/predicting the hotspot temperature where sensors are not placed is very important.

The extracted MIMO thermal filter directly interprets the physical relation between power and temperature. It differs from the previous work that uses the sensor readings to reconstruct the full-chip temperature distribution [127, 128]. Even though some other works appreciate the relation between power density and temperature solving the heat conduction equation (3.3), they use constant thermal parameters for package components [129, 130]. The variations in these thermal parameters will result in computation error during post-silicon temperature estimation. In addition, the accuracy of temperature estimation relies on the distribution of power density [129] or the location of temperature sensors [130].

In this thesis, I present the interpolation method to establish thermal filter response at any locations of interest with no sensors. Each $H_{ij}(\omega_k)$ has the amplitude and the phase information. The amplitude of each thermal filter, $|H_{ij}(\omega_k)|$, changes over space (distance) and is affected by ω_k. Assume temperature sensor #1 is not available in Fig. 3.5. If we are interested in the temperature at location #1, we need to estimate $|H_{1j}(\omega_k)|$ from the extracted $|H_{2j}(\omega_k)| \sim |H_{5j}(\omega_k)|$. Due to the strong spatial correlation of within-die variations [139], the extracted thermal filter response of the sensor closest to the location of interest is used as a baseline for the interpolation. Since the location of sensor #2 is the closest to location #1, interpolated amplitude response of $|\tilde{H}_{1j}(\omega_k)|$ can be obtained by

$$|\tilde{H}_{1j}(\omega_k)| = f(|H_{2j}(\omega_0)|, \|\vec{r_1} - \vec{r_j}\|, \omega_k), \tag{3.9}$$

where $\|\vec{r_1} - \vec{r_j}\|$ is the distance from the observation point #1 to the power (heat) source j, ω_k is the k^{th} frequency component of the power source, and $|H_{2j}(\omega_0)|$ is the gain at ω_0 (the lowest test frequency; 1Hz in our experiments) of sensor #2. $f(\cdot)$ is an empirical function due to the process variations and different packaging, thus it needs to be fitted post fabrication.

The gain far from the power source decreases relatively slower as the frequency ω_k increases. This is mainly because the heat energy does not affect the temperature at remote locations when the power oscillates quite fast; note that more heat flows vertically to a heat spreader when the frequency of power source increases [133]. For the simplicity, the same phase response obtained at the closest sensor location can be used. Then, the estimated phase at location #1 becomes

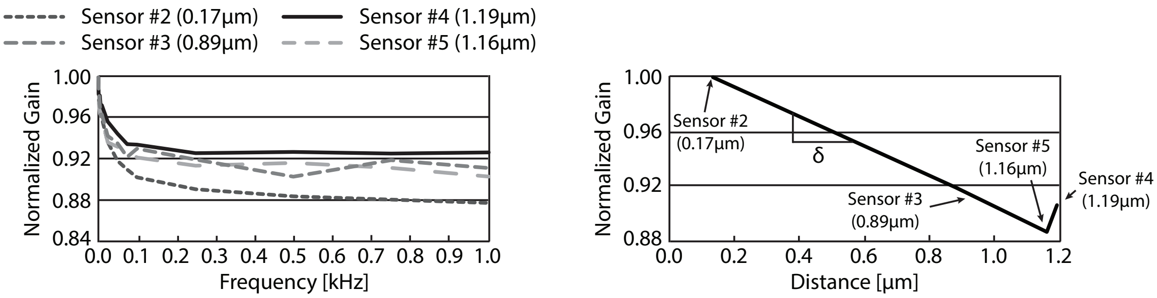
$$\angle\tilde{H}_{1j}(\omega_k) = \angle H_{2j}(\omega_k). \tag{3.10}$$

Now, the thermal filter response at a location without the temperature sensor becomes

$$\tilde{H}_{1j}(\omega_k) = |\tilde{H}_{1j}(\omega_k)| \cdot e^{j\angle\tilde{H}_{1j}(\omega_k)}, \tag{3.11}$$

where $\tilde{H}_{1j}(\omega_k)$ is defined as the *interpolated thermal filter*.

The limitation of interpolation could be under-fitting the extracted filter response. To achieve high accuracy even with less data points, we have to keep in mind that the accuracy of fitting the thermal filter response at lower frequencies is more important than that at higher frequencies (DC gain is the most important value to be fitted). Thus, when fitting the curve, the best approach is to fit all data points but if that is not feasible targeting for the lower frequency portion is a good option. Also, placing temperature sensors to collect data with various distances from each heat source is necessary since the interpolation method is utilized (thus, more data higher accuracy). Having sensor data at similar distances is not preferred.



(a)

(b)

Figure 3.12: (a) The normalized gain of each sensor (except sensor #1) at different frequencies of the heater 'B'; the distance from the heater to each sensor is denoted inside the parentheses and (b) the normalized gain of each sensor (except sensor #1) as a function of the distance from heater 'B'.

3.5.2 Experimental Validation

To validate the accuracy of the proposed interpolated thermal filter, let us assume sensor #1 is not available from our test chip. Then, we have to reconstruct the filter response at the sensor location #1 from the data already extracted by using temperature sensors (#2∼#5). To observe the impact of the frequency of a heat source on the gain, I have plotted the normalized gain of each sensor when heater 'B' is turned on with different fundamental frequencies (Fig. 3.12(a)). The gain is normalized to the gain at 1Hz (the lowest test frequency) of each temperature sensor. The gain at a location far from the heat source decreases relatively slower compared to the locations closer to the heat source. In Fig. 3.12(a), the distance from heater 'B' to each sensor location is denoted inside the parentheses. Fig. 3.12(a) tells us that the gain decays as a function of both distance and frequency. This decaying function D is inversely proportional to frequency and decays faster if the observation point (distance) is closer. Thus, D is empirically defined by fitting the curve in Fig. 3.12(a):

$$D(\|\vec{r_i} - \vec{r_j}\|, \omega_k) = (2\pi/\omega_k)^{\alpha/\|\vec{r_i}-\vec{r_j}\|^{\beta}}, \tag{3.12}$$

where $\|\vec{r_i} - \vec{r_j}\|$ is the distance from the observation point i to the power source j, α and β are fitting coefficients, and ω_k is the k^{th} frequency component of the power source. For

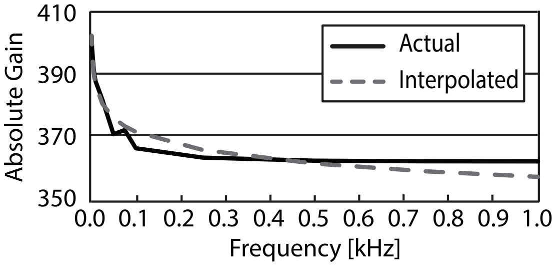


Figure 3.13: The comparison between the actual thermal filter response and the interpolated one at sensor location #1 when heater 'B' is on.

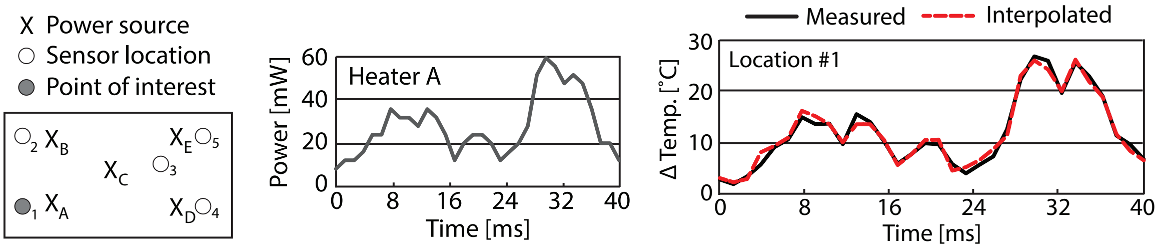
our test chip, $\alpha = 0.12$ and $\beta = 0.25$ are used. These fitting coefficients in (3.12) can be adjusted to fit the amplitude response post fabrication.

So far, the gain normalized to the gain at 1Hz of each temperature sensor was used. However, the absolute gain of each temperature sensor differs depending on its distance from the heat source. As predicted, the gain decreases if the distance of an observation point increases. From our test chip, the relation between the gain of each temperature sensor at 1Hz and the distance from the heat source is obtained. The gain is negatively linear to the distance as shown in Fig. 3.12(b). From this simulation result, the absolute gain at the observation point, where no temperature sensor is available, can be obtained by linear interpolation:

$$|H_{ij}(\omega_0)| = |H_{i^*j}(\omega_0)| + \delta \cdot (\|\vec{r_i} - \vec{r_j}\| - \|\vec{r_{i^*}} - \vec{r_j}\|), \tag{3.13}$$

where δ is the slope of the linear function (refer to Fig. 3.12(b)), ω_0 = 1Hz, and i^* is the sensor index closest to the observation point i, i.e. sensor #2.

By multiplying the result of (3.13) with (3.12), it is possible obtain an interpolation function $f(\cdot)$ in (3.9) for the location i. Fig. 3.13 shows the comparison between the interpolated and actual thermal filter response for location i = #1 when heater 'B' is turned on. This process is done for each heater (A$\sim$E) to obtain the complete thermal filter response at location #1. By using the interpolated thermal filter, it is possible to estimate temperature at a location without temperature sensor. Further, the interpolated thermal filter enables the



(a) (b) (c)

Figure 3.14: (a) A tested floorplan assuming sensor #1 is not available, (b) applied power pattern at heater 'A' (no power consumption elsewhere), and (c) the accuracy of temperature estimation at location #1 using the proposed interpolated thermal filter.

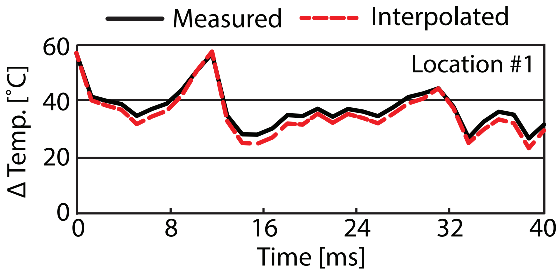


Figure 3.15: The accuracy of temperature estimation at location #1 (assuming there is no temperature sensor at #1) when power pattern set 2 is used from Fig. 3.10(a).

estimation of temperature higher than readings from on-chip temperature sensors. This is not possible for estimation methods solely relying on sensor readings.

Assume that there is no sensor at location #1 while other sensors are available (Fig. 3.14(a)) and functional block 'A' is consuming power (Fig. 3.14(b)). Then, temperature is estimated by using the interpolated thermal filter obtained by (3.11). The estimation result is shown in Fig. 3.14(c) and the estimation error was 0.73°C on average (σ = 0.81°C). For the verification of the more general case, input power pattern set 2 in Fig. 3.10(a) is used (all five heaters are turned on). The estimated temperature using the interpolated thermal filter is compared to the measured one from temperature sensor #1 (Fig. 3.15). As shown in the figure, it accurately estimates actual temperature with a small error bound. The estimation error was 2.08°C on average while σ of the error was 1.09°C.

To increase the credibility of the interpolation result, the same set of experiments was

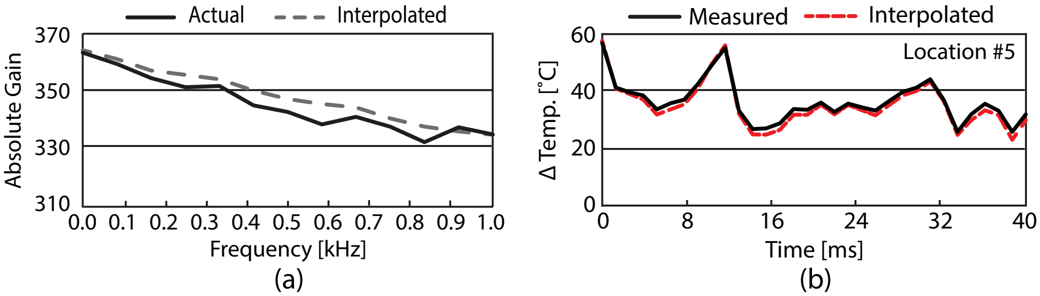


Figure 3.16: (a) The comparison between the actual thermal filter response and the interpolated one at sensor location #5 when heater 'B' is on and (b) the accuracy of temperature estimation at location #5 when power pattern set 2 is used from Fig. 3.10(a).

performed assuming that sensor #5 is not present. By applying the same method as in the example when sensor #1 is not present, an interpolated thermal filter for the location #5 is compared with the actual thermal filter response when heater 'B' is on (Fig. 3.16(a)). It has more flat curve compared with the thermal filter response of sensor #1 case (refer to Fig. 3.13). This is because location #5 is far from heater 'B' compared to sensor #1 making the heat affect less even with the steady power source (less gain at lower frequencies). In addition, the same power pattern (set 2 in Fig. 3.10(a)) is used for the general verification. As a result, temperature was estimated with average error of 1.98°C with σ of 0.12°C (Fig. 3.16(b)).

3.6 Hardware Design of MIMO Thermal Filter

In this section, hardware designs of the proposed MIMO thermal filter will be discussed. So far, the accuracy of temperature estimation using MIMO thermal filter was verified even at any locations of interest. Temperature at each observation point ($\vec{r_i}$) can be predicted by matrix multiplication in frequency domain when power distribution is provided by using an architecture level power estimator [140, 141, 142]:

$$\Delta T(\vec{r_i}, t) = F^{-1}\{\Delta T(\vec{r_i}, \omega)\} = F^{-1}\{\sum_{j=1}^{n} H_{ij}(\omega) \cdot P(\vec{r_j}, \omega)\}. \tag{3.14}$$

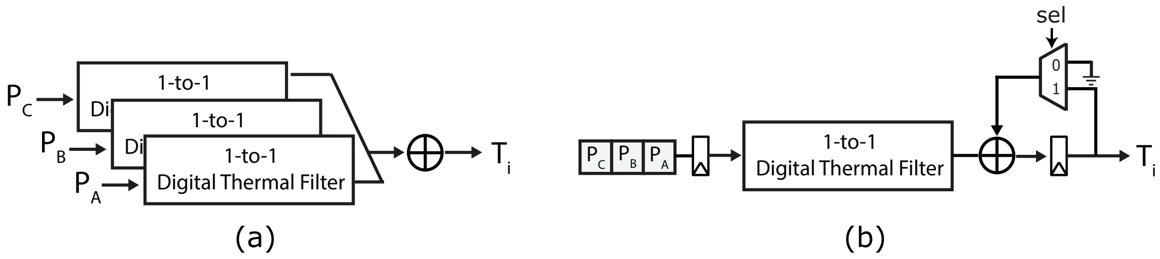


Figure 3.17: On-chip time-domain MIMO thermal filter implementation: (a) the thermal filter banks and (b) the pipelined thermal filter.

For the computation of (3.14), FFT is required before the multiplication to obtain the power spectra. After the multiplication, IFFT is utilized to obtain the temperature in the time-domain. This FFT/IFFT operation can be performed in software or hardware. To achieve faster responsiveness (less slowdown in applying DTM techniques), computation should be done in hardware. To remove the use of a complex FFT/IFFT hardware as in (3.14), a simple on-chip digital thermal filter has been designed to predict the temperature at a single location [135]. To extend the usage into a MIMO thermal system, thermal filter banks or a pipelined thermal filter can be designed as shown in Fig. 3.17. On-chip thermal filter banks can be placed any locations where the placement of the thermal filter does not affect routing of critical interconnects. This is another advantage of using thermal filters instead of temperature sensors which should be placed near hot spots to increase the accuracy [131].

To estimate the area, the dynamic power and the performance of the proposed digital hardware in [135], Nangate 45nm technology library [143] is used and synthesized the RTL design using Synopsys Design Compiler [144]. Then, PnR is executed for more accurate area estimation using Cadence Encounter [145]. An on-chip thermal filter is composed of a multiplier and an adder which performs the convolution operation. Also, a 2-to-1 mux is placed to selectively add previously computed values. A block diagram of the designed on-chip thermal filter is shown in Fig. 3.18. For the physical synthesis, a target frequency is set to 100MHz. To meet the target clock frequency after the physical synthesis, RTL synthesis is re-executed with tighter timing constraints (6ns). Then, PnR is re-executed

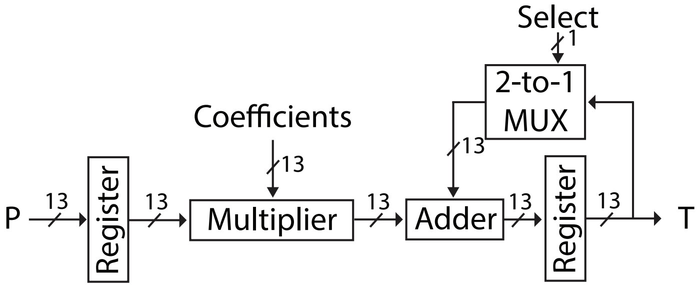


Figure 3.18: A block diagram of an on-chip digital thermal filter implementation.

satisfying the target clock frequency as shown in Fig. 3.19(a).

Operating at 100MHz, the reported area was 0.0022mm^2 which is reasonably small to put multiple thermal filters in a SoC (Fig. 3.19(a)). The total power consumption was 364.2μW after PnR (Fig. 3.19(b)). In terms of performance, it predicts temperature in '$n^2/2$' cycles when the length of the input power pattern is 'n'. For instance, when the power pattern consists of 1,000 data points and the filter operates at 100MHz, we can obtain temperature estimates in 5ms (Fig. 3.19(b)). When the sampling time of the power pattern is 1ms, one second of temperature profile can be predicted within 5ms. As expected, the computation time can be reduced by increasing the clock frequency, and hence, at the expense of higher power. As a projection to actual implementation, a hundred on-chip thermal filters to handle 10×10 MIMO thermal system occupy the area of 0.22mm^2 (consuming 36.4mW). It is definitely possible to utilize these on-chip filters to estimate larger than 10×10 MIMO thermal system by combining filter banks and pipelined filter topology.

3.7 Summary of the Chapter

The proactive estimation of thermal fields is an important ingredient in the design of the next generation of multicore processors. It is challenged by die-to-die and package variations in physical properties precluding accurate design time characterization. In this chapter, a new methodology for the post-silicon estimation of thermal fields was presented. The

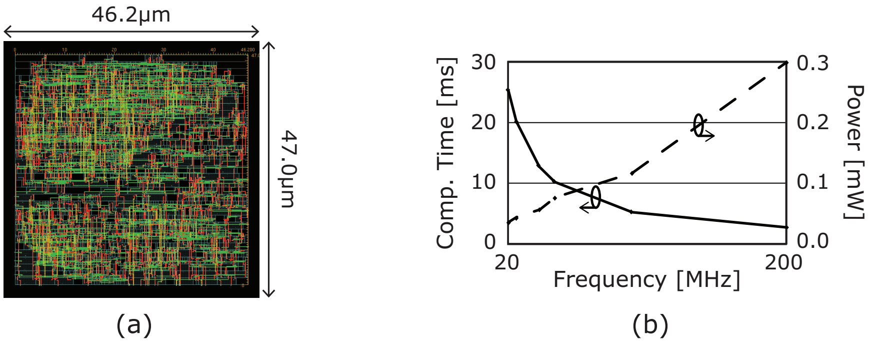


Figure 3.19: (a) The layout of the on-chip thermal filter after the physical synthesis and (b) its dynamic power consumption and the computation time.

approach is measurement based approach to the construction of a MIMO thermal filter. This methodology employs an interpolation method that enables the accurate estimation of the temperature at any location of interest. The accuracy of the proposed MIMO thermal filter is demonstrated with several experiments in a 130nm CMOS test chip. With the proposed MIMO thermal filter, temperature was estimated to be within 2°C error (3.5%). The estimation error was within 2.1°C even at the location without temperature sensor using the interpolated thermal filter. Finally, the time-domain hardware designs of the proposed MIMO thermal filter are illustrated. Collectively, these techniques enable proactive thermal management based on models customized to the specific silicon and packages.

CHAPTER 4

A GENERIC AND ENERGY-EFFICIENT ACCELERATOR FOR SIMULATING DYNAMICAL SYSTEMS

In Chapter 3, the efficient learning of a specific system, the heat propagation in ICs, was presetend. However, there is a variety of systems with completely different behaviors, thus the hardware accelerating the system learning of the specific type is not desirable. Accordingly, the fast and energy-efficient simulation of more generic systems defined by coupled ordinary/partial differential equations becomes an important problem. The accelerated simulation of coupled ODE/PDE is critical in learning/analyzing physical systems as well as computing with dynamical systems. In this chapter, the design of a fast and programmable accelerator for simulating dynamical systems will be discussed. This solver will enable the model-based learning even more efficient and generic.

The computation speed and the programmability are key features in designing a real-time generic solver to analyze dynamical systems. There are control problems necessitating the real-time update to enable faster response such as bipedal robotic walking, UAV path planning, or aircraft control [9, 16, 17, 18]. Some systems can be in massive-scale making the computation time too high in applying complex numerical algorithms [146, 147, 148]. Also, a hardware platform should be programmable to cover a wide range of dynamical systems, including coupled or even chaotic systems. The programmability implies not only the generality of target systems but also the granularity of its solution space.

Mostly, dynamical systems are modeled as coupled differential equations [9, 149, 150, 151]. The coupling between oscillators, where an individual oscillator showing simple and periodic dynamical behavior, can describe richer classes of dynamical systems [24, 25]. The acceleration of finding a numerical solution of a given dynamical system (or coupled differential equation) has been active research focus in both scientific and engineering do-

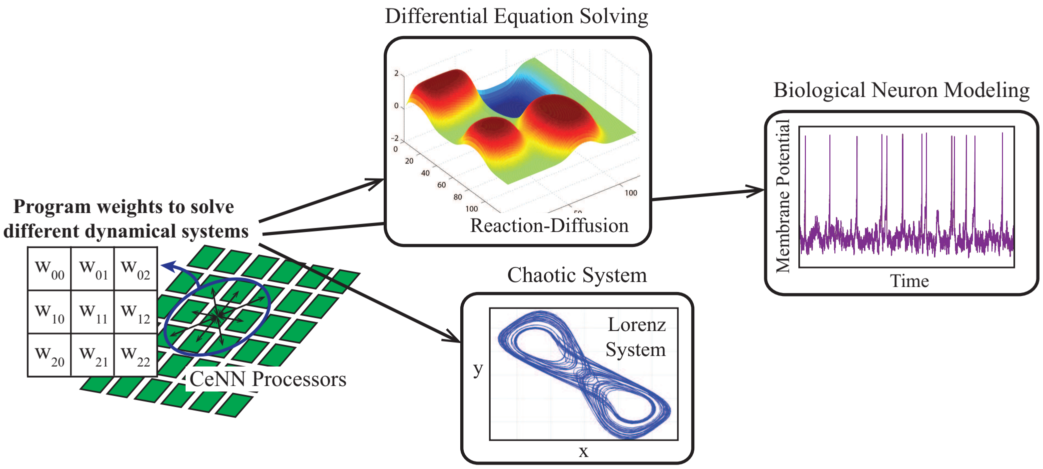


Figure 4.1: Programmable computing model by using Cellular Nonlinear Network (CeNN) with some dynamical systems that can be modeled by CeNN.

mains [28, 148, 152, 153]. As the numerical complexity of solving sets of differential equations limits the accruacy of solution, having a programmable accelerator along with CPU/GPU clusters is required.

A fundamentally different computing model for dynamical system analysis is Cellular Nonlinear Network (CeNN) [35]. The CeNN is composed of an array of cells where each cell follows an ODE based dynamics [35]. Each cell in CeNN is connected to (local) neighboring cells resulting in a system of coupled ODEs. The weight of local connections, referred to as the **template**, defines the coupling and hence, the nature of the system of the equations. A multilayer CeNN can realize a system defined by multiple coupled PDEs, where each layer represents the 'first-order' equation. The advantage of CeNN based computation originates from the inherent ODE-based cell dynamics, the high-degree of parallelism, local connectivity, and programmability to solve wide classes of 'system of equations' (Fig. 4.1).

Although the CeNN platform is most widely known for image processing, several past efforts have shown that a multilayer CeNN with linear and nonlinear templates can be used to solve different types of coupled differential equations [36, 37, 38, 39, 40, 41]. There are many studies on how to map a specific equation to CeNN algorithm [38, 39, 41, 42, 43]. Significant research efforts have also been directed to design digital and analog CeNN chips for image processing applications [44, 45, 46, 47]. However, the mapping algorithms developed in prior studies mostly focused on *specific equations* and/or *linear templates*. Likewise, the hardware accelerators were designed for image processing applications with *spatially* and/or *temporally invariant templates*. Therefore, the prior efforts are not sufficient to develop a generic dynamical system simulator with *nonlinear interactions* between equations leading to *space and time variant templates*.

4.1 Computation Model

A CeNN is defined as a regular structure where cells are locally connected to their neighboring cells within a given radius. The dynamics of each cell in CeNN is defined by:

$$\begin{aligned}\frac{\partial x_{ij}(t)}{\partial t} &= -x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} \hat{A}_{kl}(t)\cdot x_{kl}(t) + \\ \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot y_{kl}(t) &+ \sum_{C(k,l)\in N_r(i,j)} B_{kl}\cdot u_{kl}(t) + z\end{aligned} \tag{4.1}$$

$$y_{ij}(t) = f(x_{ij}(t)) = \begin{cases} -1 & \text{for} \quad x_{ij}(t) < -1 \\ x_{ij}(t) & \text{for} \quad |x_{ij}(t)| \leq 1 \\ 1 & \text{for} \quad x_{ij}(t) > 1 \end{cases} \tag{4.2}$$

where i is the row index, j is the column index, $x_{ij}(t)$ is the state, $y_{ij}(t)$ is the output, $u_{ij}(t)$ is the input, z is the offset, $\hat{\mathbf{A}}$ is the state (feedback) template which is a function of time-varying state variables $x_{ij}(t)$ and $x_{kl}(t)$, $\mathbf{A}$ is the output (feedback) template, and $\mathbf{B}$ is the feedforward template for each cell $C(i,j)$. Here, $N_r(i,j)$ represents neighbors of a cell $C(i,j)$ within radius r where (k,l) is the index of those cells.

In multilayer CeNN, each 2D array defines one equation discretized in space, while the connections between nodes in different layers represent the coupling between different equations (Fig. 4.2). The prior efforts in exploring CeNN based differential equation

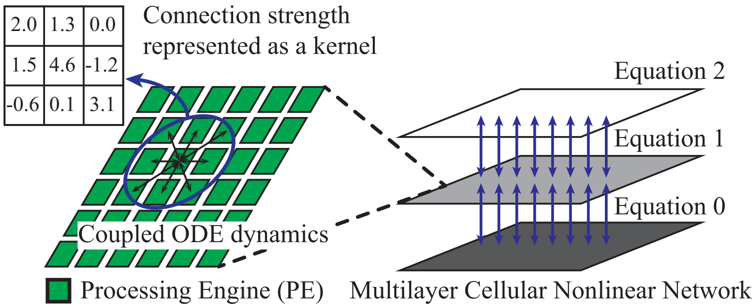


Figure 4.2: A 2-dimensional CeNN processing array having cell states locally coupled. This structure can be extended to a multilayer CeNN platform to handle coupled systems.

solvers only focused on space-/time-invariant linear templates. We develop a more generic approach to map multiple coupled equations with nonlinear interactions by programming the template weights, ($\hat{A}_{kl}(t)$, A_{kl}, and B_{kl}).

The first step is to determine the number of layers as a function of the number of variables involved and the highest order of derivatives for each variable in the system. Second, we identify each first-order differential equation and map that to a layer within the multilayer CeNN. Assume a dynamic system described by the following coupled differential equations:

$$\ddot{\omega} = f_1(\omega, \varphi) \text{ and } \dot{\varphi} = f_2(\omega, \varphi). \tag{4.3}$$

As CeNN cell dynamics is described by the first-order ODE, equation (4.3) is re-written as

$$\dot{\omega} = \chi \text{ and } \dot{\chi} = f_1(\omega, \varphi) \text{ and } \dot{\varphi} = f_2(\omega, \varphi). \tag{4.4}$$

Third, while mapping an equation to a layer in CeNN, we will identify whether the equation involves linear or nonlinear templates, and program the templates following the methods explained in Sections 4.1.1 and 4.1.2.

4.1.1 Mapping Linear Systems

A dynamical system can be described as a scalar ODE given as $\dot{\varphi} = g(x)$ where $g : \mathbb{R} \to \mathbb{R}$ is a continuous function. Let us first explain the mapping process when $g(x)$ is a linear equation, for example, heat equation given by:

$$\frac{\partial\varphi(x,y,t)}{\partial t} = \kappa \cdot \Delta\varphi(x,y,t), \tag{4.5}$$

where $\varphi(x,y,t)$ is temperature at location (x,y) at time t and κ is thermal conductivity. As CeNN cell dynamics in equation (4.1) is an ordinary differential equation (ODE), Laplace operator is discretized by Euler method. Then, equation (4.5) can be approximated as

$$\begin{aligned}\frac{\partial\varphi(x,y,t)}{\partial t} &= \kappa \cdot \{\frac{\varphi(x+h,y,t)+\varphi(x-h,y,t)-2\cdot\varphi(x,y,t)}{h^2} \\ &\quad + \frac{\varphi(x,y+h,t)+\varphi(x,y-h,t)-2\cdot\varphi(x,y,t)}{h^2}\},\end{aligned} \tag{4.6}$$

where h is the step size in $\mathbb{R}^2$ Euclidean space. With an infinitesimal h, the approximation error goes to zero. Then, heat diffusion shown in equation (4.5) can be solved by CeNN model by setting parameters in equation (4.1) to

$$\hat{\mathbf{A}} = \kappa \cdot \begin{bmatrix} 0 & 1/h^2 & 0 \\ 1/h^2 & -4/h^2+1 & 1/h^2 \\ 0 & 1/h^2 & 0 \end{bmatrix}, \mathbf{A} = \mathbf{0}, \mathbf{B} = \mathbf{0}, z = 0. \tag{4.7}$$

Likewise, if the system of interest is described by a partial differential equation, we need to discretize the equation in space by using finite difference method to make them ODE [38, 154]. This discretization decides the linear part of state (feedback) template $\hat{\mathbf{A}}$. Note that in most physical systems, the output template $\mathbf{A}$ will be zero; it is used for applications like image processing or associative memory.

4.1.2 Mapping Nonlinear Systems

The proposed approach for mapping equations with nonlinear templates uses Taylor series based approximation for wide range of nonlinear functions (beyond polynomial). Assume that there is an additive nonlinear physical behavior observed on top of heat propagation in the system. The equation (4.5) converts to

$$\frac{\partial \varphi(x,y,t)}{\partial t} = \kappa \cdot \Delta\varphi(x,y,t) + l(\varphi(x,y,t)), \tag{4.8}$$

where $l(\cdot)$ is a continuous and infinitely differentiable nonlinear function. Using Taylor series expansion a nonlinear function can be approximated by a polynomial function at specific point 'p'. By applying the Taylor series of degree three on function $l(\cdot)$ at p, equation (4.8) becomes

$$\begin{aligned}\frac{\partial \varphi(x,y,t)}{\partial t} \approx\ & \kappa \cdot \Delta\varphi(x,y,t) + \{l(p) + l^{(1)}(p)\cdot(\varphi(x,y,t)-p) \\ & + l^{(2)}(p)\cdot(\varphi(x,y,t)-p)^2 + l^{(3)}(p)\cdot(\varphi(x,y,t)-p)^3\}.\end{aligned} \tag{4.9}$$

With simple derivation from equation (4.9), the parameters in equation (4.1) becomes

$$\begin{gathered}\hat{\mathbf{A}} = \kappa \cdot \begin{bmatrix} 0 & 1/h^2 & 0 \\ 1/h^2 & -4/h^2+1 & 1/h^2 \\ 0 & 1/h^2 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{A}=0, \mathbf{B}=0, z=c_3 \\ where \begin{cases} \alpha = c_0 + c_1\cdot\varphi(x,y,t) + c_2\cdot\varphi(x,y,t)^2 \\ c_0 = l^{(1)}(p) - 2p\cdot l^{(2)}(p) + 3p^2\cdot l^{(3)}(p) \\ c_1 = l^{(2)}(p) - 3p\cdot l^{(3)}(p) \\ c_2 = l^{(3)}(p) \\ c_3 = l(p) - p\cdot l^{(1)}(p) + p^2\cdot l^{(2)}(p) - p^3\cdot l^{(3)}(p) \end{cases}\end{gathered} \tag{4.10}$$

It implies that CeNN computing model requires real-time weight updates when a nonlinear function is involved in a given dynamic system. As shown in equation (4.10), the nonlinear template having α has to be recomputed with respect to the current cell state $\varphi(x,y,t)$. By providing c_0, c_1 and c_2 data to CeNN hardware platform, the state template $\hat{\mathbf{A}}$ can be updated with specialized hardware computing new α value. Note that $c_0 \sim c_3$ can be pre-computed with a given $l(\cdot)$ and stored as a look-up table (LUT) in off-chip memory.

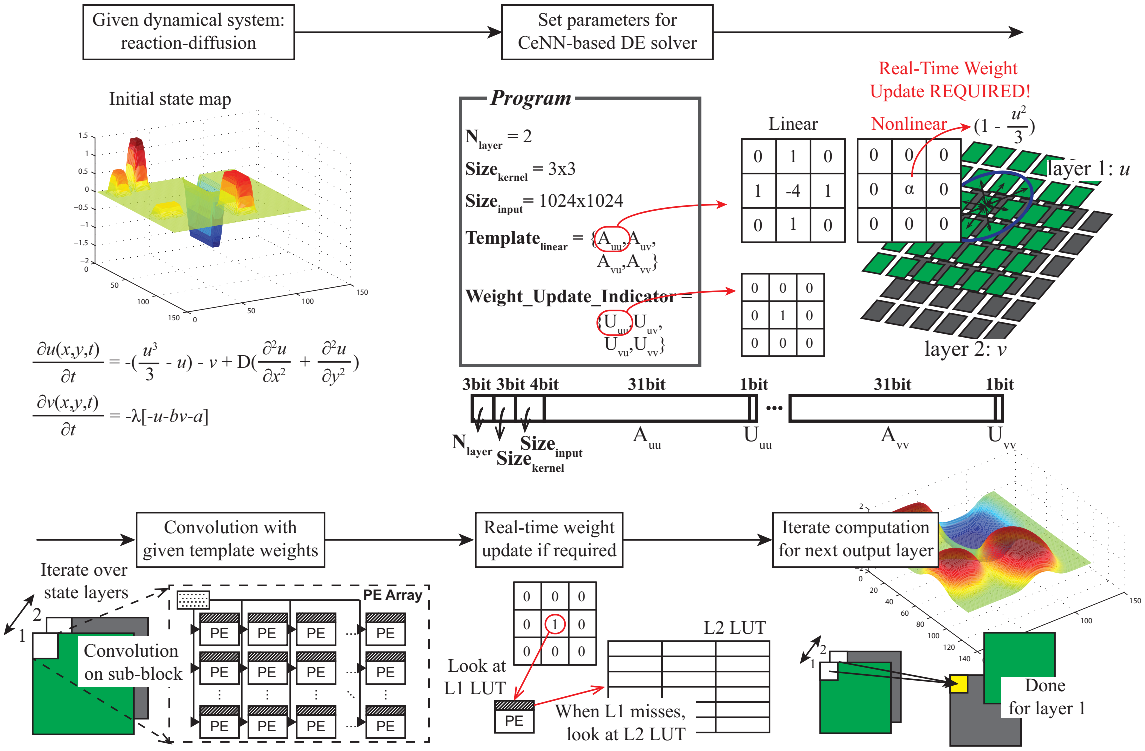

Given dynamical system: reaction-diffusion
Set parameters for CeNN-based DE solver
Initial state map
$\frac{\partial u(x,y,t)}{\partial t} = -(\frac{u^3}{3} - u) - v + D(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$
$\frac{\partial v(x,y,t)}{\partial t} = -\lambda[-u-bv-a]$
Program
$N_{layer} = 2$
$Size_{kernel} = 3x3$
$Size_{input} = 1024x1024$
$Template_{linear} = \{A_{uu}, A_{uv}, A_{vu}, A_{vv}\}$
$Weight\_Update\_Indicator = \{U_{uu}, U_{uv}, U_{vu}, U_{vv}\}$
Real-Time Weight Update REQUIRED!
Linear
Nonlinear
$(1 - \frac{u^2}{3})$
α
layer 1: u
layer 2: v
3bit 3bit 4bit
31bit
1bit
31bit
1bit
N_{layer}
$Size_{kernel}$
$Size_{input}$
A_{uu}
U_{uu}
A_{vv}
U_{vv}
Convolution with given template weights
Real-time weight update if required
Iterate computation for next output layer
Iterate over state layers
2
1
Convolution on sub-block
PE Array
PE
Look at L1 LUT
L2 LUT
When L1 misses, look at L2 LUT
Done for layer 1

Figure 4.3: The overall operation of the proposed CeNN-based differential equation (DE) solver. With a given dynamical system, template weights and other parameters are set to program the DE solver (binary bit stream is used to program). Then, a sub-block in each state map (each layer) is fed into PE array to perform convolution on them. When real-time weight update is required, it looks at LUTs at different levels.

The degree of the polynomial function determines the accuracy of approximation.

4.2 Operation of DE Solver

Before going into architecture details, the basic operation of the proposed CeNN-based differential equation (DE) solver is explained.

Set parameters: Fig. 4.3 illustrates the entire process of solving a coupled dynamical system described by the well-known reaction-diffusion (RD) equation. For a given equation, one can extract the required number of layers for the DE solver (refer to Section 4.1). As RD equation has two variables involved, $\mathbf{u} \in \mathbb{R}^2$ and $\mathbf{v} \in \mathbb{R}^2$, a two-layer CeNN model is used for simulating it. There is a nonlinear function involved in updating the activator

$\mathbf{u}$, but only linear term is present when updating the inhibitor $\mathbf{v}$. The nonlinear function is programmed to the DE solver by having state template $\hat{A}_{uu}$ (self-feedback template for layer $\mathbf{u}$) with *real-time update* as the function depends on $\mathbf{u}(t)$.

As such a nonlinear function can be any function depending on the system of interest, LUTs are utilized to store sampled function values and coefficients of Taylor series to compute function values between sampled points (refer to Section 4.3.1 for details). By looking up LUTs, it is possible to update the template weights at each cycle *if required*. Thus, we need an indicator to let PEs identify an equation that needs the real-time weight update. This update indicator flag is appended in the template data as shown in Fig. 4.3 (U_{uu} appended to A_{uu} making the total data bit-width 32bit).

Program DE solver: Accordingly, there are several parameters to be loaded to program the DE solver. They are input size ($\mathbf{Size_{input}}$), kernel size ($\mathbf{Size_{kernel}}$), number of layers ($\mathbf{N_{layer}}$), space-invariant and linear templates ($\mathbf{Template_{linear}}$), and binary indicator matrices for real-time weight update ($\mathbf{WUI}$). This can be programmed by using a bit stream pushed to the DE solver. The bit size representing each parameter bounds the maximum value of that parameter. For instance, if 3 bits are used to represent N_{layer}, then a coupled dynamical system with up to 8 layers (equivalently, 8 equations) can be solved. The size of kernel or input is programmed by providing the side length; to program 3×3 kernel, 3 is given in the bit sequence. For the input size, the side length is constrained to be the power of 2. Thus, the exponent is programmed in the bit sequence; $1010_{(2)}$ to program 1024×1024 input size.

Other than state template ($\hat{\mathbf{A}}$), feedforward template ($\mathbf{B}$) and offset (z) are also provided to the solver. They are appended to the bit sequence as well which adds another 148-Byte data at the end. For most cases, $\mathbf{B}$ and z do not require real-time update thus no weight update indicator is needed. In summary, a set of templates can be considered as a program for the DE solver to simulate a specific dynamical system.

Computation: After the DE solver is programmed, a PE array performs convolution on a sub-block in the state map of single variable (e.g. $\mathbf{u}$). At each clock cycle, multiplication between the weight and the state value within a convolution is performed. For 3×3 convolution with 8×8 PEs, it takes nine cycles to complete convolutions for 64 cells of one state variable. Then, the PE array moves to a sub-block at the same position but for the next variable (equivalently, layer) to be computed. As RD equation has two variables, convolution is done over 8×8 sub-blocks for two layers to get the updated sub-block of one layer. Then, the result is written back to off-chip memory for the computation in the next cycle.

Real-time weight update: During the multiplication, each PE checks WUI bit in the weight data and if the computation requires weight update it looks at a local (L1) LUT. If it misses, it looks at higher level (L2) LUT while setting PEs in idle mode. By having the intermediate level of shared LUT between off-chip memory and local LUT, we reduce the number of expensive accesses to DRAM. By retrieving the function value for the given state, PE becomes active again and completes the multiplication at next clock cycle. After the convolution for one output layer (update on one variable) is done, the computation moves to next layer to update next state variable. For RD equation, it needs to update two (output) variables; $\mathbf{u}$ and $\mathbf{v}$.

4.3 System Architecture

Fig. 4.4 shows the overall architecture of the proposed generic DE solver. The system architecture is composed of memory system, the real-time template update, and the processing engine cluster.

4.3.1 Real-Time Template Weight Update

A memory-centric approach using a hierarchy of look-up-tables (LUT) for real-time template update is proposed. An LUT is created to store the nonlinear function using Taylor-series expansion around different values as illustrated in Fig. 4.5. We store the finite number

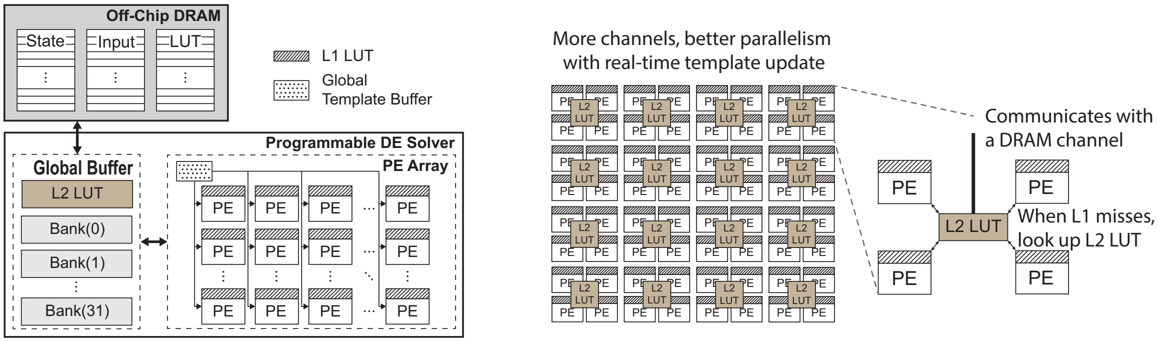
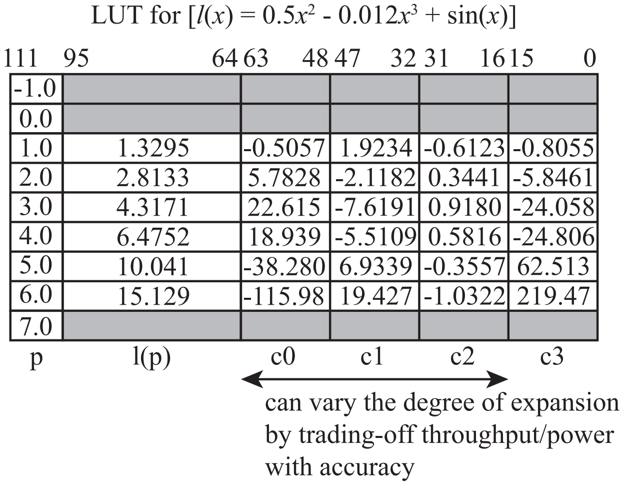


Figure 4.4: The overall architecture of CeNN-based DE solver. Look-up Tables (LUTs) allow nonlinear and real-time weight updates with complex functions.



LUT for [$l(x) = 0.5x^2 - 0.012x^3 + \sin(x)$]

p	l(p)	c0	c1	c2	c3
111	95 … 64	63 … 48	47 … 32	31 … 16	15 … 0
-1.0					
0.0					
1.0	1.3295	-0.5057	1.9234	-0.6123	-0.8055
2.0	2.8133	5.7828	-2.1182	0.3441	-5.8461
3.0	4.3171	22.615	-7.6191	0.9180	-24.058
4.0	6.4752	18.939	-5.5109	0.5816	-24.806
5.0	10.041	-38.280	6.9339	-0.3557	62.513
6.0	15.129	-115.98	19.427	-1.0322	219.47
7.0					

can vary the degree of expansion by trading-off throughput/power with accuracy

Figure 4.5: The data format stored in off-chip LUT of an examplary nonlinear function required for real-time weight update.

of exact values for $l(x)$ in equation (4.9), i.e. $l(p)$ at points 'p', expand $l(x)$ in Taylor series around 'p', and store the coefficients ($c_0 \sim c_3$). Therefore, stored data corresponding to a point p is a tuple D_{LUT} = $\{l(p), c_0, c_1, c_2, c_3 - l(p)\}$. Note, as $l(p)$ is stored, we store c_3 in equation (4.10) without the term $l(p)$.

A two-level cache hierarchy to manage the LUTs is utilized. The full LUTs are stored in the main memory. A part of the data is stored in a shared L2 LUT (one per memory channel of a chip), and multiple distributed L1 LUTs (one per processing engine). The performance of the solver depends on miss rates from the LUTs (mr_{L1} or mr_{L2}). As the LUT access is determined by the state of the CeNN cell, the miss rate depends on the size of on-chip LUTs and the distribution of states in the CeNN model.

As the number of LUT blocks is small in L1, the index is directly matched, multi-bit XNOR operation between higher 16 bits among 32bit of cell state and index in L1 LUT, to find the required value. For L2 LUT, as the size is much larger, direct matching is impossible. Therefore, a hash function utilizing modulo is being used as search index. The modulo by power-of-2 is used as the size of L2 LUT is 2^N and it is simple to design in hardware.

When there is a miss at L1 LUT, required data is copied (also fetched to PE at the same time) from L2 LUT if the data is present. The L1 LUT has write pointer which increments by one (cyclic) to provide write address whenever L1 LUT misses. When L2 LUT misses, expensive DRAM access happens thus we get multiple data points whenever it happens. In the proposed DE solver, it fetches eight data points whenever L2 LUT misses. For instance, if data for $p = 3.0$ was required in Fig. 4.5 and both on-chip LUTs missed then the solver fetches data from $p = 0.0$ to $p = 7.0$ in DRAM. When storing these data to L2 LUT, the same hash function is used to synchronize the data address with read operation.

After the function value $l(p)$ at point 'p' and coefficients (c_0, c_1, c_2, c_3) are loaded, each PE needs to decide whether to directly use the exact $l(p)$ or approximate function value using coefficients. This can be checked by looking at lower 16bit of state data as the off-chip LUT contains exact $l(p)$ with p represented by higher 16bit. Assume we have 32bit state (fixed-point) where the first half bits are integer and the rest are fractional. Then, 'p' will be an integer number for look-up index in LUTs. If the fractional part is non-zero, containing at least one high(1) bit in lower 16bit, we need to approximate a nonlinear template by computing α in equation (4.10). This is done by a specific hardware called, Template Update Module (TUM), attached to each PE (Fig. 4.6).

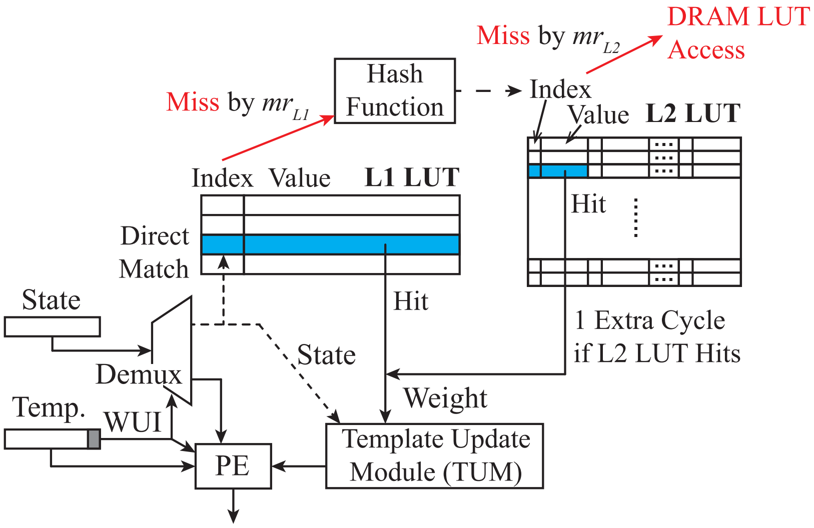


Figure 4.6: The operation of real-time weight update to compute state nonlinear template $\hat{\mathbf{A}}$.

4.3.2 Storage of States, Inputs, and Templates

The state x, input u, template weights ($\hat{\mathbf{A}}, \mathbf{B}$), and values of nonlinear functions are stored in the off-chip memory. There are 32 data banks in the proposed system and the half of them are dedicated to state x, the rest to input u of CeNN model. The data banks for each data type (x or u) are grouped into two (*primary* and *support*).

For each main memory channel, a global buffer consisting of data banks, and one level 2 (L2) LUT cache for nonlinear template update are present (Fig. 4.4). The off-chip memory communicates with the on-chip global buffer, and the global buffer pushes required data to a processing engine (PE) array which consists of a template buffer, L1 LUT cache for template updates, and PEs.

In the PE array, a template buffer is placed to store and broadcast template weight for the current convolution computation. The sequencing over this template decides which equation that PE array is currently solving (refer to Section 4.4). The data size filled in the template buffer is $N_{layer}^2 \cdot l_{kernel}^2$ for each template type (either feedback or feedforward); l_{kernel} is the length of a template. The finite state machine is used to address the template weight for each convolution operation.

Whenever a real-time weight update is required as a function of the current cell state, each PE searches for the value of that function or coefficients to perform series expansion from its local L1 LUT. If the required data is present, a PE computes its convolution without waiting. If not, the PE will wait for extra clock cycles to search the data from the connected L2 LUT.

4.3.3 Processing Engine Architecture

The detailed block diagram of PE array is shown in Fig. 4.7. There are N_{layer}^2 convolution templates in DE solver to handle and each convolution template has the size of $Size_{kernel}$. Each template weight is prefetched from shared template buffer by having two counters; one for layer indexing and the other for convolution indexing. Then, PEs start to compute convolution with the given weight and convolution index. By having proper data controller and muxes, the dedicated dataflow mode is used to move around the data within PEs as well as from data banks. The backup register in Fig. 4.7 helps PE promptly retrieve data when there is row change in template (refer to mode 2 in Fig. 4.10). The data prior to the horizontal shift is retrieved to move data to the upper PE of each PE (via x_V or u_V path). For mode 1 and mode 3, the data is moved to left within PE array; x_H or u_H path is for this purpose. This PE structure is simpler than the one presented in [155] as our PE only requires single-word register for state and input (no FIFO buffers needed).

In the following section, the proper dataflow method is explored suited for the proposed DE solver with real-time weight update. Also, the data movement in the memory system is explained in detail.

4.4 Dataflow in Proposed DE Solver

4.4.1 Exploration of Different Dataflow Schemes

The main operation of CeNN is the convolution between cell state (or external input) and programmed template kernel $\hat{\mathbf{A}}$ (or $\mathbf{B}$). Thanks to the improvement of convolutional neural

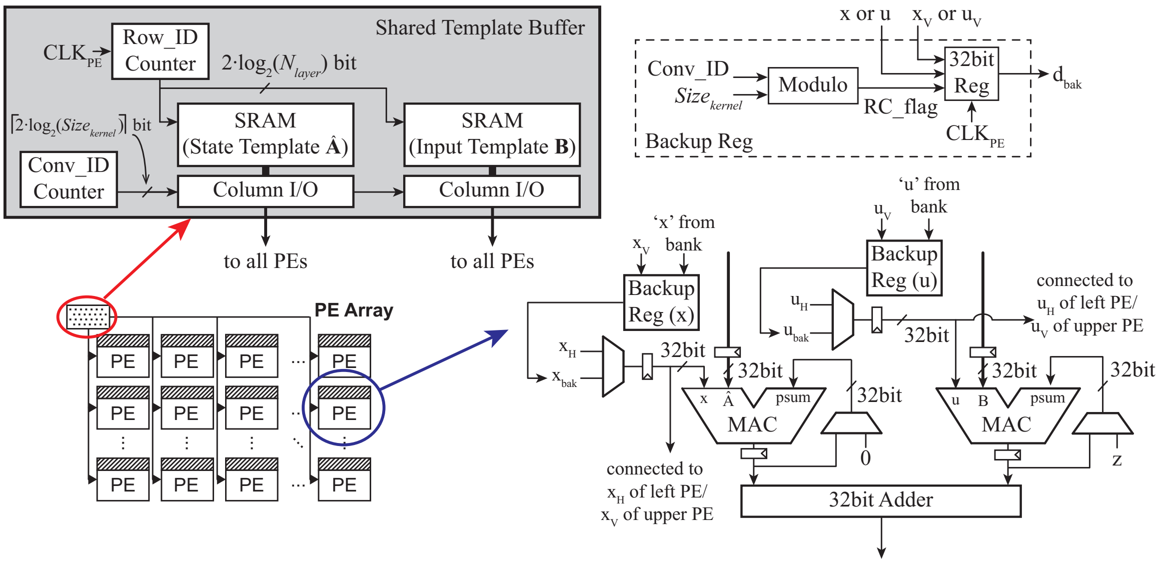


Figure 4.7: The detailed circuit block diagram of a global template buffer and a processing element.

network, various dataflow schemes exploiting data reuse are proposed and evaluated; optimized for the convolution computation [155, 156, 157, 158, 159, 160]. As CeNN-based DE solver requires the real-time weight update, the dataflow scheme that maximizes the throughput differs from the previous analyses.

In [160], various dataflow schemes are summarized and compared. The dataflow scheme can be grouped into four categories: no local reuse (NLR [156]), weight stationary (WS [157]), output stationary (OS [155, 158, 159]), and row stationary (RS [160]). According to the analysis in [160], RS dataflow showed the best energy-delay product by minimizing DRAM access and maximizing data reuse within memory units in lower hierarchy. However, one important thing to note is that other than OS dataflow different kernel (template) weights are given to PEs for convolution operation. In OS dataflow, one template weight is shared by all PEs in the processing array (refer to Fig. 4.8).

As shown in equation (4.4), some equations have a nonlinear function involved which is a function of the current state. Thus, a real-time weight update by accessing on-chip/off-chip LUTs is often needed for differential equation solving. By fetching coefficients from

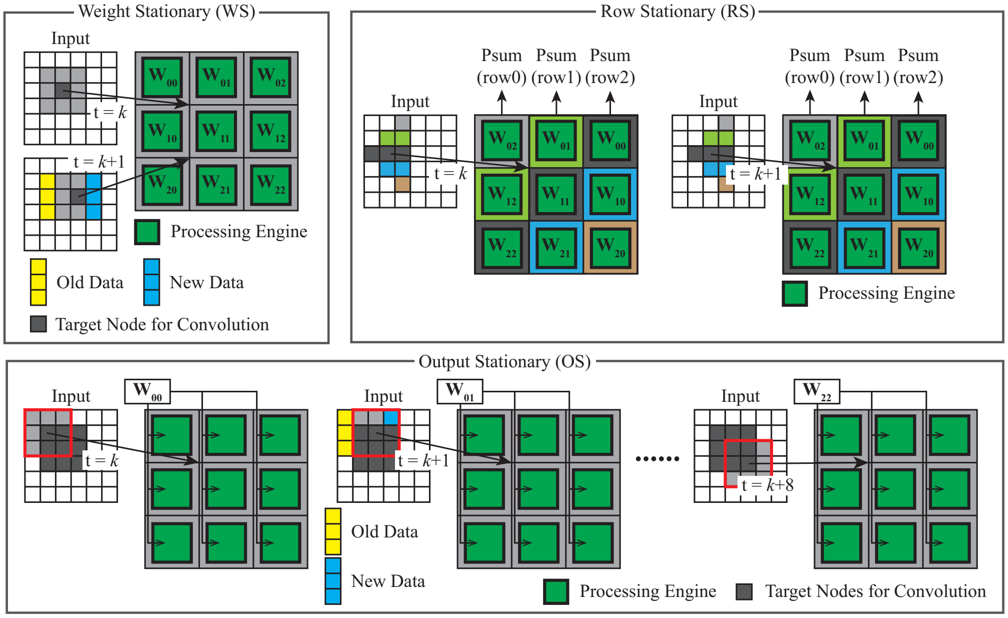


Figure 4.8: The comparison between different dataflow schemes for consecutive time steps with 3×3 template (kernel). Other than OS dataflow, entire template weights are used during the convolution operation.

the LUT, we are able to update the template with complex functions. Depending on the size of on-chip LUTs, miss rate will vary. Whenever there is a LUT miss, DRAM needs to be accessed which is expensive in terms of delay and energy consumption.

For dataflows other than OS dataflow, DRAM will be accessed at a clock cycle when at least one weight value in the template requires the update and on-chip LUT misses. Then, the number of DRAM access for the proposed DE solver becomes

$$\#DRAM_{access} = (mr_{L1} \cdot mr_{L2}) \cdot Size_{input} \cdot N(\mathbf{U}_{\mathbf{ll}^*} \neq \mathbf{0}), \tag{4.11}$$

where mr is the LUT miss rate and $N(\mathbf{U}_{\mathbf{ll}^*} \neq \mathbf{0})$ is the number of templates requires weight update. For the example shown in Fig. 4.3, it requires 100K DRAM accesses if $(mr_{L1} \cdot mr_{L2}) = 0.1$ and $N(\mathbf{U}_{\mathbf{ll}^*} \neq \mathbf{0}) = 1$.

By using OS dataflow, the number of DRAM access for real-time weight update is

$$\#DRAM_{access} = \frac{(mr_{L1} \cdot mr_{L2}) \cdot Size_{input} \cdot N(\mathbf{U_{ll^*}} \neq \mathbf{0})}{\#PEs}. \tag{4.12}$$

This is due to the weight sharing among all PEs, thus DRAM is accessed at that specific cycle when weight update is required. Then, the estimated number of DRAM access becomes 1.6K for the same scenario ($\#PEs\times$ less).

This analysis shows that OS dataflow is better than other dataflow schemes for the convolution where template needs to be updated over time. As CeNN state evolves over time, the advantage of utilizing OS dataflow piles up.

4.4.2 OS Dataflow in Proposed DE Solver

Before start computing, required data should be pushed from DRAM to the on-chip global buffer. There are 16 state banks and 16 input banks in the global buffer for 8×8 PEs in the system. Among 16 data banks, 8 banks are in the primary group and the remaining banks are in the support group. This is to utilize intra-PE data transfer to reduce data delivery energy from banks to local registers in the PE array.

As the process of input data (u_{kl}) prefetching is identical to state data (x_{kl}) prefetching, only the dataflow for states of entire layers in CeNN model is discussed. Each row block in a data bank stores $nPE_x = 8$ words where there is a [$nPE_y \times nPE_x = 8 \times 8$] PE array. The state map is divided into 8×8 sub-blocks where convolutions for those cells are handled by PE array at a time. Each bank in the primary group stores data for a row in the processing sub-block; bank ($k-1$) has data for the k^{th} row in each sub-block. The support group stores state data in an interleaved fashion as shown in Fig. 4.9. Assuming external memory as DDR3 (2 channels), each channel handles data prefetching for each bank type (primary or support).

After the state and the input data are prefetched to the global buffer, template weights

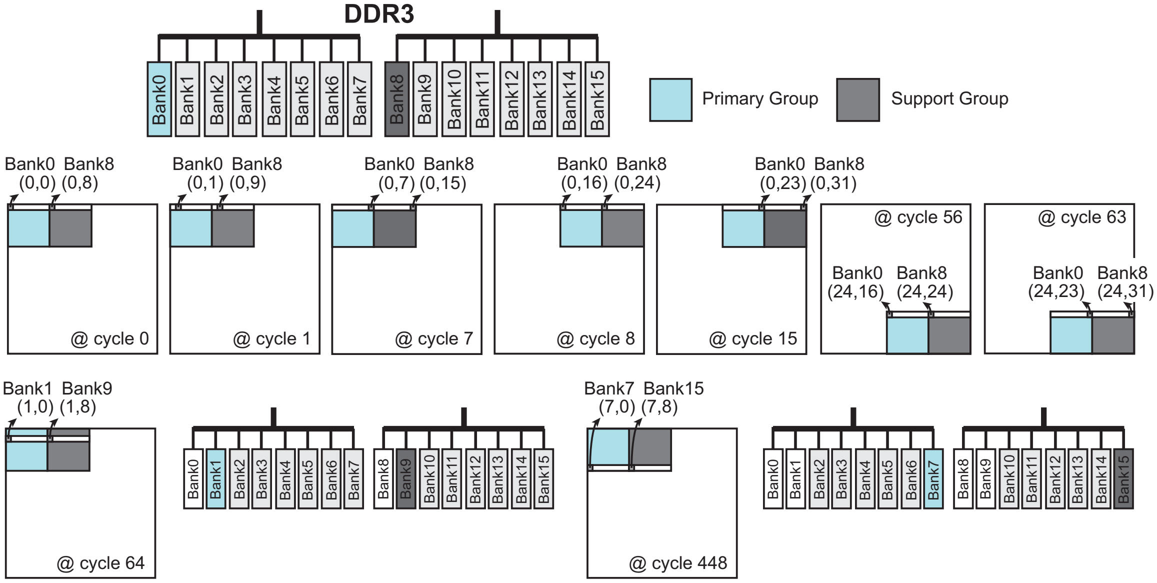


Figure 4.9: Data prefetching from off-chip DRAM to on-chip global buffer for 32×32 input. There are two bank groups (primary and support) to utilize intra-PE data transfer.

are pushed to the template buffer in PE array. The total number of weight data to be prefetched is $N_{layer}^2 \times l_{kernel}^2$ (36 for the example in Fig. 4.3). After all data is prepared for DE solver, the convolution operation begins to solve a differential equation. There are four modes during the convolution and they are shown in Fig. 4.10. A dataflow mode is selected depending on which convolution operation is currently being handled by PE array:

1. Mode 0: if $(conv\_id = 0)$
2. Mode 1: if $(0 < conv\_id < l_{kernel})$
3. Mode 2: if $(\lfloor \frac{conv\_id}{l_{kernel}} \rfloor > 0, mod(conv\_id, l_{kernel}) = 0)$
4. Mode 3: if $(\lfloor \frac{conv\_id}{l_{kernel}} \rfloor > 0, mod(conv\_id, l_{kernel}) > 0)$

The proper mode selection for the convolution with 3×3 template (kernel) is shown in Fig. 4.10.

As 64 PEs are present in the system (Fig. 4.4), 64 convolutions with 3×3 template is completed in 9 clock cycles (w.r.t. CLK_{PE}) when there is no weight update. If there is the

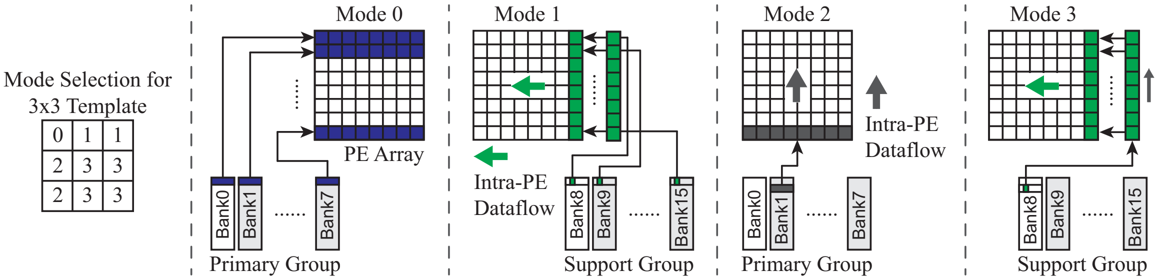


Figure 4.10: Dataflow modes from data banks to PE array during convolution operation. A proper mode selection for the convolution operation with 3×3 template is shown as an example.

need for real-time weight update for any element in the template, then the required clock cycles increase accordingly.

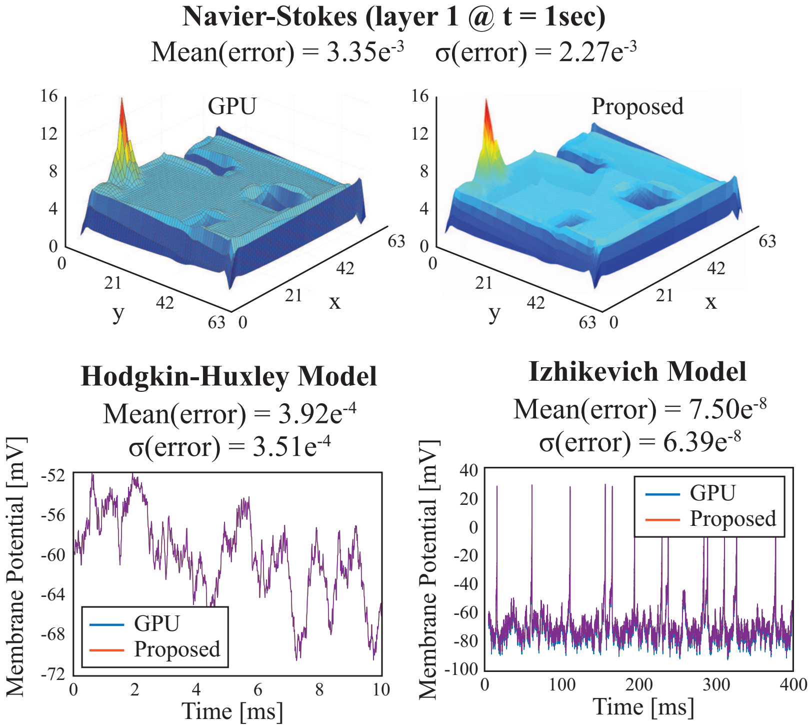
4.5 System Analysis of DE Solver

In this section, the performance and energy-efficiency of the proposed architecture with CeNN computing model are analyzed when solving wide classes of differential equations.

4.5.1 Benchmark Differential Equations

To evaluate the efficiency of the proposed CeNN-based DE solver, six differential equations describing various dynamical systems are considered. For the simplest benchmark, heat diffusion is selected as it is described by single PDE in which only linear template is present. The Navier-Stokes equation is used to simulate single PDE with nonlinear template. To simulate coupled systems, Fisher's and reaction-diffusion (RD) equations are selected. Among these, the RD equation can be used as another set of computing model, which is capable of simulating Turing machine [161]. Also, physical behaviors of cortical neurons can be modeled by coupled differential equations. They are called Hodgkin-Huxley (HH) model [162] and Izhikevich model [163]. These spiking models are candidates for a basic unit in neuromorphic computing engines. Thus, the CeNN-based DE solver not only finds the solution for a given differential equation, but it also accelerates the

	Heat	Fisher's	RD
Mean(error)	$4.09e^{-7}$	$8.01e^{-8}$	$6.48e^{-5}$
σ(error)	$1.03e^{-7}$	$3.51e^{-8}$	$5.55e^{-5}$



Navier-Stokes (layer 1 @ t = 1sec)

Mean(error) = $3.35e^{-3}$ σ(error) = $2.27e^{-3}$

Hodgkin-Huxley Model

Mean(error) = $3.92e^{-4}$

σ(error) = $3.51e^{-4}$

Izhikevich Model

Mean(error) = $7.50e^{-8}$

σ(error) = $6.39e^{-8}$

Figure 4.11: The accuracy comparison between GPU (64bit floating-point) and CeNN-based solver (32bit fixed-point).

simulation of other computing models for faster development.

When designing the DE solver, the bit-precision of a system has to be carefully determined. If the system is designed to compute with floating-point, the accuracy will be high but the energy-efficiency significantly degrades compared to fixed-point counterparts. For the power-constrained environment, such as mobile platforms or robotic controllers, the DE solver should perform fixed-point computation. Also, it becomes possible to run massive simulations with different conditions in parallel by utilizing mutiple (energy-efficient) DE solvers in finding a number of solutions to obtain near-optimal solution for a complex and large problem.

For six benchmark equations, we compared solutions with GPU (64bit floating-point) and the proposed DE solver (32bit fixed-point). The results on different dynamical systems

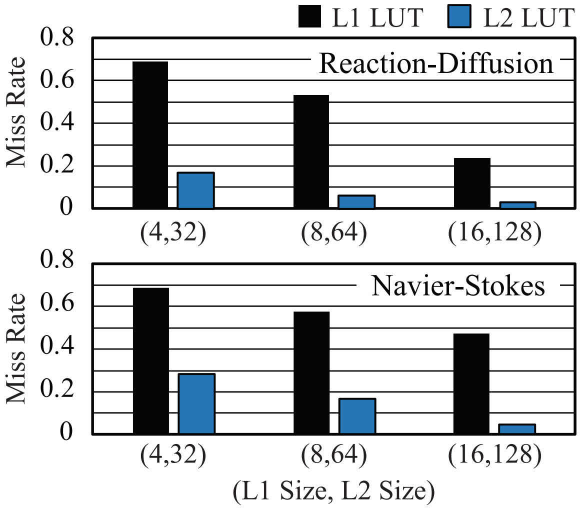


Figure 4.12: The miss rate depending on the size of on-chip LUTs for two different dynamical systems.

are summarized in Fig. 4.11. The average and standard deviation of absolute error for three benchmarks (Navier-Stokes, HH model, and Izhikevich model) are shown with the solution of each example. The error values for other differential equations are also provided as a table in Fig. 4.11. For spiking models, spikes were well-matched with the GPU simulation.

4.5.2 Miss Rate Analysis for Weight Update

As mentioned before, there exists trade-off between on-chip LUT size and miss rate. Several design choices are simulated and the results are shown in Fig. 4.12. They are simulated with two representative differential equations; one is reaction-diffusion and the other is Navier-Stokes equation. For both cases, miss rate of L1 LUT when there is only four blocks in the cache is about 0.7 which is quite high. The miss rate reduces by increasing the capacity, but with the support of larger L2 LUT the miss rate drops significantly (to 0.15-0.3) with one extra cycle. As the energy-efficiency is also crucial in some applications, the size of L1 and L2 LUT is selected to be 4 blocks and 32 blocks each for the following simulations.

4.5.3 Performance Comparison

According to the analysis in Section 4.5.1, the proposed DE solver runs with 32bit fixed-point. To simulate the performance, a cycle-level simulator is developed including all architecture details explained in Section 4.3. The simulator takes parameters in Fig. 4.3 with a configuration file (memory type, $Size_{kernel}$, $Size_{input}$, N_{layer}, $Template_{linear}$, and WUI). In the simulator, memory specification (bandwidth, # of channels, bus-width, latency), global buffer (# of banks, bank size, mr_{L2}), shared template buffer (buffer size), and PE array (# of PEs, latency, mr_{L1}) are parameterized. The miss rates, mr_{L1} and mr_{L2}, are extracted from Matlab simulation and fed to the simulator to consider actual state distribution of each benchmark.

The data prefetching from DRAM is performed in burst mode and burst length is assumed to be 8. Thus, the data are pushed to global buffer for eight consecutive cycles and data controller waits for t_{CCD} for another burst to happen. For the real-time weight update, when PE array gets required data from data banks, it checks WUI-bit in weight data and either computes or checks LUTs for nonlinear template. The clock cycle of PE array is $1/4$ of DRAM (or L2 LUT) clock as four PEs are connected to one L2 LUT.

By assuming DDR3 as an off-chip memory, the performance comparison between CPU, GPU, and the proposed DE solver is summarized in Fig. 4.13. The performance depends on the number of layers, the number of nonlinear weight updates, and the number of grid cells to be computed. About 46.48× (13.52×) performance improvement over CPU (GPU) on average is achieved by using the CeNN-based DE solver with DDR3. Two channels are assumed for DDR3 and this limits the performance as each channel connects to 8 L2 LUTs (Fig. 4.4). In the worst case, these 8 LUTs will miss and request data from DRAM forming a long request queue. As there are 16 L2 LUTs in our DE solver, the system throughput maximizes by having 16 memory channels with concurrent access.

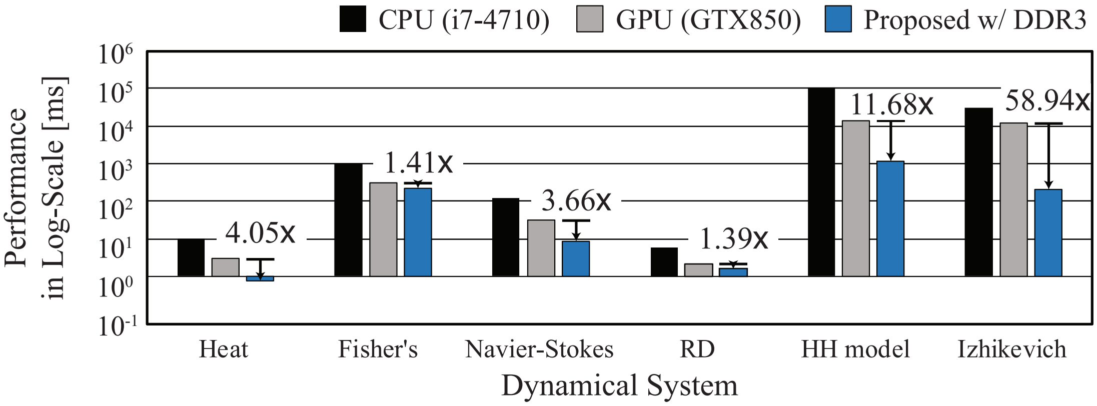


Figure 4.13: The performance comparison on six different benchmark differential equations. The speed-up using the proposed CeNN-based solver with DDR3 is shown compared to the performance using GPU (GTX 850).

4.5.4 Integration with High-bandwidth Memory

The memory-centric nature of the proposed architecture suggests that a higher bandwidth and concurrent accesses between compute and memory can enhance the system performance. As illustrated in Fig. 4.4, having more memory channels, so that each channel connects to individual L2 LUT, improves system throughput as the PE array can handle concurrent L1 LUT misses with better parallelism. A higher degree of parallelism also reduces the time required to push the states, inputs, and weights into the compute layer. Therefore, the integration of the CeNN based computing platforms with memory systems providing concurrent high bandwidth accesses, such as Hybrid Memory Cube (HMC) [164], is explored as well.

The effect of high-bandwidth memory for different benchmarks is evaluated (Fig. 4.14). Note that HMC provides two different interfaces: one externally connects to processors (HMC-EXT) and the other internally connects to processor-in-memory (HMC-INT). The simulation results show that integration with HMC significantly improves the performance compared to the one with GPU (Fig. 4.14). By using HMC-INT (or HMC-EXT), the performance improves by 23.67× (or 77.37×) on average. As the I/O clock frequency of HMC-EXT (10GHz) is much faster than that of HMC-INT (2.5GHz), the performance im-

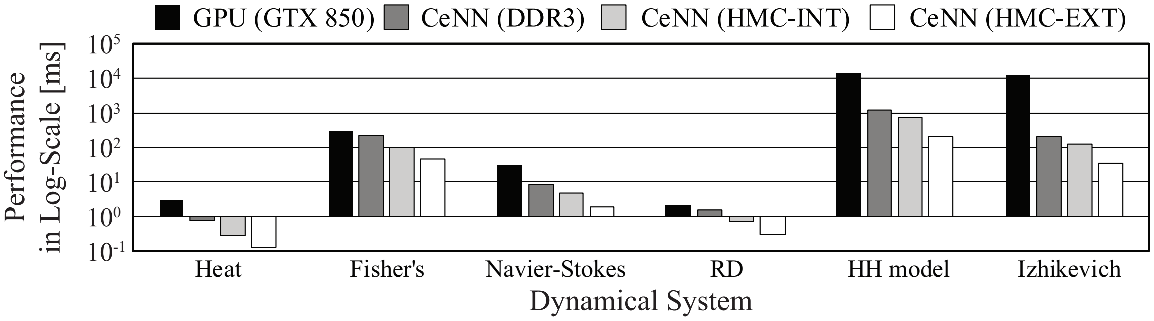


Figure 4.14: The performance improvement by using 3D memory stack with higher memory bandwidth.

provement is higher. However, this naturally leads to higher power consumption in the memory system and the processing array.

4.5.5 Power Consumption of DE Solver

To validate the power-efficiency of the proposed DE solver, the PE array is designed and synthesized using 15nm FinFET technology [165]. The power consumption of global buffer (L2 LUTs + data banks) is estimated by using PCACTI [166]. The power consumption is estimated assuming HMC-INT as an off-chip memory. As the maximum I/O clock frequency of HMC-INT is 2.5GHz, we synthesized the PE array to operate at 600MHz clock frequency. The L2 LUT runs at the same frequency as DRAM to maximize PE utilization at the worst case. The read operation of data banks or the shared template buffer runs at 600MHz which is synchronized to PE operation.

A PE array contains 64 PEs with 64 L1 LUTs. Each PE includes two backup registers, two MACs, one adder, and control logics (Fig. 4.7). This is denoted as ALU in Table 4.1. The template update module (TUM) is attached to ALU to form a PE. Table 4.1 shows the power estimation of PE array: 64 (PE-L1 LUT) pairs. The power consumption of 16 L2 LUTs or 32 banks is summarized in Table 4.2. Each L2 LUT has 16 lines of 64 Byte data (1KB); each line contains four look-up data. The global buffer has state data banks, input data banks, and shared template buffer where total size is about 2MB. As a result,

Table 4.1: The power consumption of modules in PE array having 64 (PEs-L1 LUT) pairs.

PE Array		Power (mW)	Area (mm^2)
PE	TUM	1.20	0.00308
	ALU	1.12	0.00287
	TUM+ALU	2.32	0.00594
PEs		148.48	0.380
L1 LUTs		51.20	0.0698

Table 4.2: The overall power consumption including PE array and global buffer (data banks + shared template buffer).

System	Power (mW)	Area (mm^2)
PE Array	199.68	0.450
L2 LUT	63.61	0.00627
Global Buffer	260.16	0.625
Total	523.45	1.082

the power consumption of the proposed DE solver becomes 523mW which is ∼100× less than GPU (50∼100W). Also, the area of the proposed DE solver becomes $\sim 1.1mm^2$.

4.6 Related Work

There has been research focus on developing hardware platforms for implementing CeNN model [44, 45, 46, 47, 167, 168]. They are categorized by underlying circuit type: analog/mixed-signal, FPGA, or fully digital platforms (Table 4.3). The analog CeNN hardware is fast and energy-efficient but it lacks accuracy (only 8bit precision), scalability, and programmability [45, 46, 47, 168]. There are emulated digital [167] and fully digital platforms [44] which give some level of programmability with higher accuracy. However, these works are only capable of dealing with linear time-invariant templates. Some emulated digital platforms provide specialized hardware units to handle nonlinear templates [154, 42]. Yet, they fail to provide architecture support to cover more general functions other than polynomial.

The proposed CeNN-based DE solver significantly improves the programmability compared to any previous platforms with proper architecture for real-time template update. This versatility is achieved along with high efficiency (∼103 GOPS/W). Accordingly, the

Table 4.3: The comparison of the proposed DE solver to previous hardware platforms for cellular nonlinear network.

	ACE16k [46]	**Q-Eye [168]**	**GAPU [167]**	**VAE [44]**	**This Work**
Type	Analog/ mixed-signal	Analog/ mixed-signal	FPGA	Digital	Digital
Technology	0.35um	0.18um	0.15um	0.13um	15nm
# PEs	16560	25344	1024	120	64
Power (W)	4.0	0.1	10.0	0.084	0.523
Area (mm^2)	92	25	-	4.5	1
Peak GOPS	330	0.1	1.3	22	54
GOPS/W	82.50	0.1	0.13	261.90	103.26
Nonlinear Weight Update					Yes

proposed DE solver will enable the efficient simulation of ‘computing with dynamical systems’.

4.7 Summary of the Chapter

This chapter presents the programmable differential equation (DE) solver based on CeNN computing model. The generic mapping of wide classes of dynamical systems to CeNN model is presented along with the architecture support to accelerate the computation. This solver allows more efficient way of model-based learning on various systems. The proposed DE solver includes a spatial PE array with LUTs for real-time weight update which enables handling nonlinear functions involved in differential equations. This real-time weight update is the important feature to make the solver truly programmable. The performance significantly improves by $\sim$24× to $\sim$77× on average with the use of high bandwidth memory. The energy efficiency improves by three to four orders of magnitude by using CeNN-based DE solver.

CHAPTER 5

POWER-AWARE DIGITAL FEEDFORWARD NEURAL NETWORK FOR LEARNING STATIC NONLINEAR SYSTEMS

An artificial neural network (ANN) can be considered as one of data-driven system learning methods. To train a neural network, large dataset is used to make the network accurately estimate certain behaviors of the underlying system. Among various types, a feedforward neural network (NN) is widely used due to its simplicity and ease of training [48]. An illustration of the operation and training of a feedforward NN is shown in Fig. 5.1. The network has only one directional pass with an input layer, one or more hidden layers, and an output layer. Recently, there have been active studies on digital NN hardware design to realize feedforward NN algorithms [14, 169]. As significantly large number of computations are required, the energy efficiency is a critical factor in ANN hardware design [111, 113].

This chapter presents a design method for the power-aware digital feedforward NN platform using approximate computation by integrating bit-precision control and imprecise hardware. To achieve this goal, a digital feedforward NN hardware having both accurate and approximate processing engines (PEs) is designed. An approximate PE employs recent developments in approximate multipliers to reduce energy [113, 170]. First, the proposed approach determines a set of approximate synapses in NN that have least impact on output quality using a greedy algorithm. The algorithm identifies them from the error sensitivity computed during the training phase (Fig. 5.1(b)). This error sensitivity provides the information of how much output error is coming from small perturbation at each synaptic weights in the network. Next, the selected approximate synapses are processed with reduced bit-precision and/or using approximate PEs to reduce energy dissipation. The precision control limits the bit-width of operands by forcing zero to some LSBs during run-time (software approach). Utilizing approximate PEs, however, is a hardware design

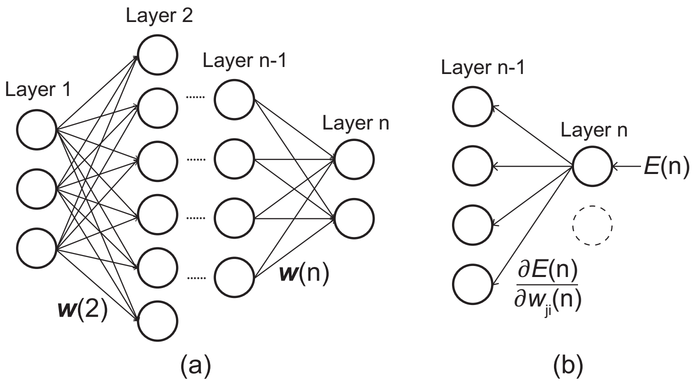


Figure 5.1: (a) The operation of feedforward neural network. (b) The backpropagation training algorithm computing the error sensitivity ($\partial E/(\partial w_{ji})$).

approach. When accurate PEs are replaced by approximate PEs, we sacrifice accuracy for power reduction.

5.1 Background

5.1.1 Feedforward Neural Network

A neural network with acyclic connection from input layer to output layer is defined as a feedforward NN [48, 49, 50]. A feedforward NN is common structure thanks to its simplicity compared to a recurrent neural network. In each layer, a neuron has multiple input connections from some (or all) of neurons in the previous layer. These connections are called synaptic weights which determines the functionality of a given NN. In the feedforward NN, the state of a neuron in layer k is defined as

$$x_j(k) = \varphi(\sum_i w_{ji}(k-1) \cdot x_i(k-1)), \tag{5.1}$$

where $x_j(k)$ is the state of j^{th} neuron in layer k, $w_{ji}(k-1)$ is the synaptic weight connecting between i^{th} neuron in layer $(k-1)$ to j^{th} neuron in layer k and $\varphi(\cdot)$ is the activation function. The computation of (5.1) from input layer to output layer is called a feedforward phase (Fig. 5.1(a)).

To train the feedforward NN, each synaptic weight w_{ji} is updated to minimize the error in the output layer generally with a given desired output (supervised learning). To update the weight, backpropagation algorithm is used that computes the sensitivity of each weight to the output error ($\partial E/(\partial w_{ji})$) from output layer back to input layer (Fig. 5.1(b)). Using the error sensitivity (gradient), each weight is updated as follows:

$$w_{ji}(n+1) = w_{ji}(n) - \eta \cdot \frac{\partial E(n)}{\partial w_{ji}(n)}, \tag{5.2}$$

where η is learning rate and $E(n)$ is the error function in layer n. According to (5.2), synaptic weights are updated based on the gradient and the learning rate until the output error is less than predetermined threshold.

5.1.2 Prior Work on Neural Network Hardware

Due to high parallelism in NN algorithms, there have been several works on design of a digital NN hardware. Most of previous works are based on the field programmable gate array (FPGA) [14, 169] and few of them discuss a system with an external memory [14]. In feedforward NN described in (5.1), multiplication and addition are dominant mathematical operations. Thus, the arithmetic unit is the key factor in designing low-power NN hardware. Multilayer perceptron with 288 processing engines using FPGA is demonstrated [169]. Although it achieves high parallelism with a large number of processing engines, there is no power-aware NN hardware design approach. Even though a full digital NN system with 8 processing engines (FPGA) and external memory controller (ARM core) has been implemented in [14], it also does not provide any power saving methods.

Recently, approximate computation based neuromorphic hardware design methods have been proposed [111, 113]. The most common way to approximate computation is reducing the precision bit-widths of operands [111]. By carefully forcing some LSBs to zero, effective power saving can be achieved with slight degradation in output quality. However, the proposed algorithm in [111] identifies approximate neurons instead of synapses which may cause error as discussed in Section 5.3. Another method to approximate NN hardware is by implementing approximate multipliers as processing engines [113]. An approximate multiplier is designed with iterative logarithmic multiplier having computation error less than 1%. However, having only approximate PEs is limited in saving power while maintaining the output quality as will be discussed in Section 5.3. Moreover, the prior works mainly focused on the PEs while evaluating approximate computation, neglecting the full-chip architecture, memory controller, and external DRAM.

5.2 Power Analysis of Processing Engines

In this section, we first analyze the power dissipation of the basic processing engine (PE), a multiply-accumulate unit. The power consumption is critical in implementing NN hardware on a mobile platform. The power consumption can be reduced by selectively programming some LSBs to zero (*software approach*) [111]. Since the switching activity significantly reduces for those bits, precision control is a popular technique used to achieve low-power operation of digital circuits. In addition, we can further reduce power consumption by replacing accurate multipliers in some PEs with approximate multipliers in design time (*hardware approach*).

Accurate PE: First, we estimate the power consumption of an accurate multiplier designed and synthesized in 130nm CMOS technology. Since the precision control is effective power reduction method, power consumptions at seven different bit-precisions are simulated with input activity of 0.3 (black bar in Fig. 5.2). As shown in Fig. 5.2, reducing the bit-width from 32bit to 8bit, the power can be saved by $\sim$53%. The cumulative error distributions due to quantization at different bit-precisions are shown in Fig. 5.3(a). The reference bit-width for the error computation is 32bit. As a notation, $Q_{1,7,8}$ represents 1 sign bit, 7 integer bits and 8 fractional bits (fixed-point representation is used). The probability of quantization error increases as the reduction in bit-width increases as expected. As

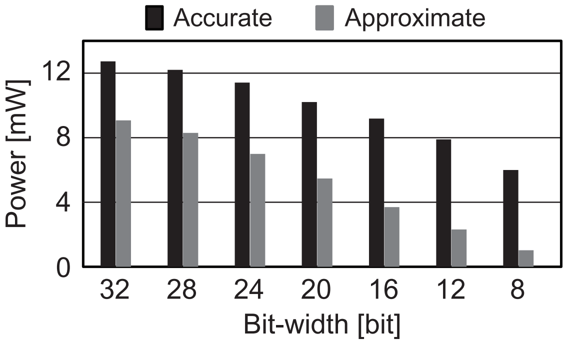


Figure 5.2: The power dissipation of the accurate multiplier (black) and the approximate multiplier (gray).

the bit-width reduces to 12bit or 8bit, most of errors are quite large (between $10^1 \sim 10^2$) due to the overflow error at integer part. Depending on the power constraint, we can select the bit-precision to be used for low-power operations.

Approximate PE: To save more power coupled with precision control, the approximate multiplier with partial error recovery [170] is utilized for a selected NN computations. This approximate multiplier preprocesses its inputs to let multiplicand have mostly 1's and let multiplier have mostly 0's to minimize the probability of error. The power saving is obtained by simplifying the adder cell for the computation of partial products. A simple error recovery circuit to reduce approximation error is also proposed to recover predetermined number of MSBs [170]. The approximate multiplier is also designed and synthesized using the 130nm CMOS. The power consumptions at the same bit-precisions are also simulated with the input activity of 0.3. As in Fig. 5.2 (gray bar), the power consumption is reduced by 29$\%$ by using the approximate multiplier compared to that using the accurate multiplier for 32bit precision. In 8bit case (zeros are forced at 24 LSBs), the power saving becomes 83$\%$ compared to the power consumption using the accurate multiplier. This additional power saving can be realized by carefully selecting some NN computations as approximate which is explained in Section 5.3.

The amount of additional error introduced by the use of approximate multiplier is analyzed when compared to the computation with all accurate multiplier at each bit-precision

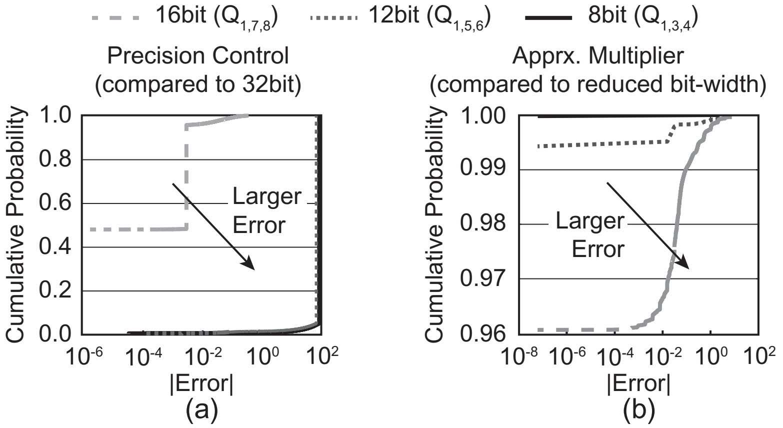


Figure 5.3: The cumulative probability of (a) quantization error due to precision control and (b) the additional error induced by using an approximate multiplier with error correction of 20 MSBs (x-axis is logarithmic scale).

(16bit, 12bit and 8bit) with error recovery of 20 MSBs (Fig. 5.3(b)). Since most of errors in 20 MSBs are recovered, less than 3% of errors are larger than 10^{-2}. The interesting observation is that as we force zeros at some LSBs (precision control) the additional error introduced by using approximate multiplier (over the precision control) become smaller. This observation is favorable for aggressive power saving with an integrated approach of precision control and approximate PE. This error distribution is used when simulating feed-forward NN with approximate PEs involved.

5.3 Approximate Synapses Selection

5.3.1 Design Methodology

A) Approximate Synapses

The underlying idea of deciding how to approximate is utilizing the error sensitivity ($\partial E/(\partial w_{ji})$ in (5.2)), similar to [111]. This is easy to implement since the backpropagation computes the sensitivity factor during the training phase. We are merely using the existing values for the approximation process. This sensitivity vector is sorted in ascending order to utilize it.

In [111], neurons are approximated by looking at the average gradient of error with re-

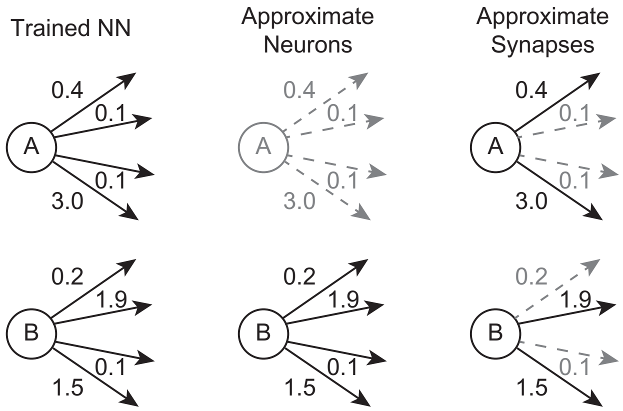


Figure 5.4: Diagram illustrating the advantage of approximating synapses over approximating neurons on a given trained feedforward NN.

spect to weights connected to a neuron. However, due to the averaging effect, some weight connections (synapses) that have large impact on output may be undesirably approximated. This is explained with a simple diagram having two neurons in Fig. 5.4. For a given trained NN, error sensitivity of each synapse is precomputed with a training data. These values are then averaged for each neuron; neuron A has average error sensitivity of 0.9 while neuron B has 0.93. Then, synapses connected to neuron A will set as approximate and use reduced bit-precisions. The problem here is that it ignores the synaptic weight 3.0 in A which is the largest among all synapses of two neurons. The proposed algorithm identifies approximate synapses instead of neurons. Using the identical example (Fig. 5.4), the same number of synapses are approximated in ascending order from the lowest synaptic weight, 0.1. As expected, synapses with low error sensitivity are selected. This fine-grained scheme is effective in reducing power consumption of the digital feedforward NN hardware.

B) Approximation by Precision Control

As a test application, a well-known hand-written digit recognition, MNIST, is selected using 28-by-28 sized input images (784 input neurons) [171]. To avoid simulation results from an overfitted feedforward NN, five different NN topologies are simulated with variable number of hidden layers and neurons in each layer. By comparing the corresponding recognition accuracy of each NN, feedforward NN with two hidden layers is selected. There are 144 neurons (12-by-12) in hidden layer 1 and 64 neurons (8-by-8) in hidden layer 2. Since MNIST classifies 10 different single digits, the number of output neurons is 10.

The objective of approximating synapses (for now, reducing bit-precision) is to lower power consumption of PEs. Using the power estimates from hardware simulation in Section 5.2, it is possible to obtain the ratio of the number of approximate synapses to all synapses (N) to achieve a given power constraint. In general, we can have m_{prec} bit-precisions available, then total power consumption (P_{tot}) of PEs become

$$(N - \sum_{k=1}^{m_{prec}-1} n_k) \cdot P_{m_{prec}} + \sum_{k=1}^{m_{prec}-1} (n_k \cdot P_k), \tag{5.3}$$

where P_k is the power consumption of a PE at k^{th} bit-precision ($P_{m_{prec}}$ is power at 32bit). However, since there are too many possible solution sets with a single equation, it is impossible to decide the best solution that satisfies the power constraint while achieving good output quality.

To simplify the analysis, we first consider there are two possible bit-precisions: 32bit (full precision) and 12bit (reduced precision). Operating PEs at each bit-precision results in P_{32bit} and P_{12bit} accordingly. The target power (average power constraint of PEs) is denoted as P_{target}. Then, the ratio $r_{12bit} = n_{12bit}/N$ has to satisfy

$$\begin{aligned} (N - n_{12bit}) \cdot P_{32bit} + n_{12bit} \cdot P_{12bit} \leq N \cdot P_{target} \\ r_{12bit} = \frac{n_{12bit}}{N} \leq (\frac{P_{target} - P_{32bit}}{P_{12bit} - P_{32bit}}). \end{aligned} \tag{5.4}$$

In the first equation of 5.4, the right-hand side represents the total power constraint in computations involving all N synapses. The left-hand side divides the power consumption to accurate (32bit) and approximate synapse computation (12bit). Thus, r_{12bit} has to be set satisfying the second equation in (5.4).

Likewise, the percentage of approximate synapses to satisfy different power constraints

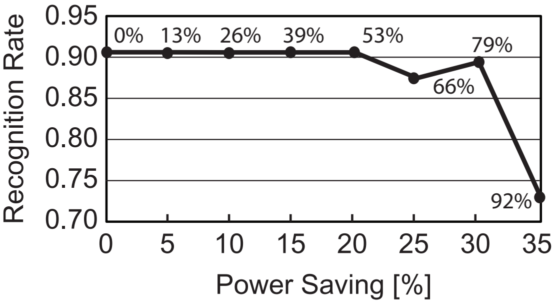


Figure 5.5: Recognition rate from MNIST dataset when approximation of synapses (*only precision control*) is allowed to meet a set of given power constraints. The percentage of approximate synapses to meet each power constraint is provided in the plot as well. The analysis assumes only two precision levels.

is computed (indicated in Fig. 5.5). After sorting the absolute value of error sensitivity (also called gradient) in ascending order, we simply select r_{12bit} of synapses from the sorted list to meet a given power constraint. The recognition rate of MNIST at a given power saving percentage (compared to the power consumption with full precision; 32bit) is obtained. As observed in Fig. 5.5, the recognition accuracy begin to fall when power saving exceeds 20%. This means that approximating 40∼50% of synapses can be considered acceptable but approximating beyond that in MNIST application is not recommended since it sharply degrades recognition accuracy. These approximate synapses are, so far, realized by programming reduced bit-precisions (software approach) achieving some level of power savings (Fig. 5.5).

C) Power Saving Using Approximate Multiplier

In the previous subsections, how to set approximate synapses was explained by utilizing the error sensitivity obtained during the backpropagation algorithm. If the feedforward NN hardware needs to be employed in more power constrained platforms, we can replace some PEs with approximate PEs in design time (hardware approach). As shown in Fig. 5.2, an approximate PE reduces power by 51% on average over various bit-precisions compared to an accurate PE. This additional power saving would be very attractive only if utilizing approximate hardware on the feedforward NN platform does not degrade quality much.

Fortunately, when the bit-precision is reduced, the additional errors introduced by approximate PEs become non-observable (Fig. 5.3(b)). This is because some LSBs are forced to zeros by precision control and the approximate PE does not produce any error at bits that are zeros. This is favorable as explained when approximate PE operation is coupled with reduced precision. However, the number of approximate PEs is predetermined in design time. Then, some computations involving accurate synapses may be fed into approximate PEs. The optimal number of approximate PEs cannot be determined in advance since the ratio of approximate synapses is different depending on the application that the NN hardware is running.

As mentioned earlier, depending on a given P_{target}, some synapses, which are not set as approximate, need to be fed into approximate PEs to ensure full parallelism. The following condition needs to be satisfied to meet P_{target} in general with multiple bit-precisions n_{prec}:

$$\begin{aligned} P_{target} \geq \sum_{k=1}^{m-1} r_k \cdot P_{app(k)} + \sum_{k=m+1}^{n_{prec}} r_k \cdot P_{acc(k)} \\ + min(1-\gamma, 1-x) \cdot P_{acc(n_{prec})} \\ + max(x-\gamma, 0) \cdot P_{acc(m)} \\ + max(\gamma - x, 0) \cdot P_{app(n_{prec})} \\ + min(x, \gamma) \cdot P_{app(m)}, \end{aligned} \tag{5.5}$$

where r_k is the ratio of k^{th} bit-precision among all, $x = \sum_{k=1}^{m} r_k$ represents the ratio of bit-precisions that entirely/partially fed into approximate PEs, γ is the ratio of approximate PEs to all PEs (0.4 in the example) and $P_{acc(k)}$ ($P_{app(k)}$) is the power consumption of an accurate (approximate) PE operating at k^{th} bit-precision. Note that γ is predetermined when designing the NN hardware with approximate PEs and x is the value to be determined depending on P_{target}. Similar to the case in Section 5.3.1, various design choices make it impossible to decide one solution from (5.5).

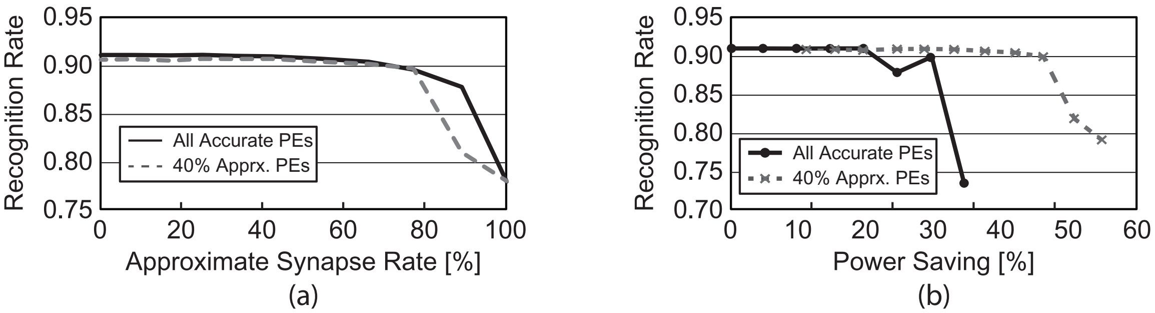


Figure 5.6: MNIST recognition rate with accurate (100% accurate) and approximate (40% approximate, 60% accuarte) PEs (a) at different rates of approximate synapses and (b) at different power constraints.

For simple understanding, it is assumed there are two different precisions (32bit and 12bit). It is also assumed that the proposed feedforward NN hardware consists of 40% of approximate PEs and 60% of accurate PEs. To compare the accuracy with 40% approximate PEs (60% accurate PEs) to that with 100% accurate PEs, r_{12bit} is varied to have the same number of approximate synapses. The resulting recognition rate is shown in Fig. 5.6(a). There is little accuracy degradation by using 40% approximate PEs compared to the accuracy having 100% accurate PEs in the system until when 77% of synapses are approximated. Note that it consumes less power with 40% approximate PEs when the number of approximate synapses is same (Fig. 5.6(b)). This means that with approximate PEs we can save more power by approximating the same number of synapses.

5.3.2 Greedy Algorithm for Low-Power Design

To find the minimum power at a given quality constraint when various bit-precisions are available, a fine-grained greedy algorithm is proposed that iteratively selects the lower power consuming bit-precision at a time until it satisfies the target quality. This greedy algorithm is described in Algorithm 1 and illustrated in Fig. 5.7(a). As proposed in Section 5.3.1, the precomputed error sensitivity (*Grad*) is utilized by sorting the list from low to high (this is important to maximize the output quality while saving significant power).

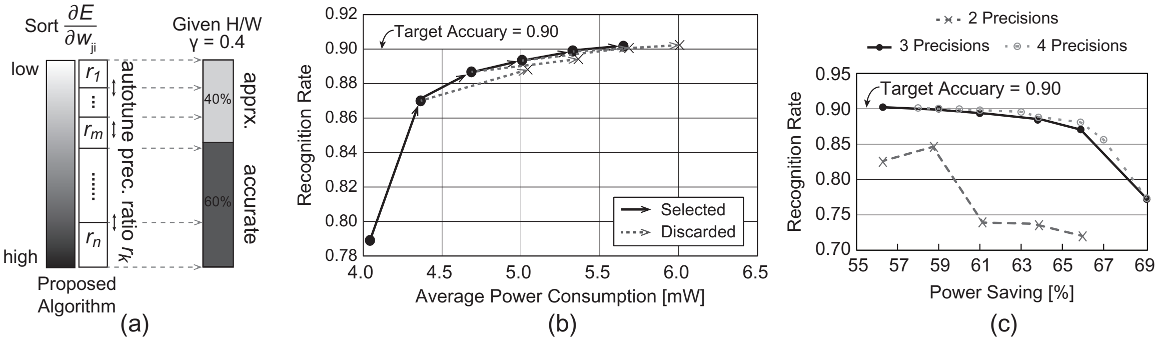


Figure 5.7: (a) Overview of the proposed greedy algorithm with $\gamma = 0.4$. (b) Experimental result on quality-aware low-power design methodology. This method dynamically selects (solid line) precision bit-widths which increase accuracy with less power increase. (c) Recognition rate comparison between two, three, and four different bit-precisions at varying power constraints.

Briefly, the algorithm starts from the least quality with the lowest possible power and increase bit-precision in a way that results in less increase in power consumption (Fig. 5.7(b)).

First, we initialize the precision ratio set ($\mathbf{R_{prec}}$) so that the NN hardware has the minimum precisions (all 8bits); equivalently the minimum power. $\mathbf{R_{prec}} = \{1.0, 0.0, , 0.0\}$ means that the system has only 8bit precisions for all NN computations. Then, initial feedforward phase using $\mathbf{R_{prec}}$ is performed to compute the current quality and power estimates on testing dataset ($\mathbf{D_{test}}$). $\mathbf{P_{est}}$ is a set of power estimates of both accurate and approximate PE obtained by hardware simulation as in Section 5.2. After the initialization, the algorithm iterates until the target quality is met. At each iteration, two different bit-precision ratio sets ($\mathbf{R_{trial1}}, \mathbf{R_{trial2}}$) are generated. Using these two trial sets, corresponding quality and power estimates are computed by performing feedforward phase. After a feedforward phase, each score value is computed (the weighted sum of normalized quality increase and the negative of normalized power increase). By comparing two scores, the algorithm selects the bit-precision ratio with better score. When it meets the target quality, the power consumption with output precision set is obtained. After $\mathbf{R_{prec}}$ is decided, we identify synapses in ordered *Grad* whether to fed into accurate or approximate PE by adding one dirty bit to indicate this.

Algorithm 1 Power-aware feedforward NN design methodology

```
Input:
    Grad: Computed error sensitivity
    $\mathbf{P_{est}}$: Estimated power
    $\gamma$: Ratio of approximate PEs
    $Q_{const}$: Quality constraint
    $\mathbf{D_{test}}$: Test dataset
Output:
    $P_{min}$: Minimum power
    $\mathbf{R_{prec}}$: Precision ratio set
Begin
Initialize $\mathbf{R_{prec}} \leftarrow \{1.0, 0.0, ..., 0.0\}$
$[Q, P] \leftarrow apprx\_feedforward(\gamma, \mathbf{R_{prec}}, \mathbf{D_{test}}, \mathbf{P_{est}})$
while $Q < Q_{const}$ do
    $[\mathbf{R_{trial1}}, \mathbf{R_{trial2}}] \leftarrow generate\_nextRprec(\mathbf{R_{prec}})$
    $[Q_{trial1}, P_{trial1}] \leftarrow apprx\_feedforward(\gamma, \mathbf{R_{trial1}}, \mathbf{D_{test}}, \mathbf{P_{est}})$
    $[Q_{trial2}, P_{trial2}] \leftarrow apprx\_feedforward(\gamma, \mathbf{R_{trial2}}, \mathbf{D_{test}}, \mathbf{P_{est}})$
    $Score1 \leftarrow compute\_score(Q_{trial1}, P_{trial1}, Q, P)$
    $Score2 \leftarrow compute\_score(Q_{trial2}, P_{trial2}, Q, P)$
    if $Score1 > Score2$ then
        $[\mathbf{R_{prec}}, Q, P] \leftarrow update(\mathbf{R_{trial1}}, Q_{trial1}, P_{trial1})$
    else
        $[\mathbf{R_{prec}}, Q, P] \leftarrow update(\mathbf{R_{trial2}}, Q_{trial2}, P_{trial2})$
    end if
end while
$P_{min} \leftarrow P$
return $P_{min}, \mathbf{R_{prec}}$
End
```

The overall transient process of Algorithm 1 is shown in Fig. 5.7(b). Here, three different bit-precisions are used; 32bit, 16bit and 8bit with a quality constraint of 0.9 for MNIST application. Also, 40% of PEs are replaced by approximate PEs. The number of bit precisions can be larger than this for finer grained search. As a result, 0.9017 recognition rate is achieved with 56% power saving compared to the result with all 32bit computation using 100% accurate PEs (0.9053 recognition rate is achieved).

In addition, the results of the proposed algorithm with two (12 and 32bit), three (8, 16 and 32bit) and four (8, 12, 16 and 32bit) bit-precisions are compared in Fig. 5.7(c). For two bit-precisions, equation (5.5) is used to compute the percentage of synapses (r_{12bit}) to be precision controlled. Using the proposed greedy algorithm, the precision ratio sets for three and four bit-precision cases are obtained. Going from two to three bit-precisions increases the accuracy significantly with the same power saving; however, improvement is marginal from three to four bit-precisions, and saturates beyond that.

In Fig. 5.8, the efficiency of the proposed algorithm is shown by comparing the resulting accuracy and the amount of power saving with only precision control (software approach), approximate neurons with only precision control ([111] with 3-level bit-precision) or only approximate PEs (hardware approach). For the software approach, 50% of synapses are computed with 16bit and the rest with 8bit (obtained by the proposed algorithm with 100% accurate PEs). For [111], 62.5% of neurons are selected as approximate and precision is reduced to 16bit (the rest with 32bit) by using their algorithm. For the hardware approach, 100% approximate PEs are used for the computation. The proposed greedy algorithm effectively reduces power consumption (56%) with negligible quality degradation with a given digital feedforward NN hardware.

5.4 Full System Power Analysis

The full system of digital feedforward NN is implemented in 130nm CMOS based on the system architecture shown in Fig. 5.9. The external DRAM we assumed is 8-channel 3-D

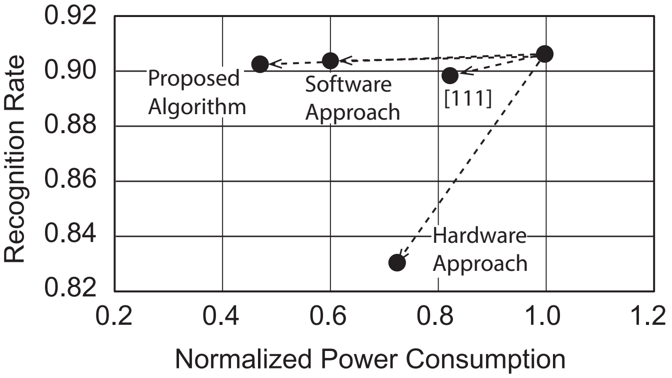


Figure 5.8: Comparison of the proposed algorithm to just doing software (reduced bit-precisions) or hardware (approximate PEs) approach. Proposed approach couples both reduced bit-preciosn and approximate PE.

wide I/O with 2KB page size using 64bit bus width. The number of PEs and SRAMs are obtained so as to guarantee the full utilization of PEs with a given memory specification. The 192 PEs are synthesized to operate at 50MHz while 24 SRAMs (4KB each) connected to PEs operate at 400MHz in 1.2V. One of the most important parts in the system is the memory controller, which delivers the data from cache to PE or from DRAM to cache based on the network topology. We should note that all data sequences from memory is pre-determined; therefore PEs do not have to request data to memory, but data is delivered from memory to PEs in order. In addition, memory address is not random, but is organized thus data will be fetched from memory row by row (implemented by a simple counter). Approximate H/W controller is placed to identify computations that should go into the approximate PEs. One dirty bit is used for the decision.

The full system of digital feedforward NN is implemented in 130nm CMOS based on the system architecture shown in Fig. 5.9. The external DRAM we assumed is 8-channel 3-D wide I/O with 2KB page size using 64bit bus width. The number of PEs and SRAMs are obtained so as to guarantee the full utilization of PEs with a given memory specification. The 192 PEs are synthesized to operate at 50MHz while 24 SRAMs (4KB each) connected to PEs operate at 400MHz in 1.2V. One of the most important parts in the system is the memory controller, which delivers the data from cache to PE or from DRAM to cache

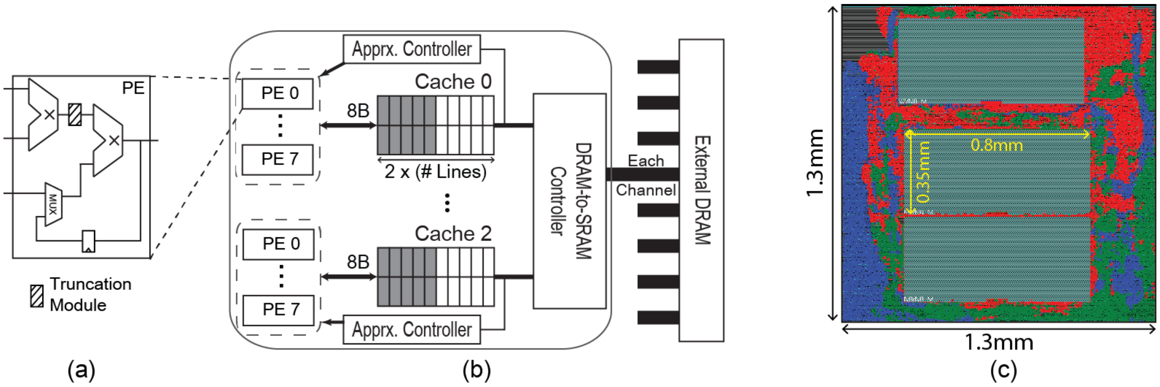


Figure 5.9: A system diagram: (a) a processing engine (PE), (b) the overall NN system including PEs, caches and controllers and (c) the layout of the components connected to each channel; 3 SRAMs and 24 PEs (40% approximate and 60% accurate) with required controllers.

based on the network topology. We should note that all data sequences from memory is pre-determined; therefore PEs do not have to request data to memory, but data is delivered from memory to PEs in order. In addition, memory address is not random, but is organized thus data will be fetched from memory row by row (implemented by a simple counter). Approximate H/W controller is placed to identify computations that should go into the approximate PEs. One dirty bit is used for the decision.

The systems are compared in terms of total power, area and output quality between the system with 100% accurate PEs at full precision (*system1: completely accurate*), the one with 100% accurate PEs with 50% of 16bit computation and the rest with 8bit (*system2; software approach*), the one with 100% approximate PEs at full precision (*system3; hardware approach*) and the one with 40% approximate PEs (60% accurate PEs) with bit-precisions obtained by Algorithm 1 (*proposed*). The overhead of dealing with dirty bits is negligible since the power and area of DRAM and SRAM controller is dominant. The power, area and accuracy analysis between four different systems are summarized in Table 5.1. The power consumption of PEs in the proposed system is 1.05W while that of other components is same. The PE power is reduced by 57% compared to the system 1. However, due to the noticeable SRAM power, the system power is reduced by 38% com-

Table 5.1: The system analysis in terms of power, area, and recognition accuracy

	Power (W)				Area (mm^2)			
	system1	*system2*	*system3*	*proposed*	*system1*	*system2*	*system3*	*proposed*
PEs	2.47	1.94	1.76	1.05	6.49	6.49	5.72	6.18
SRAMs	1.29	1.29	1.29	1.29	7.20	7.20	7.20	7.20
Controllers	0.024	0.024	0.024	0.024	0.58	0.58	0.58	0.58
Total	3.78 (ref)	3.26 (-14%)	3.07 (-19%)	2.36 (-38%)	14.27	14.27	13.50	13.96

	Recognition Rate
system1	0.905 (ref)
system2	0.902 (-0.3%)
system3	0.832 (-7.3%)
proposed	0.902 (-0.3%)

system1: 100% accurate PEs at full precision, *system2*: 100% accurate PEs with 50% of 16bit computation and the rest with 8bit,
system3: 100% approximate PEs at full precision, *proposed*: 40% approximate PEs and 60% accurate PEs
with 60% of 16bit computation and the rest with 8bit.

pared to system1. This amount of power saving is achieved by sacrificing only 0.4% of recognition accuracy.

5.5 Summary of the Chapter

This chapter presents a design method for a low-power digital feedforward neural network platform using approximate computing. The proposed approach identifies approximate synapses using the error sensitivity vector precomputed during the training phase of a feedforward NNs. The computation energy of approximate synapses are reduced using a coupled software (precision control) and hardware (approximate PEs) based approximation schemes. A greedy algorithm is presented to select the optimal bit-precisions of the synapses. The integrated approach helps reduce the full-chip power of a digital NN engine (processing engines, on-chip caches, and controller) integrated with a 3-D Wide I/O DRAM by 38% with little (0.4%) quality degradation compared to an accurate design. In summary, the proposed coupled software-hardware based approximate computing can help design low-power digital NN for learning static nonlinear systems in future SoCs.

CHAPTER 6

ENERGY-EFFICIENT LEARNING OF DYNAMIC NONLINEAR SYSTEMS

In Chapter 5, a low-power design method for feedforward neural network hardware is presented. Recently, deep learning architecture such as AlexNet [49] or GoogLeNet [172] have been widely used for complex problems like image classification and object recognition. However, deep learning architectures based on feedforward neural networks are not suitable for applications with sequential data processing. A recurrent neural network (RNN), illustrated in Figure 6.1, is capable of dealing with systems having temporal behaviors with external input sequences. Consequently, RNN has ability to memorize the history of past inputs or states and is useful for applications that require the notion of time, for example, language modeling [84], medical diagnosis [173], human activity recognition [174] and mapping finite state machines [83], to name a few. RNN is a more general NN compared to the deep learning networks, and has great potential due to its ability to approximate nonlinear dynamical systems. The research on algorithm and application of RNN is rapidly progressing [175, 176], but there is relatively little effort in energy-efficient RNN hardware.

This chapter presents the concept of feedback-controlled dynamic approximation for energy-accuracy trade-off in RNN. The approximate computing using reduced bit-precision and/or in-exact arithmetic have been investigated to enable energy-accuracy trade-off in hardware accelerators for feedforward NNs [111, 112]. However, approximate computation for digital RNN is currently unexplored. The main challenge for approximation in RNN is that, unlike feedforward NNs that operate on stationary inputs, an RNN operates on input sequences. Some parts of a given sequence may allow approximation while some parts may require precise computation. The recurrent connectivity in RNN can also lead to propagation of error over time. Hence, on-line, real-time, and dynamic control of the

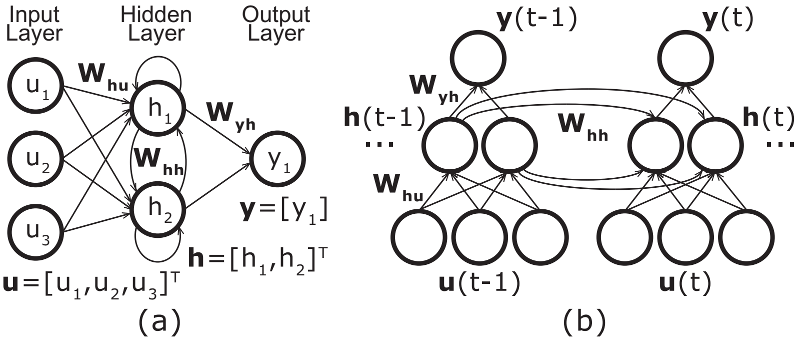


Figure 6.1: Overview of RNN: (a) RNN with one hidden layer where recurrent connections ($\mathbf{W_{hh}}$) are established between hidden neurons. (b) The same topology unfolded in time.

approximation is required in RNN and is presented in this chapter.

6.1 Preliminaries

6.1.1 Recurrent Neural Network

Recurrent neural network (RNN) is the network where states of hidden neurons (the neurons in a hidden layer) evolve over time. A RNN is designed by additional recurrent connections (forming any cycles in the network) in a feedforward neural network, such as multilayer perceptron [51]. There are various types of RNNs depending on where recurrent connections are established. The following dynamics define the standard RNN where recurrent connections are formed from outputs of neurons in a hidden layer to their inputs (Figure 6.1(a)).

$$\begin{aligned} \mathbf{h}(t) &= \mathbf{W_{hh}} \cdot (\mathbf{h}(t-1)) + \mathbf{W_{hu}} \cdot \mathbf{u}(t) + \mathbf{b_h} \\ \mathbf{y}(t) &= \mathbf{W_{yh}} \cdot \sigma(\mathbf{h}(t)) + \mathbf{b_y} \end{aligned} \tag{6.1}$$

where $\mathbf{u}(t)$, $\mathbf{h}(t)$, $\mathbf{y}(t)$ are inputs, states of hidden neurons and outputs at time t, respectively; $\mathbf{W_{hu}}$, $\mathbf{W_{hh}}$, $\mathbf{W_{yh}}$ represent the weight matrices; and $\mathbf{b_h}$, $\mathbf{b_y}$ are the biases. Activation function of a neuron in the hidden layer is denoted as $\sigma(\cdot)$; mostly, a sigmoid type function. Owing to the recurrence in the network, RNN is capable of approximating any dynamic nonlinear functions [74].

The operation of RNN can be easily understood by unfolding the recurrent connections in time (Figure 6.1(b)). By running RNN, we can predict an output vector sequence, $\mathbf{y} = [\mathbf{y}(0), \mathbf{y}(1), ..., \mathbf{y}(T)]$ for discrete-time case, with given an input vector sequence, $\mathbf{u} = [\mathbf{u}(0), \mathbf{u}(1), ..., \mathbf{u}(T)]$ where $\mathbf{u}(k) \in \Re^{nin}$ and $\mathbf{y}(k) \in \Re^{nout}$. The goal is to minimize (total) prediction error

$$E_{tot} = \sum_k E(k) = \sum_k \|\mathbf{y}(k) - \mathbf{y}^*(k)\|^2/2 \tag{6.2}$$

where $\mathbf{y}^*$ is the target output in supervised learning. The error function E_{tot} can be different depending on applications. Thus, the proper training of RNN has to be performed to minimize the prediction error during the inference.

6.1.2 Training of Recurrent Neural Network

Training RNN is more complex than training feedforward NN due to its ability to learn temporal information. The backpropagation through time (BPTT) [177] is a well-known training algorithm that extends the (standard) backpropagation algorithm by unfolding the temporal behavior of the RNN to form a layered feedforward NN. The objective of BPTT is to minimize the total error E_{tot} in (6.2) by changing weight matrices for each training epoch. In RNN training, an epoch is one input vector sequence $\mathbf{u}$. Each weight update is given by

$$\Delta w_{ji} = -\eta \frac{\partial E_{tot}}{\partial w_{ji}} \tag{6.3}$$

where w_{ji} is a connection weight from node i to j and η is the learning rate. The number of layers of constructed multilayer feedforward NN linearly increases by an additional time step involved in the training. Thus, the memory space required for BPTT increases as the length of a training sequence increases, making this method mostly suitable for off-line training of RNN.

For on-line training, real-time recurrent learning (RTRL) can be performed which updates the synaptic weights while operating the network [75]. To train RNN in real-time, RTRL tries to minimize the instantaneous error (k) in (6.2). The RTRL approximates the gradient of total error with respect to weight changes by an instantaneous estimate of the gradient:

$$\Delta w_{ji,k} = -\eta \frac{\partial E_k}{\partial w_{ji}} \tag{6.4}$$

where k is the current time step. The computational complexity of RTRL is high since weight update is required at every time steps.

6.1.3 Approximate Feedforward Neural Network

The recent studies have developed low-power feedforward NNs using approximate computing [111], [Chapter 5]. The error sensitivity of each synaptic weight ($\partial E/\partial w_{ji}$) is computed during conventional backpropagation training of a feedforward NN. This information is utilized for the practice of approximate computing (e.g. precision control) in inference stage. As an early work regarding this topic, Venkataramani et al. minimized power by reducing bit-precisions for those neurons with lower average error sensitivity (approximate neuron) [111]. A layer with the most power consumption is selected and neurons in that layer are iteratively selected for approximation. This approach can limit the power reduction under a quality constraint as a neuron may have a few strong connections but low average error sensitivity, selecting such a neuron for approximation may limit quality.

Even better, approximate synapse is presented in Chapter 5 which is a fine-grained approach to identify synaptic weights (edges) instead of neurons in the network (see Figure 6.2). This approach sorts entire error sensitivities and assign lower bit-precisions from a fixed bit-precision set (e.g. [24bit, 16bit, 8bit]) to synaptic connections with lower error sensitivities. A greedy algorithm is presented that starts with the least possible bit-precision (e.g. 100% 8bit operation) and iteratively increases the ratio of higher bit-precisions until the inference of validation set reaches a given quality constraint.

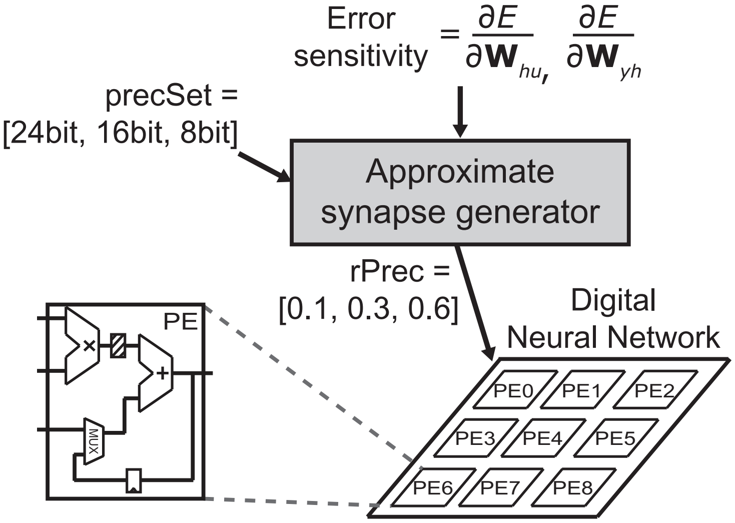

Error sensitivity = $\frac{\partial E}{\partial \mathbf{W}_{hu}}, \frac{\partial E}{\partial \mathbf{W}_{yh}}$
precSet = [24bit, 16bit, 8bit]
Approximate synapse generator
rPrec = [0.1, 0.3, 0.6]
Digital Neural Network
PE
×
+
MUX
PE0
PE1
PE2
PE3
PE4
PE5
PE6
PE7
PE8

Figure 6.2: Overview of approximate synapse for low-power operation of digital neuromorphic hardware. Here, *p*recSet represents the fixed bit-precision set and *r*Prec is the ratio of each bit-precision.

6.1.4 Sequence Classification Using RNN

Figure 6.3 illustrates the video classification (or any sequence classification) using RNN. First, an input data sequence is fed into a local feature extractor. This local feature extractor can be any type of pre-processors to identify important local features (or raw input data can be used directly). The features at each time step are passed through RNN algorithm to compute output sequences. The number of output neurons in RNN is identical to the number of video activities to classify. Each output neuron corresponds to each activity. By looking at each output vector, the maximum index at each time step k is computed (by *Max index identifier*). Finally, the video activity is decided by selecting the one with the largest probability in the maximum index set for a pre-defined time length L (in *Classifier*). This time length L is named as *decision window*. This process is continued until the input sequence ends (until T in Figure 6.3).

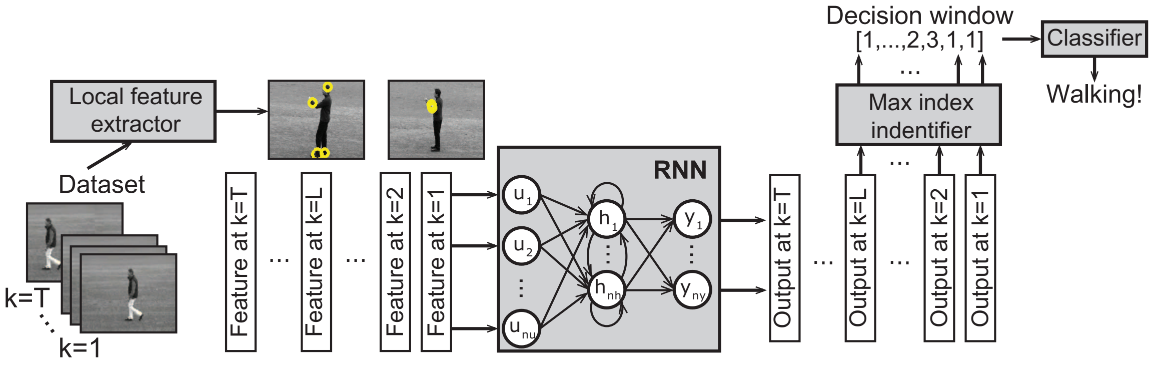


Figure 6.3: Block diagrams of video classification using RNN. Local feature extractor is used to recognize important local features to obtain an input sequence for RNN. Then, RNN outputs a sequence which can be post-processed to decide which activity a subject in the video is performing.

6.2 Approximate Computing in RNN

6.2.1 Static Approximation in RNN

The method of approximate synapse for RNN is adopted as the *static approximation*. Approximate synapse algorithm (or approximate neuron) has been presented with feedforward NN. However, there is little effort on the application of approximate computing in RNN which naturally propagates error through time. For this reason, static approximation in RNN is evaluated as well as dynamic approximation which will be presented in the next subsection. Our approach can also utilize approximate neuron, however, for brevity only approximate synapse is considered.

When BPTT training algorithm is used, the only difference between the feedforward NN and RNN is an additional weight matrix $\mathbf{W_{hh}}$. Thus, the approximate synapse algorithm can be directly applied to RNN for computing *rPrec* (refer to this as static approximation). During BPTT training algorithm, the following error sensitivity vectors are computed: $\partial E_{tot}/\partial \mathbf{W_{hu}}$, $\partial E_{tot}/\partial \mathbf{W_{hh}}$, $\partial E_{tot}/\partial \mathbf{W_{yh}}$. Then, static approximation algorithm sorts entire error sensitivities in ascending order to assign bit-precisions from low to high. As a result, it outputs *rPrec*, e.g. [0.1, 0.3, 0.6] when three possible bit-precisions are available.

Note the power reduction using the static approximation is limited by the choice of the input bit-precision set, e.g. [24bit, 16bit, 8bit]. The minimum required bit-precision for reasonable accuracy at the lowest power dissipation is difficult to determine a-priori and will depend on applications. Also, the feedforward NN takes stationary inputs, not a sequence. However, RNN deals with input sequences, there may be parts of the sequence that is difficult to classify with lower precision while other parts may be fairly easy. Hence, there is a need to dynamically modulate the level of approximation of each synapse over time for better power management in digital RNN.

6.2.2 Dynamic Approximation in RNN

The *dynamic approximation* algorithm is proposed for the energy-efficient operation of digital RNN for sequence classification problems. It utilizes the static approximation as a basis to determine the bit-precision ratios (*rPrec*) and dynamically changes the bit-precision set (*precSet*). The proposed feedback control system is not limited to precision control but can be applied to any approximate computing methods, such as imprecise hardware or voltage over-scaling. The *rPrec* is kept constant for entire RNN computations during the operation of dynamic approximation. However, *precSet* is adaptively changed which deviates from static approximation (Figure 6.4).

The limitation is that static approximation uses the pre-determined bit-precision set. This set cannot be changed when each input is equally important as in the feedforward NN. In RNN, there may be input sequences which are easy to detect the corresponding activity even with very low bit-precisions. On the other hand, some input sequences will make RNN to have less confidence in decision of which class (or activity) the given sequence falls in. In this scenario, reducing bit-precision may lead to decision error (accuracy degradation). Therefore, adapting *precSet* by looking at the current decision confidence will allow better power management in RNN. This confidence level can be checked by looking

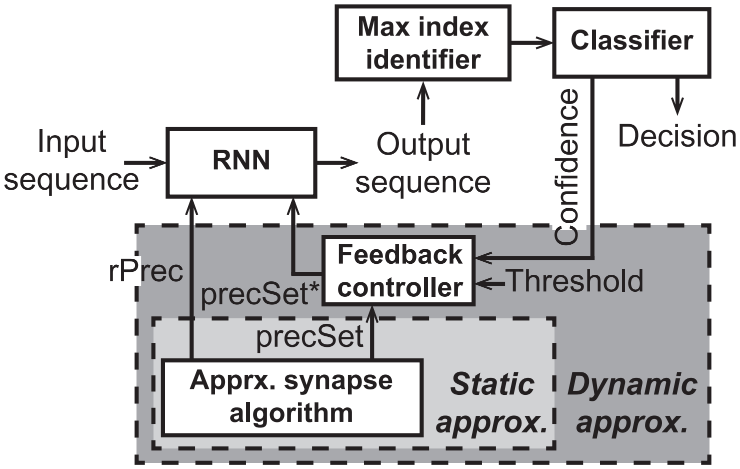


Figure 6.4: The proposed approximation algorithm for RNN in video (or sequence) classification.

at the actual counting value, equivalently a probability mass function (PMF), of the current decision index in decision window. This is compared to pre-determined *Threshold*. Then, the feedback controller changes *precSet* according to the relation between confidence level and *Threshold*. For instance, when the confidence is higher than the *Threshold*, *precSet* can be reduced to *precSet** = [18bit, 10bit, 6bit] instead of using [24bit, 16bit, 8bit] (see Figure 6.4). The entire process of the dynamic approximation algorithm is also described in Algorithm 2. This entire process can be performed in software level. However, when the sequence that we are dealing with has faster sampling rate (such as speech), the feedback control has to be fast enough to compute next *precSet* (*precSet**) in time.

6.2.3 Feedback Controller in Digital RNN

For analysis, two different feedback controllers are designed and compared: 1) hysteretic controller and 2) proportional controller. A digital RNN with the feedback controller is shown in Figure 6.5. The dynamic precision control is done sequence by sequence (discrete-time control). A hysteretic controller has two thresholds (upper and lower). The operation is quite simple: if confidence level exceeds upper threshold, bit-precisions in *precSet* are decreased by one. Likewise, if the confidence level is less than lower threshold,

Algorithm 2 Dynamic Approximation in RNN

```
Input:
    grad: error sensitivity
    $\mathbf{W_{hu}}, \mathbf{W_{hh}}, \mathbf{W_{yh}}$: weight matrices
    $Q_{target}$: target quality
    $\mathbf{D_{valid}}, \mathbf{D_{test}}$: datasets
    precSet: initial bit-precision set
    Threshold: reference threshold for feedback controller
Output:
    recog: recognition accuracy

Begin
rPrec ← static_approximation($Q_{target}, grad, \mathbf{D_{valid}}, precSet$)
for i in $|\mathbf{D_{test}}|$ do
    initialize maxCntArray to [0, 0, ..., 0]
    initialize decWindow
    while $i^{th}$ $\mathbf{D_{test}}$ sequence ends do
        $\mathbf{RNN_{OUT}}$ ← RNN($i^{th}$ $D_{test}$ sequence, $\mathbf{W}$, $rPrec, precSet$)
        $max\_idx$ ← max_index_identifier($\mathbf{RNN_{OUT}}$)
        decWindow ← update_window($max\_idx$, decWindow)
        [$Confidence, decision\_idx$] ← classifier(decWindow)
        precSet ← feedback_control($Confidence, Threshold, precSet,$
                    $prev\_decision, decision\_idx$)
        set $prev\_decision$ to $decision\_idx$
        maxCntArray[$decision\_idx$] increase by 1
    end while
    recog ← update_recognition_accuracy(maxCntArray)
end for
return recog
End
```

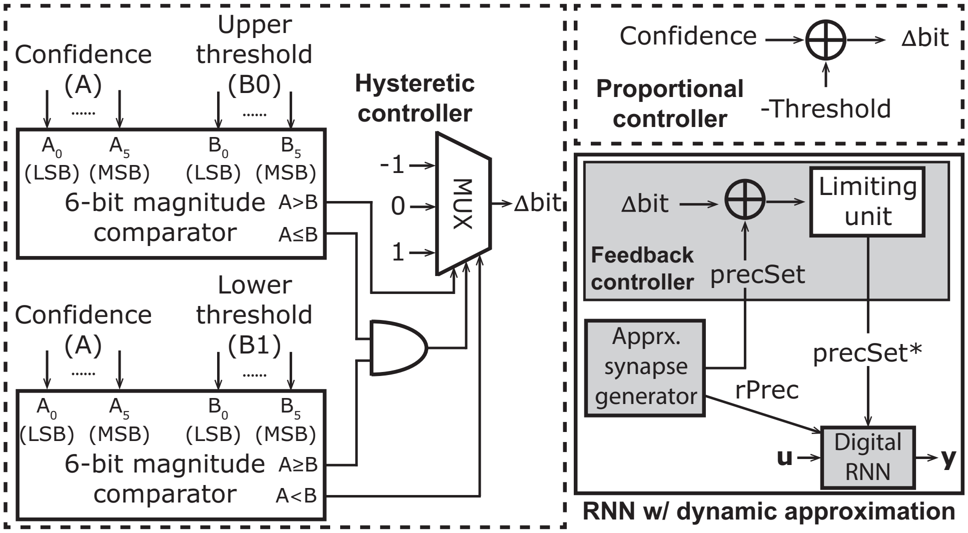


Figure 6.5: The proposed feedback controllers to implement dynamic approximation for RNN in hardware.

bit-precisions in *precSet* are increased by one. When the confidence of classifiers output is between upper and lower threshold, *precSet* is kept as it is.

Another possible controller is a proportional controller. Hysteretic controller has a drawback when the confidence level bounces back and forth near upper (or lower) threshold which may lead to increase in switching power. The proportional controller changes *precSet* depending on difference between a given threshold and confidence level. The number of bits to be changed in the *precSet* is defined as follows:

$$\begin{aligned} diff &= Threshold - Confidence, \\ \Delta bits &= \lfloor \alpha \cdot diff + 0.5 \rfloor \end{aligned} \tag{6.5}$$

where α is the proportional factor and $\Delta bits$ is the number of bits to be increased or decreased in *precSet*. The α is set to 8 for following experiments (making it power of 2 simplifies hardware by using a shifter). Due to the proportionality, the controller does not change the *precSet* when it reaches near the threshold.

As common for both controllers, *precSet* bounces back to the pre-defined (reasonably high) bit-precision set if the current decision differs from the previous decision. The change of decision may be caused by approximation error (due to the reduced bit-precision) or different activities detected (which is important to track with high precision). Also, there are upper and lower limits where bit-precision can reach. The upper limit may be set by hardware design specification (the maximum bit-width in hardware). The lower limit is set to keep the accuracy above a certain level.

6.3 Simulation Results

6.3.1 Benchmark: Human Activity Recognition

To verify the proposed method, benchmarks are briefly introduced in this subsection. To validate the effect of dynamic approximation in RNN hardware, datasets related to human activity recognition are selected as benchmarks [178, 179, 180]. KTH dataset contains 6 different human actions performed by 25 subjects in four different scenarios recorded by a camera [178]. Another dataset used in the experiment is UCFG dataset which consists of 10 different actions performed by 12 subjects in four different directions (this dataset is recorded by the ground camera) [179]. The last dataset is USC-HAD in which the data is captured by using wearable sensors (3-axis accelerometer and gyroscope) [180]. It contains 12 different daily human activities captured from 14 subjects with 5 trials. Each dataset is divided into three sets: training, validation and inference sets as summarized in Table 6.1. In the simulation, space-time interest points (STIP [181]) is utilized as a local feature extractor for KTH and UCFG datasets. Since USC-HAD dataset is already measured by wearable sensors, raw data are directly used as local features. Sampling rates are also noted in Table 6.1. Since video sampling rate is low, both software and hardware approaches are possible for the dynamic approximation.

6.3.2 Operation of Dynamic Approximation

To simulate RNN with dynamic approximation (Figure 6.4), a recurrent neural network with each feedback controller is implemented using Theano library [182]. The hidden

Table 6.1: The classification of each dataset into training, validation and inference sets

Benchmark	Category	Subject Index
KTH (25fps)	Training	11,12,14,15,16,17,18,25
	Validation	1,4,19,20,21,23,24
	Inference	2,3,5,6,7,8,9,10,22
UCFG (60fps)	Training	1,2,3,4
	Validation	5,6,7
	Inference	8,9,10,11,12
USC-HAD (100Hz)	Training	6,9,10,11,12
	Validation	1,3,5,14
	Inference	2,4,7,8,13

neurons are fully connected with each other. With the training set in Table 6.1, a RNN is trained for each dataset using BPTT training algorithm. The resulting synaptic weight matrices and error sensitivities are used in the experiment. The operations of the proposed RNN with feedback control are shown in Figure 6.6. In this example, a handclapping (a-d) and a boxing (e-f) video are used where each has 200 input sequences. The decision window L is set to 50. A subject is performing each activity and the objective of RNN is to correctly classify the video.

Figure 6.6(a) and (b) show the operation of a hysteretic controller. The upper threshold is set to 0.7 and lower threshold is set to 0.5. Since the maximum confidence level at initial video frames (identify as boxing) exceeds the upper threshold, *precSet*[0] and *precSet*[1] are reduced by one at each sequence which are 24bit and 16bit initially. The minimum bit-precision (8bit) is not controlled in this example. As the confidence level stays between upper and lower thresholds, bit-precisions are kept as it is. At around sequence 35, the activity with the maximum confidence level changes from boxing to handwaving, thus the *precSet* sharply increases to the pre-defined bit-precisions. The bit-precision continues to change depending on the confidence level.

The same video set is used to show how proportional controller modifies *precSet* [Figure 6.6(c) and (d)]. Initially, as in the hysteretic controller, *precSet* decreases due to high confidence level. The difference is that *precSet* reaches the minimum set faster than hys-

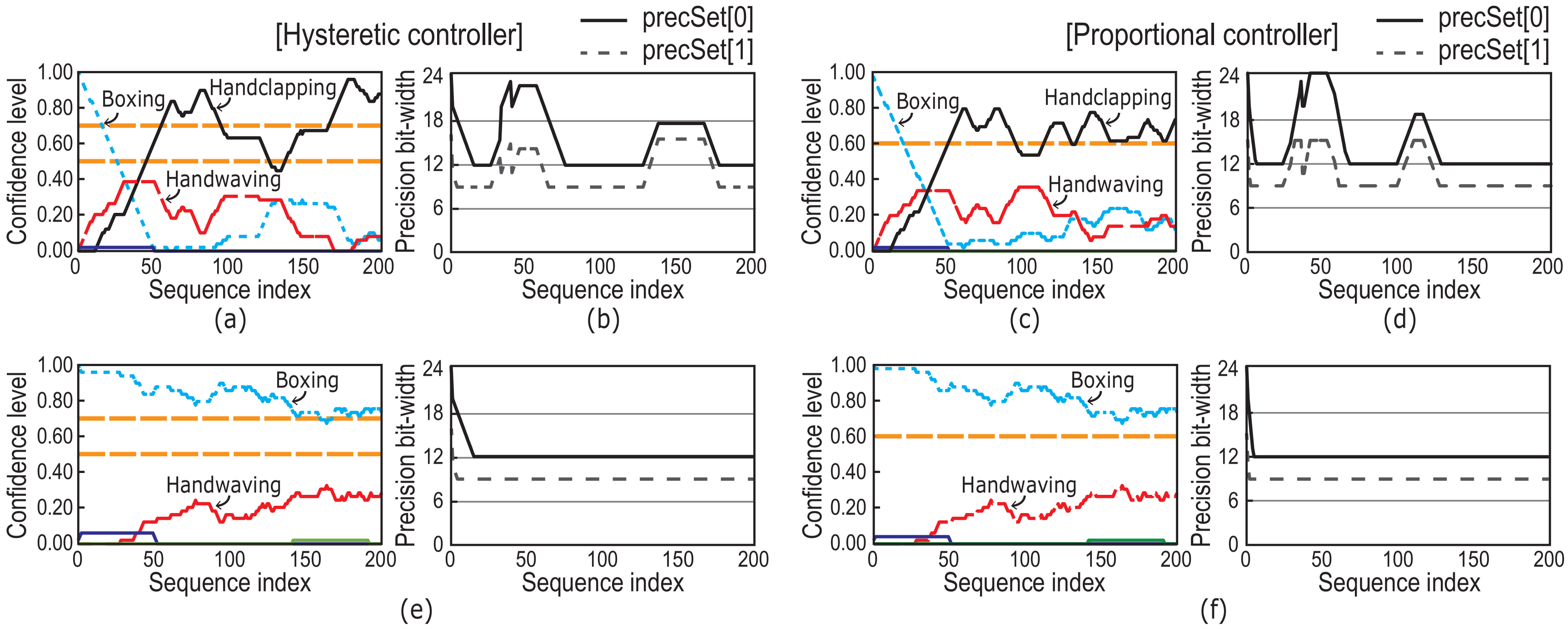


Figure 6.6: The operation of the dynamic approximation to adaptively change bit-precisions of synapses. (a) Confidence level of each activity and (b) the resulting precSet by using hysteretic controller. (c) Confidence level of each activity and (d) the resulting precSet by using proportional controller (handclapping). (e-f) The same set of experiment for different human activity (boxing).

teretic controller. This comes from the proportionality as in (6.5). It is also true when bit-precisions are required to be increased. Since hysteretic controller does not change bit-precision within the threshold band, the confidence level can go below the lower threshold. However, proportional controller keeps the confidence level near threshold quite well (Figure 6.6(c)). The same experiments on boxing activity shows that the feedback controller keeps bit-precision set at the lower limit when the input sequence is quite easy to classify which shows why the dynamic approximation is beneficial [6.6(e) and (f)].

6.3.3 Digital RNN with Dynamic Approximation

This subsection presents the design of the digital RNN engine with a feedback controller, synthesized in 28nm CMOS technology. In digital neuromorphic hardware, the most power-consuming block is the processing engine (PE) consisting of a multiply-and-accumulate (MAC) unit and a data controller. For reasonable performance of RNN accelerator, 16 PEs are assumed in the system where they can handle MAC computations in parallel. The bit-width of PE is designed to be 24bit and 20 LSBs can be controlled to force zeros depending on the bit-precision of each computation (see inset of Figure 6.7). Full-precision of a MAC unit is set to 24bit by simulating each benchmark with different bit-precisions.

The last 2bits in each data work as dirty bits to inform which bit-precision to be used (all simulations reflect this). The *precSet** is realized by the generation of a mask to additionally force zeros to simplify hardware. The static control is possible without the feedback controller and the AND gate. It is important to compute the area/power overheads of the feedback controller compared to the other on-chip hardware components required to implement digital RNN with dynamic approximation. Since the dynamic approximation requires a controller per memory channel that controls data for PEs, four data controllers are designed in the system (assume 4 memory channels). The remaining components are also shown in Figure 6.7.

The digital RNN with dynamic approximation is integrated and synthesized in 28nm

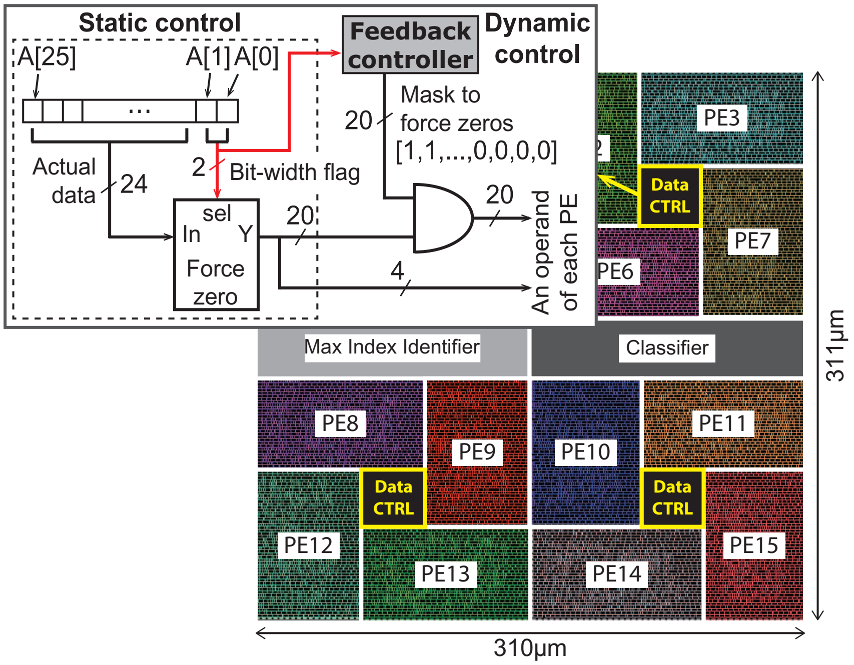


Figure 6.7: Synthesis and placement of the RNN hardware with dynamic approximation. A small data controller is placed near the center.

technology. The power consumption and area are reported in Table 6.2 (prior to the placement) and the layout after the placement is shown in Figure 6.7. The total area is 0.096mm^2 after the placement. The most power is consumed by the PE group (96.97%). Four data controllers, the only additional component to realize dynamic approximation, takes negligible portion (0.31%) of power consumption. Likewise, the area of four controllers occupies only 1.22∼1.36% of the total area. Even though proportional controller has smaller form factor than the hysteretic controller, both options do not impose much hardware overhead. The overhead from the static approximation is only 0.06mW of power (0.15%) and 451.3μm^2 of area (0.57%).

6.3.4 Energy-Accuracy Trade-off

The baseline operation of the RNN engine is with a single fixed point for all computations (e.g. 100% 24bit). The minimum bit-precision to maintain the highest possible recognition accuracy is selected for each benchmark as a baseline (refer to Figure 6.8). This is to make

Table 6.2: Power consumption and area breakdown of the proposed RNN hardware with dynamic approximation

		16 PEs	Data CTRL (Hyst, Prop)	Max Index Identifier	Classifier
Power (mW)		40.48	0.131, 0.128	0.596	0.539
		96.97%	0.31%, 0.31%	1.43%	1.29%
Area (μm^2)		70053.92	1077, 964	4194.19	3914.17
		88.41%	1.36%, 1.22%	5.29%	4.94%

sure not to overemphasize the efficiency of the dynamic approximation algorithm. The experimental results by utilizing the static approximation algorithm in RNN is obtained as well. For both cases, the average power is computed by:

$$P_{avg} = \sum_{i} rPrec[i] \cdot P_{precSet[i]} \tag{6.6}$$

where $rPrec[i]$ is the i^{th} precision ratio (e.g. 0.6), $precSet[i]$ is the ith bit-precision used in the system (e.g. 18bit) and $P_{precSet[i]}$ is the power dissipation of a single PE when computing with $precSet[i]$. For a RNN with dynamic approximation, the power consumption of a PE changes over time because the operating bit-precision of a PE changes dynamically. Thus, the average power consumption of single PE can be computed by:

$$P_{avg} = \sum_{i} rPrec[i] \cdot (\sum_{k=1}^{T} P_{precSet(k)[i]}/T) \tag{6.7}$$

where T is the length of an input sequence and $precSet(k)$ is the precSet at time step k.

Using the Theano simulator, recognition accuracy of each dataset is obtained by using inference dataset and results are shown in Figure 6.8. Compared with the baseline case, the proposed dynamic approximation degrades accuracy by 4.64% (hysteretic) and 3.93% (proportional) on average. With this slight accuracy degradation, average power consumption of a PE is reduced by 35.72% (hysteretic) and 36.36% (proportional) on average. Also, the accuracy degradation and power saving are compared with those obtained by using the

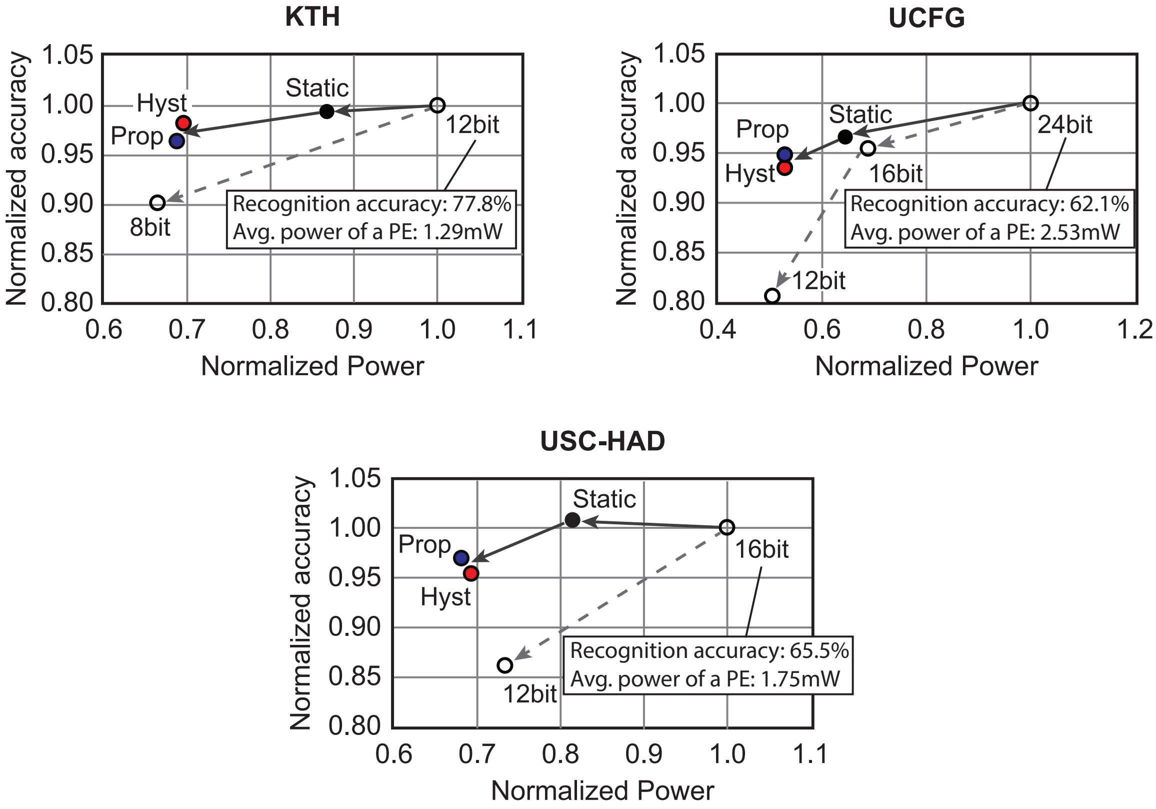


Figure 6.8: Simulation results showing normalized accuracy vs. normalized power for each benchmark with different computation methods: single fixed point, static approximation and dynamic approximation (hysteretic or proportional).

static approximation. The additional power (equivalently, energy) reduction by using dynamic control is 13.82% on average while sacrificing accuracy by 3.2% on average. Note the blind control of the bit-precisions of all computations can lead to significant degradation in quality. Between two feedback control methods, a proportional controller gives slightly better performance with less power consumption. This is due to the fact that hysteretic controller stops changing bit-precision within the threshold band which may degrade accuracy as in Figure 6.6(a) at around sequence 140.

6.4 Summary of the Chapter

This chapter presents the dynamic approximation algorithm for energy-efficient operation of digital RNN. The proposed approach adaptively controls the level of approximation in synaptic weights depending on the confidence level of output sequence. The easy sequences are computed with more approximation and hard sequences with less approximation. The dynamic control of the bit-precision is considered to verify the proposed approach. The analysis shows that the dynamic approximation allows significant power saving with graceful degradation of quality. The presented approach can enable application of RNN at the sensor/camera front-ends for real-time activity detection/recognition.

CHAPTER 7

ANALYSIS OF ENERGY-ACCURACY TRADEOFF IN DIGITAL CELLULAR NONLINEAR NETWORK

In previous chapters, the energy-efficient hardware platforms of two different system learning methods, model-based and data-driven, are presented. In Chapter 4, Cellular Nonlinear Network (CeNN) is selected as a computing algorithm to accelerate the model-based system learning. However, CeNN model can be used in data-driven learning as well since the network can be trained with large data by Hebbian learning [183] or a genetic algorithm [184] to estimate underlying systems. Therefore, it is important to understand energy-accuracy tradeoff for CeNN algorithm as it is capable of serving as either model-based or data-driven system learning. As test benchmarks for the analysis, various image processing applications are selected.

There hardware accelerators for neural and non-Boolean image processing have gained significant interest for mobile System-on-Chips (SoCs) [185, 186, 187]. CeNN is a neuro-inspired parallel computing architecture using a two-dimensional array of cells where each cell is connected to a set of local neighbors (Figure 7.1) [188]. The cell dynamics is determined by an ordinary differential equation (ODE) with local interaction defined by feedback **A** and feedforward **B** templates. The CeNN has shown significant performance advantage on image processing problems, thanks to the concurrent processing and local connectivity.

Although, application of CeNN as embedded image processing platform has received significant interest [189, 190], low operating power with feasibility of dynamic scaling of energy, performance, and quality-of-result is of critical importance to make CeNN viable as an accelerator in mobile SoCs. Boolean image processors often allow errors during computation to achieve low power with graceful degradation in image quality [191, 192,

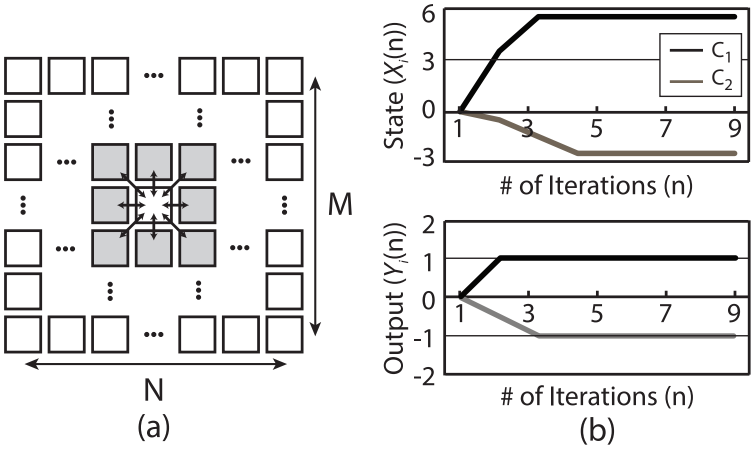


Figure 7.1: (a) Structure of a $M \times N$ CeNN with a cell locally connected to neighbor (gray) cells and (b) dynamic behavior of the state (X_i) and the output (Y_i) of two randomly selected cells (C_i).

193]. The voltage over scaling (i.e. reducing voltage below the minimum limit imposed by critical path delay) has been proposed to reduce voltage with controlled increase in bit-error-rate (BER) [191]. Methods have focused on protecting more significant bits (MSBs), reduction of bit-width (precision control), and design of imprecise adders and multipliers to reduce power [192, 193].

This chapter explores the potential of accuracy-aware processing to reduce power dissipation and enable dynamic energy scaling in CeNN based image processors. The image processing using CeNN and using conventional Boolean architecture fundamentally differs in two ways: (i) algorithms are implemented directly in the ODE based cell dynamics; and (ii) computation is performed by local interactions between cells in a network. There is a need to understand the impact of error in CeNN to evaluate the potential of energy-accuracy tradeoff.

7.1 Background

7.1.1 Fundamentals of Cellular Nonlinear Network (CeNN)

Fig. 7.1(a) shows an $M \times N$ CeNN having cells placed in M rows and N columns. The state of each cell is computed by the sum of the weighted inputs and outputs from the cell and its neighboring cells. The differential equation of each cell is described as [188]:

$$C\frac{\partial x_{ij}(t)}{\partial t} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} A_{kl} \cdot y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B_{kl} \cdot u_{kl} + z \tag{7.1}$$

$$y_{ij}(t) = f(x_{ij}(t)) = \begin{cases} -1 & \text{for} \quad x_{ij}(t) < -1 \\ x_{ij}(t) & \text{for} \quad |x_{ij}(t)| \leq 1 \\ 1 & \text{for} \quad x_{ij}(t) > 1 \end{cases} \tag{7.2}$$

where i is the row index, j is the column index, $x_{ij}(t)$ is the state, $y_{ij}(t)$ is the output, u_{ij} is the input, z is the offset, **A** is the feedback template, and **B** is the feedforward template for each cell $C(i,j)$. Here, $N_r(i,j)$ represents neighbors of a cell $C(i,j)$ within radius r where (k,l) is the index of those cells. Figure 7.1(a) depicts the CeNN structure with $r = 1$.

The weights of template **A** and **B** (representing the local interconnection) as well as offset z are programmed by users depending on which image processing applications to run. The templates are learnt for different applications offline by using a CeNN truth table [194] or a genetic algorithm [184]. As learning is performed off-line, the energy impact of learning, and hence, potential of energy-accuracy tradeoff during learning is not considered.

The dynamic range of $x_{ij}(t)$, which is the maximum range of the state, and the stability of CeNN are proved in [188]. The stability of CeNN guarantees that the system converges to a stable state at $t \to \infty$. When CeNN reaches the steady-state, the following properties should be satisfied:

$$\frac{\partial x_{ij}(t)}{\partial t} = 0, \lim_{t\to\infty} |x_{ij}(t)| > 1 \, for \, \forall(i,j). \tag{7.3}$$

These properties are true when ($A_{ij} > 1/R$) is satisfied in (7.1). As a result, the output of each cell in CeNN always provides a constant value as the transient decays to zero:

$$\lim_{t\to\infty} y_{ij}(t) = \pm 1. \tag{7.4}$$

This is evident by looking at the steady-state condition (7.3) and the output function in (7.2).

7.1.2 Dynamic Route of CeNN

The stability of a CeNN system can be understood further by looking at a dynamic route as shown in Figure 7.2. This basically shows the transient of a state variable ($\partial x_{ij}(t)/\partial t$) depending on the state of a cell, $x_{ij}(t)$. The dynamic route for each cell is defined by rewriting the right-hand side of (7.1) [188]:

$$C\frac{\partial x_{ij}(t)}{\partial t} = -f(x_{ij}(t)) + g(t), \tag{7.5}$$

where

$$f(x_{ij}(t)) = -A_{ij} \cdot y_{ij}(t) + \frac{1}{R} x_{ij}(t) \tag{7.6}$$

and

$$g(t) = \sum_{C(k,l)\in N_r(i,j), C(k,l)\neq C(i,j)} (A_{kl} \cdot y_{kl}(t) + B_{kl} \cdot u_{kl}) + B_{ij} \cdot u_{ij} + z. \tag{7.7}$$

In (7.6), $y_{ij}(t)$ can be written as a function of $x_{ij}(t)$. The shape of a piecewise linear plot (Figure 7.2) is determined by $-f(x_{ij}(t))$ and offsets by $g(t)$, called offset level.

For convenience, let's assume $R = 1$, $C = 1$, $A_{ij} = 2$ and $g(t) = 0$. This assumption can be made without violating parameter requirements as in [188]. R and C values can be predetermined by the user and weights of each template are determined accordingly during the learning process. Then, a cell has two stable equilibrium points at -2 and 2. The arrow on the route shows how $x_{ij}(t)$ settles to its stable equilibrium points as time goes by. For

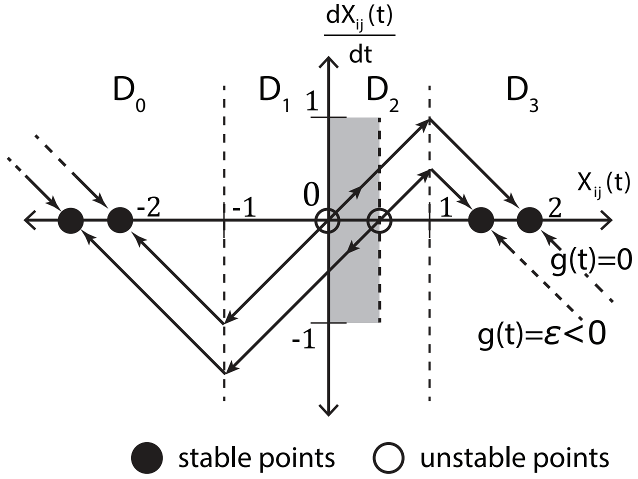


Figure 7.2: Dynamic routes and two equilibrium points of the state $x_{ij}(t)$ when $g(t) = 0$ [188]; dynamic route of a cell changes due to error ϵ (direction reverses in the shaded area).

instance, if $x_{ij}(t)$ is having a value on region D_2 or D_3 ($x_{ij}(t) > 0$), the state tends to settle at 2. After reaching the steady-state, when (7.3) is satisfied, $x_{ij}(\infty)$ equals to 2 which makes $y_{ij}(\infty) = 1$ by (7.2).

7.2 Impact of Error on CeNN

In this section, the error characteristics of a CeNN is analytically studied, specifically focusing on error propagation. The first-order Euler approximation can be applied to obtain a discrete-time CeNN equation (by setting discretization step $h = RC = 1$). Then, the CeNN equation (7.1) with an additive error (e^g) becomes

$$x_{ij}(n+1) = \sum_{C(k,l)\in N_r(i,j)} A_{kl} \cdot y_{kl}(n) + \sum_{C(k,l)\in N_r(i,j)} B_{kl} \cdot u_{kl} + z + e^g_{ij}(n+1). \quad (7.8)$$

where n represents the current time step (iteration index) and e^g_{ij} is the amount of generated error added to the next state value at $C(i,j)$. Essentially, an additive random error is assumed generated in each step of the CeNN computation. The error may have any distribution. The state $x_{ij}(n+1)$ of CeNN deviates from its original value depending on the magnitude of e^g_{ij}. Note equation (7.8) considers the addition of error as a real number (analog value) to obtain a more generic understanding. In Section **??**, the sources of error in digital cells due to voltage over scaling (physical bit flip error due to timing violation) and reduced bit-precision (quantization error) are considered, both of which lead to e^g_{ij} in equation (7.8).

7.2.1 Impact of Error on Cell Dynamics

To understand the effect of error, the abscissa is divided into four different domains ($D_0 \sim D_3$) in Figure 7.2. The worst scenario is when the sign of $x_{ij}(t)$ changes (move from D_2/D_3 to D_0/D_1) due to error forcing an output $y_{ij}(\infty)$, that is supposed to settle at 1, to converge at -1, which degrades accuracy significantly. Even though the sign of $x_{ij}(t)$ has not been changed, $|x_{ij}(t)|$ may stay below 1 (D_1 or D_2), and $y_{ij}(\infty)$ does not settle to either -1 or 1. Since the CeNN converges to its equilibrium points when (7.3) is satisfied for all cells, CeNN may run forever (convergence time$\rightarrow \infty$) unless there is a constraint on the number of iterations.

The equilibrium points are determined by $g(t)$ described in (7.7). Assume $g(t)$ is defined as $g(n)$ in digital CeNN. Then, $g(n)$ is dynamically affected by $\sum A_{kl} \cdot y_{kl}(n)$, which is the sum of feedback effects from neighbor cells; other variables remaining constant. Now, let's analyze the effect of error propagation in CeNN. First, we need to understand whether an error added to the state will lead to error in the output (input to output error propagation). Next, how error generated during state computation in the current iteration propagates through the network to affect the state computation in the next iteration is mathematically analyzed.

Error propagation of the state to output: The error may not propagate to cell $C(i,j)$ when $|x_{ij}(n)| > 1$ because $y_{ij}(n)$ is clamped to -1 or 1 [see (7.2)]. In this case, error affects the output $y_{ij}(n+1)$ only when $e^g_{ij}(n+1)$ is large enough to change state to $|x_{ij}(n+1)| < 1$ or to an opposite signed value. Thus, two error terms are defined: state error ϵ^x_{ij} and output error $\epsilon^y_{ij} = NL(\epsilon^x_{ij}, x_{ij})$, where $NL(\cdot)$ is a nonlinear function which bounds the error propagation due to (7.2). $NL(\cdot)$ is a function of ϵ_{ij}^{x} and x_{ij} since ϵ_{ij}^{y} depends on the value of $(\epsilon_{ij}^{x} + x_{ij})$. It can be defined as follows:

$$\epsilon_{ij}^{y} = \begin{cases} -2 & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \leq -1 \\ (\epsilon_{ij}^{x} + x_{ij}) - 1 & \text{if} \quad |(\epsilon_{ij}^{x} + x_{ij})| < 1, \quad x_{ij} \geq 1 \\ 0 & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \geq 1 \end{cases} \tag{7.9}$$

$$\epsilon_{ij}^{y} = \begin{cases} -(x_{ij} + 1) & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \leq -1 \\ \epsilon_{ij}^{x} & \text{if} \quad |(\epsilon_{ij}^{x} + x_{ij})| < 1, \quad |x_{ij}| < 1 \\ 1 - x_{ij} & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \geq 1 \end{cases} \tag{7.10}$$

$$\epsilon_{ij}^{y} = \begin{cases} 0 & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \leq -1 \\ (\epsilon_{ij}^{x} + x_{ij}) + 1 & \text{if} \quad |(\epsilon_{ij}^{x} + x_{ij})| < 1, \quad x_{ij} \leq -1 \\ 2 & \text{if} \quad (\epsilon_{ij}^{x} + x_{ij}) \geq 1 \end{cases} \tag{7.11}$$

Note that CeNN converges to either -1 or 1 without any error. The condition (7.9) and (7.11) are thus dominantly observed as CeNN system converges. This implies that the distribution of ϵ_{ij}^{y} finally has high probability at 0, -2, or 2 independent of the distribution of ϵ_{ij}^{x}. In Section 7.2.3, this observation is numerically validated using error rates from a digital cell.

Error propagation through the network: Let us first examine both error terms at $n = 1$:

$$\begin{aligned} \epsilon_{ij}^{x}(1) &= e_{ij}^{g}(1), \\ \epsilon_{ij}^{y}(1) = NL(\epsilon_{ij}^{x}, x_{ij}(0)) &= NL(e_{ij}^{g}(1), x_{ij}(0)) \end{aligned} \tag{7.12}$$

where $x_{ij}(0)$ is an initial state of CeNN and e_{ij}^{g} is assumed to be a random variable. At $n = 2$, the state error then becomes

$$\epsilon_{ij}^{x}(2) = \sum_{C(k,l) \in N_r(i,j)} A_{kl} \cdot \epsilon_{kl}^{y}(1) + e_{ij}^{g}(2) = \sum_{C(k,l) \in N_r(i,j)} A_{kl} \cdot NL(e_{ij}^{g}(1), x_{kl}(0)) + e_{ij}^{g}(2). \tag{7.13}$$

By generalizing (7.13) at time step $(n+1)$, the state error is expressed as (7.14). Here, $\epsilon_{ij}^{x}(n+1)$ is represented as a function of e_{ij}^{g} of which distribution is given. According to (7.14), the impact of output error ϵ_{kl}^{y}, propagated from neighboring cells and the cell itself at previous time step, is amplified by weights of the feedback template A_{kl}. It implies that applications with strong feedback are likely to have enhanced error propagation (and hence, more quality degradation) compared to applications with weaker feedback.

$$\begin{aligned}\epsilon_{ij}^{x}(n+1) &= \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot \epsilon_{kl}^{y}(n) + e_{ij}^{g}(n+1)\\ &= \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot NL(\epsilon_{ij}^{x}(n), x_{ij}(n-1)) + e_{ij}^{g}(n+1) = \cdots\\ &= \sum A_{kl}\cdot NL(\sum A_{kl}\cdot NL(\cdots(\sum A_{kl}\cdot NL(e_{kl}^{g}(1), x_{ij}(0))\\ &\qquad + e_{ij}^{g}(2))\cdots) + e_{ij}^{g}(n)) + e_{ij}^{g}(n+1).\end{aligned} \tag{7.14}$$

Due to the nonlinearity of $NL(\cdot)$, equation (7.14) cannot be expressed as a closed form expression. If $|x_{ij}| \geq 1$, y_{ij} reaches its equilibrium point (either 1 or -1) which may clamp out the impact of small error. For an intuitive understanding, therefore, let's focus on the range of $|x_{ij}| < 1$ where small error is propagated through the network. This condition is important to understand the role of error at the early stage of CeNN operation where states are within the linear region. In this case, (7.13) becomes:

$$\epsilon_{ij}^{y}(2) = \epsilon_{ij}^{x}(2) = \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot\epsilon_{kl}^{y}(1) + e_{ij}^{g}(2) = \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot e_{ij}^{g}(1) + e_{ij}^{g}(2). \tag{7.15}$$

Noting that e_{ij}^{g} is an independent and identically distributed random variable, indices of cell position and time can be removed. Then, the standard deviation of $\epsilon_{ij}^{y}(2)$ becomes

$$\sigma(\epsilon_{ij}^{y}(2)) = \sqrt{\sum_{C(k,l)\in N_r(i,j)} (A_{kl}\cdot\sigma(e^{g}))^2 + \sigma^2(e^{g})} = \sqrt{(A_{SUM^2}+1)\cdot\sigma^2(e^{g})} \tag{7.16}$$

where $A_{SUM^2} = \sum A_{kl}^2$. Then, the variance of output error at time step m (assuming $|y_{ij}|$

does not reach 1) is:

$$\sigma^2(\epsilon_{ij}^{y}(m)) = (\sum_{p=0}^{m} A_{SUM}^{p} \cdot \sigma^2(e^g)). \tag{7.17}$$

The equation (7.17) implies that the generated error at each cell propagates through the network and its variance is amplified by the sum of squared element values of the feedback template (**A**) at each time step. This gives us the intuition of avoiding error at early stage of CeNN operation in applications with larger feedback template.

7.2.2 Convergence Time Constraints

As discussed at the beginning of Section 7.2.1, strict convergence may not be achieved with error since ($x_{ij}(n) - x_{ij}(n-1) \neq 0$) and the state $x_{ij}(n)$ with error may also settle at an incorrect level. Thus, a (relaxed) convergence time is defined for the following experiments. The convergence time considering the pseudo steady-state is defined as:

$$|x_{ij}(n_{conv} - x_{ij}(n_{conv} - 1)| < \beta, \, |x_{ij}(n_{conv})| \geq 1 \tag{7.18}$$

where n_{conv} is the convergence time and β is a fluctuation bound ($\beta = 2$ is used in our simulation). We say that CeNN is converged when all CeNN cells satisfy (7.18). This convergence time may vary depending on the error rate, which will be shown in Section 7.2.3. Hence, for image processing under performance constraints (e.g. real-time case), the effect of error on output quality needs to be compared with a fixed number of iterations (*convergence time constraint*).

7.2.3 Experimental Characterization of Error Propagation

In this subsection, the error propagation in CeNN is characterized. A fully digital realization of the CeNN dynamics is considered and random bit-errors are applied while computing (7.8) for each cell at each step. First, the numerical error distributions of the state x_{ij} and the output y_{ij} of CeNN cells are examined. As an example, the hole filling is used with

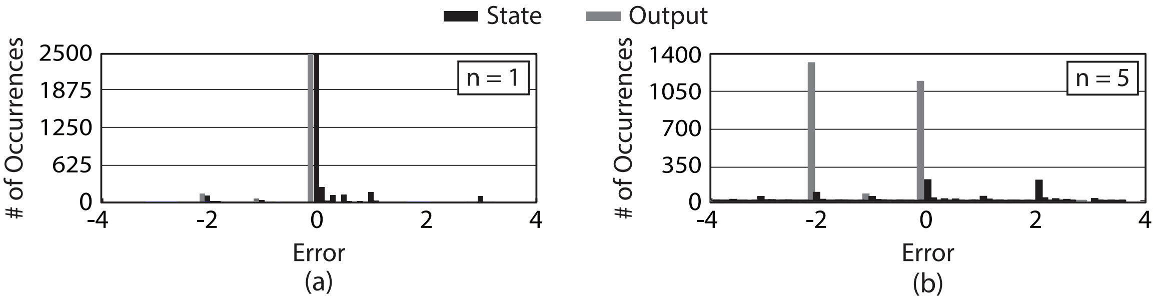


Figure 7.3: Histogram of error in CeNN state and CeNN output at time step (a) $n = 1$ and (b) $n = 5$. The analysis is shown with the hole filling at 5% bit error rate.

the size of input image 50×50. The bit error rate (P_b) is set to 5% with 12-bit precision. The state/output error ϵ^x_{ij} (or ϵ^y_{ij}) is computed by subtracting the state (output) of CeNN under error to that evaluated without error at each iteration time. Since the bit flip error is allowed with probability of P_b for all bit positions, the state error is uniformly distributed except at 0 (95% of cells have no error). The error distribution with 5% bit error rate is shown in Figure 7.3(a), for n=1, and Figure 7.3(b), for n=5. The distribution of the state error at $n = 1$ is the generated error [e^g in (7.8)] which is a property of the hardware. As expected the distribution is non-Normal. The computed standard deviation of the generated error at $n = 1$ is 2.05. The distribution of the state error at $n = 5$ captures the effect of error propagation [as discussed in (7.14)]. The distribution of the output error has several sharp peaks as iteration goes due to the bounding function $NL(\cdot)$ as in (7.9)$\sim$(7.11). The distribution of the output error at $n = 5$ shows peaks at -2, 0, or 2 as predicted in Section 7.2.1.

Next, the impact of feedback template (**A**) on error propagation is verified. For comparison, two different applications are selected: edge detection and hole filling. The edge detection ($\mathbf{A_1}$) has zero neighboring feedback weights while the hole filling ($\mathbf{A_2}$) has feedback weight of 1 for directly connected neighbor cells (inset in Figure 7.4(b)). Empirical cumulative distribution function (CDF) is utilized to identify the impact of error propagation depending on the feedback template weights. Figure 7.4(a) shows the error propagation behavior of hole filling template. It is observed that with increasing number of

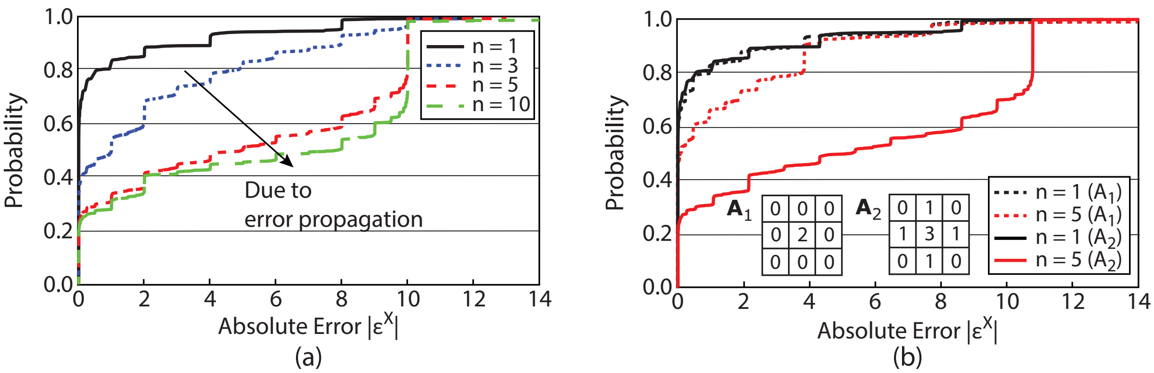


Figure 7.4: Impact of the feedback template $\mathbf{A}$ on error propagation by looking at empirical CDF with P_b = 5%: (a) detailed trajectory of error propagation for hole filling algorithm and (b) comparison of the error propagation between two different templates ($\mathbf{A_1}$: edge detection and $\mathbf{A_2}$: hole filling).

iterations, there is higher probability of having larger error showing error propagation as expected from (7.14). Figure 7.4(b) compares the error propagation characteristics between two applications to study the effect of feedback template on the error characteristics. Note empirical CDF at $n = 1$ is identical between two applications, since the generated bit error e^g at $n = 1$ is determined mainly by the characteristics of the hardware. Since the same processing engine for both applications is used, the identical error distribution is expected. However, there is higher probability of having larger error with template $\mathbf{A_2}$ than with $\mathbf{A_1}$ at $n = 5$ (Figure 7.4(b)). This verifies that larger feedback weights accelerate error propagation as expected from equation (7.14).

Finally, the effect of template weights and bit error (P_b) on convergence time is studied considering the templates $\mathbf{A_1}$ and $\mathbf{A_2}$. Due to larger fluctuations in the state value with increased P_b, the (relaxed) convergence time increases and may not converge (Figure 7.5(a)). Also, for image processing under performance constraints, we need to compare output quality after a fixed number of iterations. The edge detection algorithm is selected ($\mathbf{A_1}$) for illustration. Structural similarity (SSIM) index [195] is used to estimate the output image quality with error. For the baseline, the maximum number of iterations is fixed to 20 (it takes 3 iterations with no error). SSIM falls below 0.8 when the P_b becomes larger than

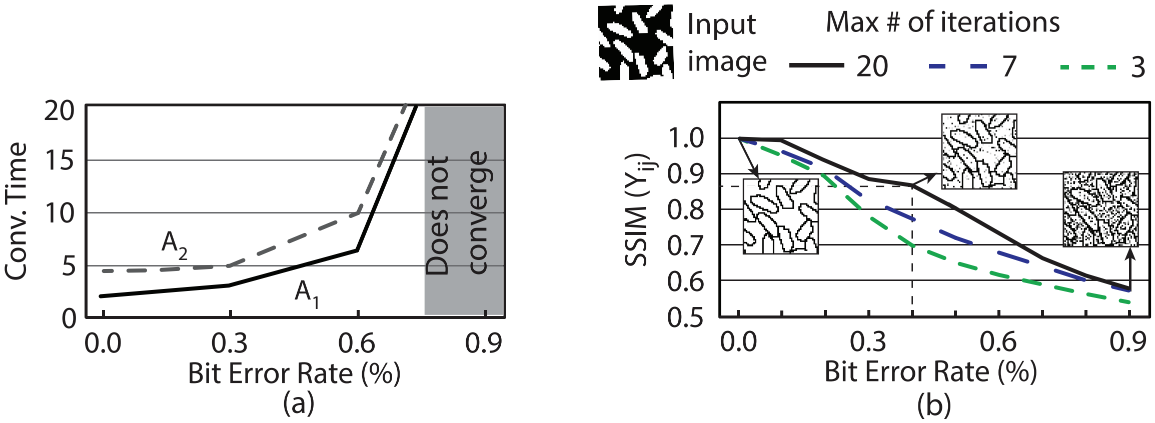


Figure 7.5: (a) Convergence time increase due to error and (b) accuracy degradation in terms of SSIM of the CeNN output when number of iterations is fixed for real-time CeNN under different P_b's.

0.5% (black line in Figure 7.5(b)). The output images at some P_b's (0, 0.4, and 0.9%) are shown in Figure 7.5(b). It is evident that increasing bit error rate increases the computed standard deviation of the error magnitude as well. In other words, a higher standard deviation of the error magnitude results in more convergence time and less image quality.

7.3 Design of a Digital CeNN

A CeNN cell is designed in 130nm CMOS for experimental analysis of energy-accuracy tradeoff. Two approaches are considered for power scaling, namely, voltage scaling and reduced precision. In this analysis, reduced precision is referred to as reducing precision bit-width as well as voltage but ensuring zero bit errors. The voltage over scaling (VOS) is referred to as reducing voltage below a critical limit that generates random bit errors due to timing violations in the critical path. During VOS the supply voltage is controlled to reach a finite bit error rate.

7.3.1 Hardware Design of CeNN

A digital CeNN cell, defined by equation (7.8), includes (i) a node consisting of storage elements for state (x_{ij}), input (u_{ij}), and output (y_{ij}); and (ii) a processing element (PE) that

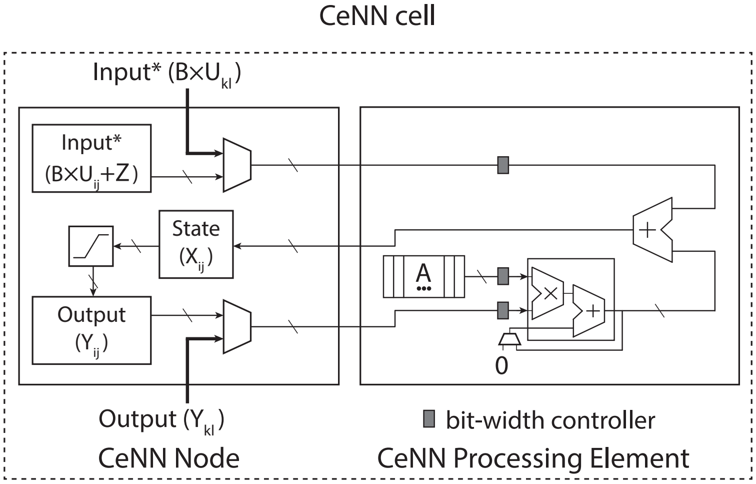


Figure 7.6: Digital CeNN implementation of a single cell consists of a CeNN node and a processing element (PE).

executes the multiplication and addition. Figure 7.6 illustrates a block diagram of a CeNN cell showing the node and the PE. In the CeNN node, there are three registers for input, output, and state respectively and a clamping module for the output function. In the CeNN node, the bold lines represent the propagated input (u_{kl}) and output (y_{kl}) from neighbor cells. The node delivers its input and output or the propagated input and output from the neighbor cells to its PE (data controller).

By using the data given by the CeNN node, a PE connected to each node computes the next state, $x_{ij}(n+1)$. A simple step-by-step diagram of PE computation is shown in Figure 7.7. To complete the computation of spatial convolution using a 3×3 template using a Multiply-Accumulate unit (MAC), a PE needs nine clock cycles. In Figure 7.7, each $\mathrm{T_i}$ represents this nine clock cycles. Since ($\sum_{C(k,l)\in N_r(i,j)} B_{kl} \cdot u_{kl} + z$) is a constant over time, it is pre-computed and stored in an input register file as a modified input (Input* in Figure 7.6). By doing so, only one MAC unit is required in a PE design reducing the area and power overhead. This pre-computation step requires nine clock cycles before running the application ($\mathrm{T_1}$ in Figure 7.7). After the pre-computation of input phase, template **A** is programmed in the PE since the template is assumed to be spatially invariant. Then, each step of CeNN state computation is performed until it reaches the iteration limit n.

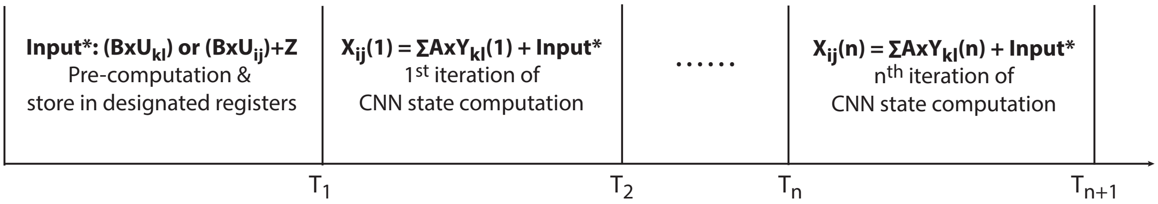

Input*: (BxU_kl) or (BxU_ij)+Z
Pre-computation &
store in designated registers
X_ij(1) = ΣAxY_kl(1) + Input*
1st iteration of
CNN state computation
......
X_ij(n) = ΣAxY_kl(n) + Input*
nth iteration of
CNN state computation
T1
T2
Tn
Tn+1

Figure 7.7: Simple timing diagram showing CeNN computation steps using the hardware with one ALU as shown in Figure 7.6.

The input and the output of CeNN are scaled between -1 and 1 while the values of template could be any non-integer values [188]. The baseline CeNN cell considers 12-bit signed fixed point representation (e.g. $Q_{3,8}$: 1-bit sign, 3-bit integer, 8-bit fractional number) to reduce the overhead of an arithmetic logic unit (ALU). Although the fixed-point number system has a limited range that may cause overflow, we should note that the input and the output of CeNN are always between -1 and +1 preventing the overflow of the state from changing the output if that limited range is able to represent [-1,1]. It will be shown that the 12-bit fixed-point representation results in only a minor quality degradation compared to the ideal (floating-point) representation; however, significantly reduces the hardware complexity.

Reducing data precision can reduce the power dissipation and delay, since it decreases the number of transitions in the ALU. However, a reduced precision also implies a higher numerical error. Therefore, while reduced precision is useful for low-power, as observed in Section 7.2.3, it is important to be able to change to higher precision mode for stronger feedback templates. The entire CeNN operates at the same precision level during the operation of an application (fixed precision control). However, the precision can be controlled from one application to the other. This is done by designing the CeNN cell such that some of LSBs can be selectively set to zero. For example, $Q_{3,2,6}$ means that 3 bits are integer part, 2 bits are fractional part, and 6 bits are zero (i.e. 6-bit precision). This is achieved by placing bit-width controllers before the inputs of a MAC and the input of a 2-input adder

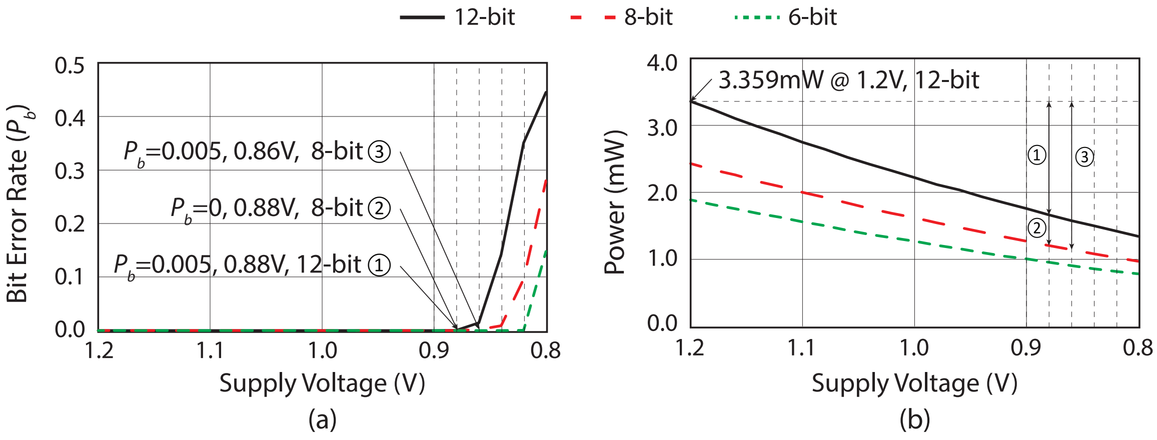
(Figure 7.6). We should note that truncating LSBs of integer part may degrade CeNN accuracy since values between -1 and 1 are important in CeNN.

The variable supply voltage of the CeNN cells (nodes and PEs) is also considered to reduce the power dissipation. As reducing voltage increases the delay of the critical paths within the PE, for a fixed frequency operation, reducing voltage beyond the critical limit leads to random bit errors due to timing violations. This is referred to as the voltage over scaling (VOS). As discussed in Section 7.2.3, the bit errors modulate output quality and convergence. Further, the effects of bit-errors due to voltage scaling also depends on templates.

7.3.2 Power and Bit Error Rate Analysis of Digital CeNN

The designed CeNN cell is simulated using SPICE to estimate the power, delay, and bit error rate at different voltages and precisions. Note in this study only the precision of the operands is reduced (reduces power in the PEs), the precision of the template is kept fixed at 12-bit. The clock frequency and convergence time (number of iterations) are kept constant for all analysis to guarantee same performance under all conditions. The maximum delay of the CeNN node is 1.18ns and that of the CeNN PE is 2.47ns at 1.2V with 12-bit precision. Power dissipation is also simulated using 3ns clock period and random input vectors (at **A**, **B**, offset z, and image input u_{ij}). The CeNN node consumes 392.45μW while the PE consumes 2.97mW at 1.2V with 12-bit precision.

When LSBs are set to zero (reduced precision), the maximum signal propagation length in the ALU reduces and hence, the maximum delay in ALU decreases. Therefore, for a target P_b, reducing the precision allows more headroom on voltage scaling. In addition, reducing precision also reduces the number of transitions in ALU and hence, the switching power. Figure 7.8 shows P_b and power dissipation at different supply voltages from 0.8V to 1.2V with 12, 8, and 6-bit precisions. Figure 7.8(a) shows that P_b sharply increases (by about 0.1/30mV) when the supply voltage is below the critical threshold (V_{crit}). As



(a) (b)

① 1.665mW (-51%), 0.7647 ② 1.204mW (-64%), 0.9382 ③ 1.144mW (-66%), 0.7451

Figure 7.8: (a) Bit error rate (P_b) at different supply voltages and precision bit-widths and (b) the corresponding power dissipation. The various options for energy-accuracy trade are shown with selected target P_b.

expected, V_{crit} for 6-bit or 8-bit precision is lower than that of 12-bit precision. For example, V_{crit} of $Q_{3,8,0}$ (12-bit) to meet $P_b = 0.005$ is 0.88V while it is 0.86V for $Q_{3,4,4}$ (8-bit). Reducing precision from 12-bit to 8-bit results in 34% change in power at a given V_{DD} (=1.2V).

7.4 Energy-Accuracy Tradeoff in CeNN

7.4.1 Analysis of Image Quality

Now, let's consider the role of reduced precision and higher P_b on the image quality. Table 7.1 shows SSIM of edge detection results on grayscale image using different bit precisions. The input image is assumed to be 12-bit pixels. Also, in this analysis the templates are assumed to be 12-bit (full precision). In contrast to black-and-white image, a grayscale image needs finer precision to represent 256 levels between -1 and 1.

Then, the error characteristics considering reduced precision versus random bit error are compared. To characterize the quantization error due to reduced precision, the 8-bit CeNN operation is compared with 12-bit operation. The state and output errors are computed by

Table 7.1: SSIM Index by Running an Edge Detection Algorithm with Different Precision Bit-width

SSIM	12-bit ($Q_{3,8,0}$)	8-bit ($Q_{3,4,4}$)	6-bit ($Q_{3,2,6}$)
No error	0.9987	0.9382	0.7669
$P_b = 0.5\%$	0.7647	0.7451	0.5949

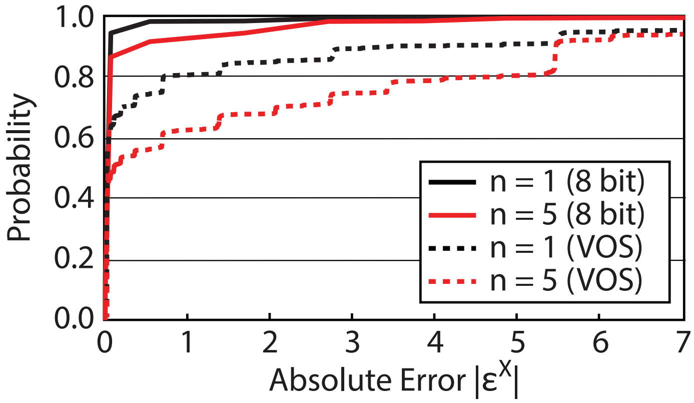


Figure 7.9: The comparison of empirical CDF of the absolute state error between reduced precision and voltage over scaling (VOS).

subtracting the state (output) of CeNN with 8-bit precision to that with 12-bit. The error distribution due to quantization is studied (not shown for brevity) for $n = 1$ and $n = 5$ to verify the prediction. As expected, it is observed the error is mostly concentrated near zero and the (numerical) standard deviation of the error is 0.02 at $n = 1$. This can be considered as e^g in (7.8). As noted in Section 7.2.3, the standard deviation of e^g for 5% bit error rate is about 2.05. Therefore, a smaller error rate is expected with reduced precision than VOS. Figure 7.9 shows the empirical CDF of absolute state error. As shown in Figure 7.9, the amount of error is initially larger for VOS due to the nature of error (expected by higher standard deviation). Moreover, error is generated in each step [i.e. $e^g_{ij}(n)$ in (7.14)] and propagated from neighbor cells. Therefore, it is concluded that reducing precision introduce less error than increasing bit-error-rate.

Table 7.1 shows the quality degradation with reduced precision and random bit errors. It shows that using 8-bit instead of 12-bit slightly degrades the image quality while 6-bit precision degrades the image quality significantly. Considering voltage scaling and random bit error, both 12-bit and 8-bit precisions still show SSIM higher than 0.7 (P_b = 0.5%) while 6-bit precision has SSIM less than 0.6. It means that finer precision smaller than 2^{-2} is desired to represent fractional number during the CeNN operation.

7.4.2 Energy-Accuracy Tradeoff

The VOS and the reduced precision are considered to evaluate potential of energy-accuracy tradeoff in CNN (Figure 7.8). The analysis always assumes full (12-bit) precision for the templates, while the precision for the operands are changed. First, VOS is applied while keeping the 12-bit precision to achieve a target P_b (= 0.5%) (①). Second, precision is reduced to 8-bit and voltage is scaled (from 1.2V to 0.88V) while maintaining P_b = 0 (②). Third, VOS is applied at reduced precision (8-bit) to meet the target P_b (= 0.5%) (③). It is observed that 8-bit precision operating at critical voltage ($\sim$0.88V) but without error gives 64% power saving from the baseline (1.2V with 12-bit precision) while maintaining SSIM $>$ 0.9. The above analysis shows that energy-accuracy tradeoff by reduced precision is more effective. This is because, even a small P_b due to VOS can result in a wider range of numerical error due to potential of flipping in MSBs.

In Figure 7.10, energy dissipation versus image quality is shown using two representative image processing applications, edge detection (weak feedback) and hole filling (strong feedback). The x-axis in the figure represents the normalized energy dissipation of the CeNN PE. Figure 7.10 shows that the energy can be dynamically scaled for the CeNN but at the expense of quality-of-results. There are two choices to scale energy: i) reducing precision and ii) scaling supply voltage. For the same amount of change in energy (e.g. from 1.00 to 0.46) reducing precision gives more accurate output image than scaling voltage. Moreover, the SSIM index of hole filling application falls more sharply compared to edge detection case when voltage falls below the critical point. This again shows that degradation in image quality for equivalent change in supply voltage and hence, energy dissipation, depends on the application characteristics.

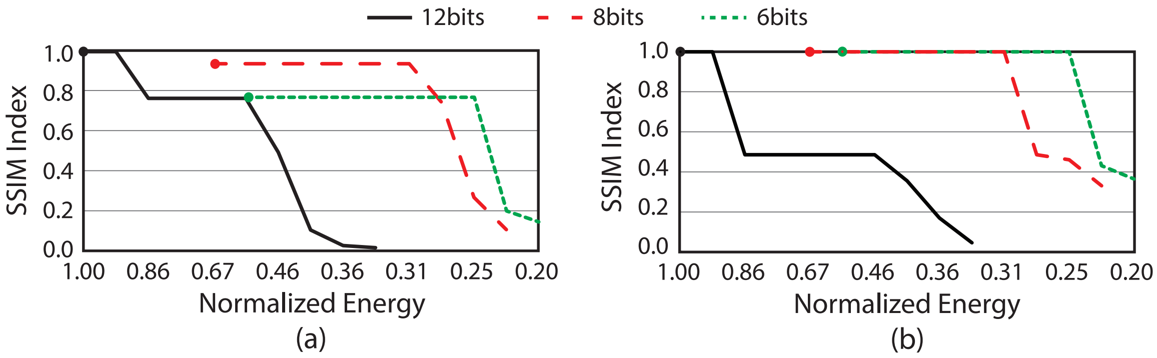


Figure 7.10: Energy-accuracy tradeoff plot using (a) edge detection and (b) hole filling applications.

7.5 Impact of Applications

7.5.1 Simple Image Processing Applications

In this section, the role of application is analyzed (i.e. depending on strength of the template **A**) on the energy-accuracy tradeoff. Only 12-bit and 8-bit are considered for illustration. Six different applications are simulated: edge detection, diffusion, halftoning, binarization, rotation detector, and hole filling [196]. To investigate the tradeoff between energy saving and image quality, three different options described in Figure 7.8 are used: 12-bit at 0.88V (P_b = 0.5%), 8-bit at 0.88V (P_b = 0), and 8-bit at 0.86V (P_b = 0.5%). Each result is compared to the image obtained by using an ideal (64-bit) floating point representation (Figure 7.11). The 64-bit floating point is set as reference in order to show 12-bit at 1.2V is good enough as a baseline design of the digital CeNN. As the number of iterations and the clock period are fixed, the performances of the CeNN under all of the above scenarios are constant.

Edge detection, which has zero neighbor feedback weights, generates the reasonable image quality even with both VOS and reduced precision (66% saving by ③). The degradation of image quality, however, is visible with non-zero feedback weights. Since diffusion, halftoning, and binarization have weaker feedback than rotation detector and hole filling,

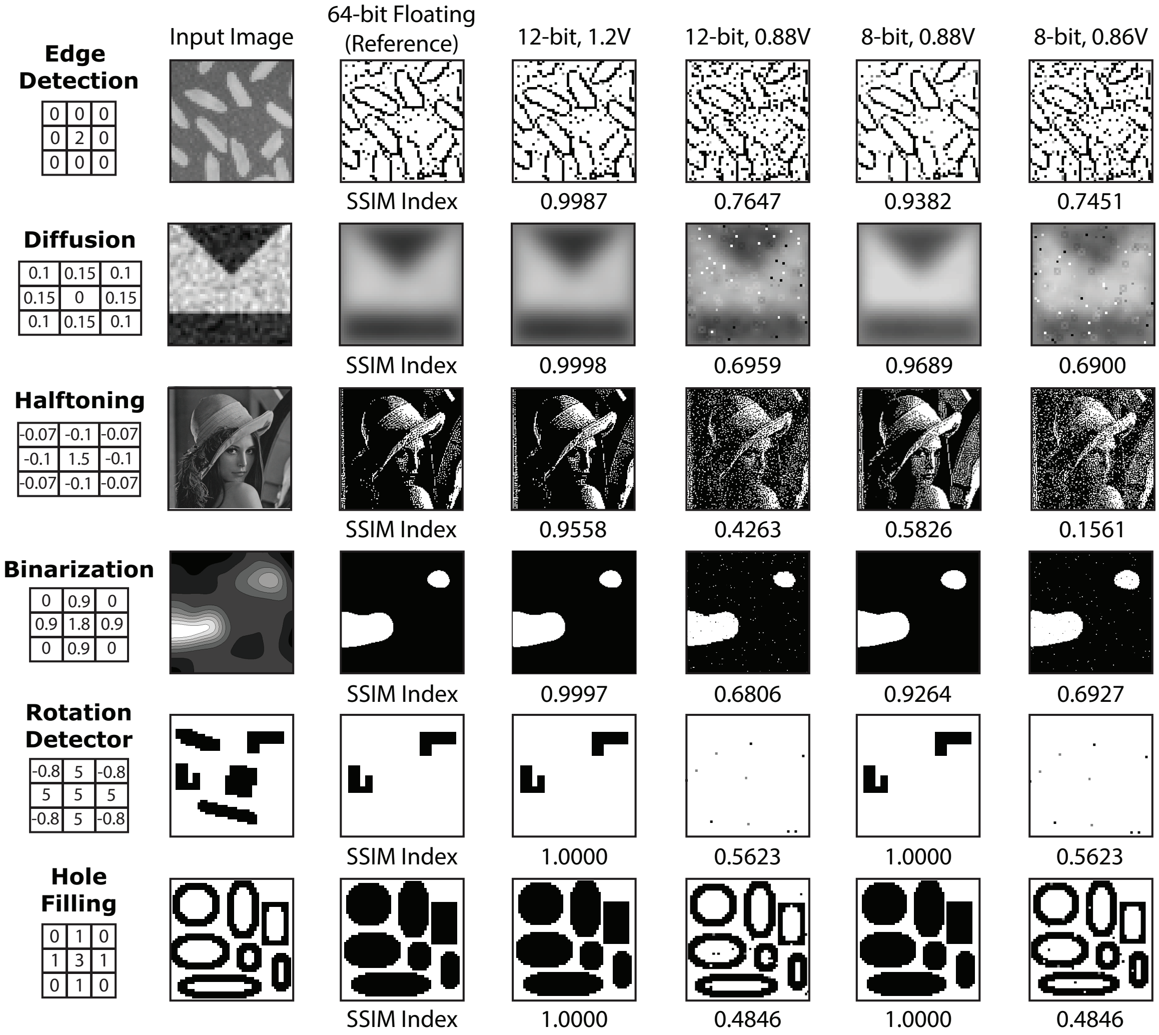


Figure 7.11: The impact of voltage scaling and precision control on output image quality for various applications.

those applications with 0.5% error rate still show the recognizable output images with salt and pepper noise while rotation detector and hole filling cannot show any desired output image. Hence, only precision control can be utilized for algorithms with strong feedback such as rotation detector and hole filling (power saving is limited; 64% saving by ②). For the diffusion or binarization algorithm, both VOS and precision control can be utilized depending on the error tolerance of the application they are involved in.

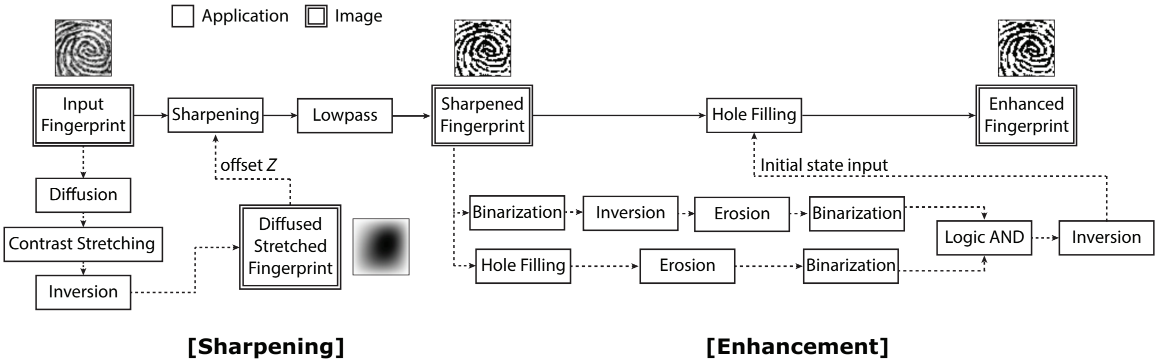


Figure 7.12: Overall block diagram of the fingerprint preprocessing application [197], which can be divided into two main algorithms (sharpening and enhancement). Solid lines represent the main flow while dotted lines represent the sub-flow of the application.

7.5.2 Case Study: Fingerprint Preprocessing Application

As shown in Figure 7.11, errors propagate and degrade the output quality significantly on some applications. Thus, VOS should be utilized only for applications with weak neighbor feedback or less number of iterations. In this subsection, a fingerprint preprocessing application [197] is tested, which consists of multiple image processing algorithms. The preprocessing (enhancement) is a crucial step before doing the fingerprint recognition. The fingerprint recognition system is important for the security and is used in various mobile platforms for biometric analysis [198]. The opportunities for energy-accuracy tradeoff are studied in the different intermediate tasks considering reduced precision for all tasks and voltage over scaling (along with reduced precision) for tasks with weak feedback weights. This goal is to understand whether reduced precision can maintain application quality and what is opportunity for additional energy-quality tradeoff with VOS for certain tasks.

Overall block diagram of the fingerprint preprocessing is shown in Figure 7.12. There are two main algorithms; sharpening and enhancement. The sharpening stage removes noise or shaded regions in the input image. The enhancement stage fills up the hole in important fingerprint lines. In detail, there are nine different image processing algorithms in use; sharpening, lowpass filtering, diffusion, contrast stretching, inversion, hole filling,

Allow Precision Control Only		Allow Voltage & Precision Control	
Hole Filling	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	Diffusion	$\begin{bmatrix} 0.1 & 0.15 & 0.1 \\ 0.15 & 0 & 0.15 \\ 0.1 & 0.15 & 0.1 \end{bmatrix}$
Sharpening	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	Contrast Stretching	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Lowpass	$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$	Binarization	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Erosion	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Inversion	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
		Logical AND	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Figure 7.13: Allocation of energy reduction schemes for each algorithm in the fingerprint preprocessing application depending on the strength of feedback template **A**.

binarization, erosion, and logic AND. As observed, hole filling has strong feedback which makes the application prone to bit flip error caused by VOS (refer to Section 7.5.1). In addition, sharpening, lowpass filtering, and erosion also have strong feedback templates as shown in Figure 7.13. Other than these four algorithms, weights of feedback templates are relatively weak or zero so that we could allow VOS and precision control.

Figure 7.14(c) and (d) show enhanced fingerprints when 12-bit and 8-bit are used, respectively. SSIM index higher than 0.9 indicates precision control maintains quality-of-results. The baseline energy consumption is computed when 12-bit is used. The fingerprint preprocessing requires total 727 iterations of CeNN operation and each iterations needs nine clock cycles to complete the convolution computation, which makes 6,543 clock cycles. Considering that clock cycle is 3ns, total energy consumption of a single CeNN PE required for fingerprint recognition with 12-bit precision at 1.2V is 6,543cycles × 3ns/cycle × 3.36mW = 65.95nJ. When 8-bit precision is used, 23.63nJ (64.17% reduction in energy consumption) is required for the entire computation.

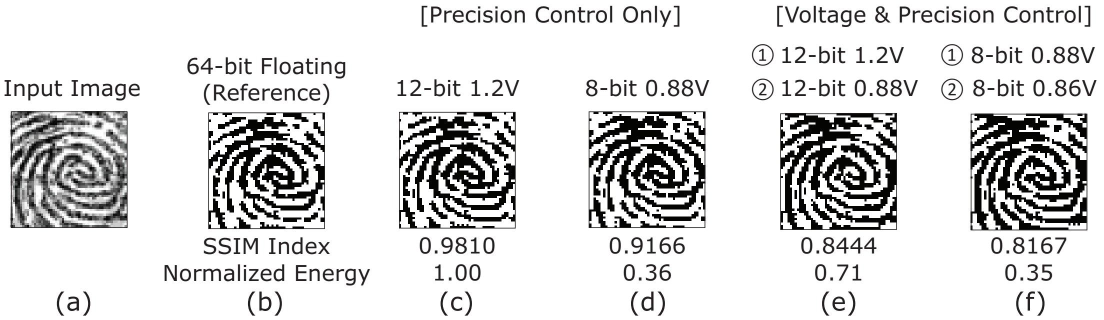


Figure 7.14: (a) Input fingerprint image and enhanced fingerprint images with (b) 64-bit floating point representation, (c) 12-bit precision without error, (d) 8-bit precision without error, (e) 12-bit precision with 0.5% bit error, and (f) 8-bit precision with 0.5% bit error.

The voltage over scaling could be allowed (0.5% bit error rate) on some algorithms with weak feedback template (on the right side of Figure 7.13) for more energy savings. Other than hole filling, sharpening, lowpass filtering, and erosion, the remaining image processing algorithms are simulated with 0.5% bit error. As a result, Figure 7.14(e) and (f) are obtained by using 12-bit and 8-bit precision, respectively. Then, 304 iterations are computed with no error while the remaining 423 iterations are computed with 0.5% bit error. Although, major connections are still recognized, the SSIM drops to 0.8. With VOS, energy consumption becomes 46.59nJ with 12-bit precision (Figure 7.14(e)) and 22.95nJ with 8-bit precision (Figure 7.14(f)). If we compare against the 12-bit implementation without VOS (i.e. Figure 7.14(c)), adding VOS to selective tasks saves 29.36% energy reduction. However, if we compare against the reduced precision (8-bit) without VOS, the additional energy saving by allowing error is negligible, but an appreciable degradation in image quality is observed.

7.6 Discussions

7.6.1 Effect of Truncation in the Templates

The discussions in Section 7.5 considered templates are always kept at 12-bit and with no error. To investigate the role of truncation in the templates, we have simulated three dif-

[Halftoning Template]

A

-0.07	-0.1	-0.07
-0.1	1.5	-0.1
-0.07	-0.1	-0.07

B

0.07	0.1	0.07
0.1	0.32	0.1
0.07	0.1	0.07

12 bit

-0.0703	-0.0996	-0.0703
-0.0996	1.5	-0.0996
-0.0703	-0.0996	-0.0703

0.0703	0.0996	0.0703
0.0996	0.3203	0.0996
0.0703	0.0996	0.0703

8 bit

-0.0625	-0.0938	-0.0625
-0.0938	1.5	-0.0938
-0.0625	-0.0938	-0.0625

0.0625	0.0938	0.0625
0.0938	0.3125	0.0938
0.0625	0.0938	0.0625

(a)

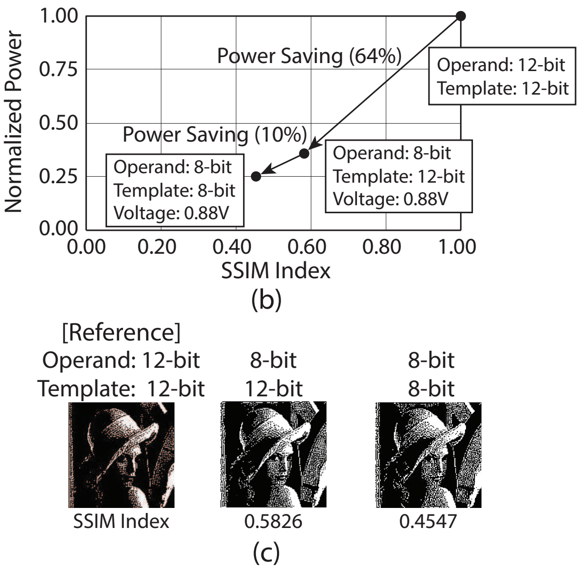


Figure 7.15: Impact of template truncation on the output quality of CeNN considering the halftoning example: (a) the halftoning templates depending on the precision bit-width, (b) the power versus quality for template and operand truncation, and (c) images comparing SSIM index with template truncation.

ferent cases: 12-bit operands with 12-bit templates (baseline), 8-bit operands with 12-bit templates (the operand truncation), and 8-bit operands with 8-bit templates (both template and operand truncation). In all cases the designs are simulated to operate at minimum voltage without a timing error. The operands are the state, the output, and the input of CeNN. The template values of feedback (**A**) and feedforward (**B**) considering 12-bit and 8-bit resolution are shown in Figure 7.15(a). The output using 12-bit operands with 12-bit templates is the baseline for the comparison purpose. Compared to the 8-bit operand and 12-bit template (SSIM ∼0.58), using 8-bit template degrades quality (SSIM ∼0.45). On the other hand, for a constant 8-bit operand, using the 8-bit truncated templates provides 10% additional energy saving compared to the 12-bit template case as illustrated in Figure 7.15(b). Figure 7.15(c) shows output images for the three different cases. The above analysis shows that template truncation, along with operand truncation, can lead to additional energy saving at the expense of graceful degradation in quality.

Table 7.2: Simulation Results on Adaptive Precision Control

	12-bit	12 to 8	8 to 12	8-bit
Edge Detection	0.9987	0.996	0.9454	0.9382
Diffusion	0.9998	0.9946	0.9742	0.9382
Halftoning	0.9558	0.9549	0.5829	0.5826

7.6.2 Adaptive Precision Control

In this subsection, the potential of adaptive precision, i.e. changing the precision during the evolution of the network, will be discussed. To evaluate the opportunity, two cases of adaptive precision are simulated: (1) higher precision at the early stage and lower precision at the later stage of the network (12-bit at initial 5 iterations and 8-bit at the remaining iterations) and (2) lower precision at the early stage and higher precision at the later stage of the network (8-bit at initial 5 iterations and 12-bit at the remaining iterations).

Simulation results are shown in Table 7.2 considering edge detection, diffusion, and halftoning. Using higher bit precision at early stage of CeNN operation and using lower bit precision at the remaining iterations results in high accuracy with large energy savings. For the opposite case (8-bit to 12-bit), accuracy does not improve much from the 8-bit case even though 2/3 of the total iterations (15 iterations) used 12-bit. This is because small error propagates through the network when the state x_{ij} of CeNN is in the linear region ($|x_{ij}| < 1$) [see (7.17)] i.e. at the early stage of the dynamics. Hence, higher precision at early stage helps improve accuracy. The adaptive precision control (12-bit to 8-bit) saves 48.12% of energy compared to the 12-bit case while using 8-bit saves 64.16% with more quality degradation.

7.7 Summary of the Chapter

This chapter analyzes the error characteristic of CeNN based image processing to evaluate the potential of energy-accuracy tradeoff. The algorithmic analysis shows that errors directly modulate the cell dynamics and their impacts not only degrade the image quality but also affect the convergence time (performance). In particular, the effect of error is more prominent for applications with higher feedback weights as error propagates more strongly through the network. The algorithmic analysis has been coupled with circuit level power, performance, and error rate simulation of digital CeNN cells considering reduced precision and voltage scaling. It is observed that reduced precision provides a better energy-accuracy tradeoff than voltage over scaling, particularly, for applications with strong feedback template. Application dependent precision and voltage control was observed to be an effective approach to reduce CeNN power at target performance with graceful quality degradation.

CHAPTER 8

CONCLUSION

In this thesis, design algorithms and realization of energy-efficient hardware platforms for complex system learning are presented. Two different approaches are explored for the same purpose, model-based and data-driven learning. Each approach has its own advantages, thus, one needs to select a proper learning method depending on the type of the underlying system.

For the model-based approach, the method to efficiently understand thermal behavior of an IC is studied and related hardware components are evaluated. Post-silicon thermal analysis is a challenge as physical uncertainties are present during the chip fabrication and with different packaging. The cohesive method of frequency-domain analysis and interpolation is presented to simplify learning process. Some systems, however, are modeled by highly nonlinear functions making it difficult to simplify the mathematical model. For these systems, a time-consuming numerical simulation should be performed. Therefore, the challenge is to accelerate the simulation in hardware platforms. To tackle this, a programmable accelerator is presented for simulating coupled differential equations. This solver exploits cellular nonlinear network (CeNN) as a computing model supported by novel system architecture to handle nonlinear computations. As a result, the proposed system significantly improves performance in solving wide classes of differential equations by innovating system design techniques at different levels.

For those cases where large set of data can be collected, i.e. IoT-based systems, the data-driven approach such as deep learning can be a good candidate to understand an unknown system or solve a complex problem. As the number of computations exponentially increases with the size of the network in deep neural network (DNN), the energy-efficiency becomes a critical factor to utilize DNN as an embedded computing solution. Accordingly, the design algorithms for low-power NN accelerators suited for embedded platforms are presented. To achieve this, approximate computing is utilized during the inference so that the energy consumption of accelerators can be significantly reduced. The challenge in allowing approximation in NN algorithms is the reliability; maintaining high accuracy in the presence of approximation error. To tackle this challenge, the mathematical analysis on how hardware-induced error propagates through a feedback-type NN, called cellular nonlinear network (CeNN) is performed. For more widely accepted NNs, an automated algorithm which selects synapses with different approximation level is presented and is extended to dynamic precision control depending on time-varying inputs for RNN.

In conclusion, I believe that combining both approaches (*heterogeneous learning platform*), model- and data-driven methods, will allow better learning of complex systems or faster exploration of solution spaces for given differential equations with high efficiency. For instance, pollution in the water can be modeled by a set of transport equations. However, due to environmental variabilities, actual spread of wastewater cannot be accurately modeled. Thus, the model might be supported by analyzing real data points by capturing aerial images and processing with deep convolutional neural network. By doing this, we will be able to perform in-situ fine-tuning of parameters in the given mathematical model or even discover additional physical expressions for unkown phenomena. This simple example can be extended to wide classes of problems and it will be helpful to find impactful applications of the heterogeneous learning platform.

REFERENCES

[1] R. C. Jaeger and T. N. Blalock, *Microelectronic circuit design*. 2006.

[2] B. Fang, A. G. Kelkar, S. M. Joshi, and H. R. Pota, “Modeling, system identification, and control of acoustic-structure dynamics in 3-D enclosures,” *Control Engineering Practice*, vol. 12, no. 8, pp. 989–1004, 2004.

[3] M. Cho, W. Song, S. Yalamanchili, and S. Mukhopadhyay, “Thermal system identification (TSI): A methodology for post-silicon characterization and prediction of the transient thermal field in multicore chips,” in *IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2012, pp. 118–124.

[4] G. Louppe, K. Cho, C. Becot, and K. Crammer, “QCD-aware recursive neural networks for jet physics,” *Computing Research Repository (CoRR)*, vol. arXiv: 1702.00748v1, 2017.

[5] C. Evans, P. J. Fleming, D. C. Hill, J. P. Norton, I. Pratt, D. Rees, and K. Rodriguez-Vazquez, “Application of system identification techniques to aircraft gas turbine engines,” *Control Engineering Practice*, vol. 9, no. 2, pp. 135–148, 2001.

[6] M. Wu, S. V. David, and J. L. Gallant, “Complete functional charaterization of sensory neurons by system identification,” *Annual Review of Neuroscience*, vol. 29, pp. 477–505, 2006.

[7] K. D. Cole, J. V. Beck, A. Haji-Skeikh, and B. Litkouhi, *Heat conduction using Green's functions*. CRC Press, 2011.

[8] S. Kondo and T. Miura, “Reaction-diffusion model as a framework for understanding biological pattern formation,” *Science*, vol. 329, no. 5999, pp. 1616–1620, 2010.

[9] J. W. Grizzle, C. Chevallereau, R. W. Sinnet, and A. D. Ames, “Models, feedback control, and open problems of 3D bipedal robotic walking,” *Automatica*, vol. 50, pp. 1955–1988, 2014.

[10] S. A. Bortoff, “Path planning for UAVs,” in *Proceedings of the American Control Conference*, IEEE, 2000, pp. 364–368.

[11] M. H. Gutknecht, “A brief introduction to Krylov space methods for solving linear systems,” in *Frontiers of Computational Science: Proceedings of the International Symposium on Frontiers of Computational Science*, Y. Kaneda, H. Kawamura, and

M. Sasai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 53–62, ISBN: 978-3-540-46375-7.

[12] W. F. Ford and J. A. Pennline, “Accelerated convergence in Newton's method,” *SIAM Review*, vol. 38, no. 4, pp. 658–659, 1996.

[13] S. Chen, S. A. Billings, and P. M. Grant, “Non-linear system identification using neural networks,” *International Journal of Control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[14] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, “A 240 G-ops/s mobile coprocessor for deep neural networks,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2014, pp. 696–701.

[15] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, “Hardware accelerated convolutional neural networks for synthetic vision systems,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 257–260.

[16] Y. Hurmuzlu, F. Genot, and B. Brogliato, “Modeling, stability and control of biped robots - a general framework,” *Automatica*, vol. 40, pp. 1647–1664, 2004.

[17] B. Li and Z. Shao, “A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles,” *Knowledge-Based Systems*, vol. 86, pp. 11–20, 2015.

[18] T. Luukkonen, “Modelling and control of quadcopter,” Aalto University, Tech. Rep., 2011.

[19] F. Rothganger, C. D. James, and J. B. Aimone, “Computing with dynamical system,” *IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–3, 2016.

[20] S. Bandini, G. Mauri, G. Pavesi, and C. Simone, “Computing with a distributed reaction-diffusion model,” in *Machines, Computations, and Universality: 4th International Conference, MCU 2004, Saint Petersburg, Russia, September 21-24, 2004, Revised Selected Papers*, M. Margenstern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 93–103, ISBN: 978-3-540-31834-7.

[21] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, “Information processing capacity of dynamical systems,” *Scientific Reports*, vol. 2, 514 EP –, Jul. 2012.

[22] 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 Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

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

[24] P. C. Matthews, R. E. Mirollo, and S. H. Strogatz, "Dynamics of a large system of coupled nonlinear oscillators," *Physica D: Nonlinear Phenomena*, vol. 52, no. 2, pp. 293–331, 1991.

[25] D. E. Nikonov, G. Csaba, W. Porod, T. Shibata, D. Voils, D. Hammerstrom, I. A. Young, and G. I. Bourianoff, "Coupled-oscillator associative memory array operation for pattern recognition," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 1, pp. 85–93, 2015.

[26] N. Shukla, A. Parihar, M. Cotter, M. Barth, X. Li, N. Chandramoorthy, H. Paik, D. G. Schlom, V. Narayanan, A. Raychowdhury, and S. Datta, "Pairwise coupled hybrid vanadium dioxide-MOSFET (HVFET) oscillators for non-boolean associative computing," in *2014 IEEE International Electron Devices Meeting*, 2014, pp. 28.7.1–28.7.4, ISBN: 0163-1918.

[27] A. Parihar, N. Shukla, S. Datta, and A. Raychowdhury, "Computing with dynamical systems in the post-CMOS era," in *2016 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, 2016, pp. 110–111.

[28] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM Transactions on Graphics (TOG)*, ACM, vol. 22, 2003, pp. 908–916.

[29] W. R. D. Boyd III, "Massively parallel algorithms for method of characteristics neutral particle transport on shared memory computer architectures," PhD thesis, Massachusetts Institute of Technology, 2014.

[30] O. Storaasli, "Computing faster without CPUs: Scientific applications on a reconfigurable, FPGA-based hypercomputer," in *6th Military and Aerospace Programmable Logic Devices (MAPLD) Conference*, 2003.

[31] L. Zhuo and V. K. Prasanna, "High performance linear algebra operations on reconfigurable systems," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, IEEE Computer Society, 2005, pp. 1–12.

[32] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. S. Trew, A. McCormick, G. Smart, *et al.*, “Maxwell-a 64 FPGA supercomputer.,” *Adaptive Hardware Systems*, vol. 7, pp. 287–294, 2007.

[33] A. D. Little and A. C. Soudack, “On the analog computer solution of first-order partial differential equations,” *Mathematics and Computers in Simulation*, vol. 7, no. 4, pp. 190–194, 1965.

[34] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, “SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor,” *2008 IEEE International Joint Conference on Neural Networks*, pp. 2849–2856, 2008.

[35] L. O. Chua and L. Yang, “Cellular neural network: Theory,” *IEEE Trans. Circuits Syst*, vol. 35, pp. 1257–1272, 1988.

[36] L. O. Chua, M. Hasler, G. S. Moschytz, and J. Neirynck, “Autonomous cellular neural networks: A unified paradigm for pattern formation and active wave propagation,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 10, pp. 559–577, 1995.

[37] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, “Simulating nonlinear waves and partial differential equations via CNN–part i: Basic techniques,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 10, pp. 807–815, 1995.

[38] F. Gollas and R. Tetzlaff, “Modeling complex systems by reaction-diffusion cellular nonlinear networks with polynomial weight-functions,” in *2005 9th International Workshop on Cellular Neural Networks and Their Applications*, IEEE, 2005, pp. 227–231.

[39] A. Slavova and P. Zecca, “Complex behavior of polynomial FitzHugh–Nagumo cellular neural network model,” *Nonlinear Analysis: Real World Applications*, vol. 8, no. 4, pp. 1331–1340, 2007.

[40] A. C. B. Delbem, L. G. Correa, and L. Zhao, “Design of associative memories using cellular neural networks,” *Neurocomputing*, vol. 72, no. 10, pp. 2180–2188, 2009.

[41] B. Zineddin, Z. Wang, and X. Liu, “Cellular neural networks, the Navier–Stokes equation, and microarray image reconstruction,” *IEEE Transactions on Image Processing*, vol. 20, no. 11, pp. 3296–3301, 2011.

[42] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, “Two-dimensional compressible flow simulation on emulated digital CNN-UM,” in *2008 11th International Work-*

shop on Cellular Neural Networks and Their Applications, IEEE, 2008, pp. 169–174.

[43] J. C. Chedjou and K. Kyamakya, “A universal concept based on cellular neural networks for ultrafast and flexible solving of differential equations,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 4, pp. 749–762, 2015.

[44] S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo, “24-GOPS 4.5-digital cellular neural network for rapid visual attention in an object-recognition SoC,” *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 64–73, 2011.

[45] P. Kinget and M. S. J. Steyaert, “A programmable analog cellular neural network cmos chip for high speed image processing,” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 235–243, 1995.

[46] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L Carranza, E. Roca-Moreno, R. Carmona-Galán, F. Jiménez-Garrido, R. Domínguez-Castro, and S. E. Meana, “ACE16k: The third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 851–863, 2004.

[47] B. E. Shi and T. Luo, “Spatial pattern formation via reaction-diffusion dynamics in 32×32×4 CNN chip,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 939–947, 2004.

[48] S. S. Haykin, *Neural networks and learning machines*, 3rd. 2009, pp. 152–257.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

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

[51] H. Jaeger, “A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach,” German National Research Center for Information Technology, Tech. Rep., 2002.

[52] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[53] O. Karan, C. Bayraktar, H. Gumuskaya, and B. Karlik, “Diagnosing diabetes using neural networks on small mobile devices,” *Expert Systems with Applications*, vol. 39, no. 1, pp. 54–60, 2012.

[54] D. Janglova, "Neural networks in mobile robot motion," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 15–22, 2004.

[55] J. Jin, V. Gokhale, A. Dundar, B. Krishnamurthy, B. Martini, and E. Culurciello, "An efficient implementation of deep convolutional neural networks on a mobile coprocessor," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2014, pp. 133–136.

[56] F. Rosenblatt, "The perceptron-a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Tech. Rep. 85-460-1, 1957.

[57] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[58] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[59] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[60] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, Society for Artificial Intelligence and Statistics, 2010, pp. 249–256.

[61] L. Gong, C. Liu, Y. Li, and Y. Fuqing, "Training feed-forward neural networks using the gradient descent method with the optimal stepsize," *Journal of Computational Information Systems*, vol. 8, no. 4, pp. 1359–1371, 2012.

[62] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 1989, pp. 762–767.

[63] P. Koehn, "Combining genetic algorithms and neural networks: The encoding problem," Master's thesis, The University of Tennessee, Knoxville, 1994.

[64] A. R. M. Kattan, R. Abdullah, and R. A. Salam, "Training feed-forward neural networks using a parallel genetic algorithm with the best must survive strategy," in *Proc. International Conference on Intelligent Systems, Modelling and Simulation*, IEEE Computer Society, 2010, pp. 96–99, ISBN: 978-0-7695-3973-7.

[65] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu, "A hybrid particle swarm optimizationback-propagation algorithm for feedforward neural network training," *Applied Mathematics and Computation*, vol. 185, no. 2, pp. 1026–1037, 2007.

[66] S. Mirjalili, S. Z. M. Hashim, and H. M. Sardroudi, “Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm,” *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11 125–11 137, 2012.

[67] Y. V. Venkatesh and S. K. Raja, “On the classification of multispectral satellite images using the multilayer perceptron,” *Pattern Recognition*, vol. 36, no. 9, pp. 2161–2175, 2003.

[68] T. Raiko, H. Valpola, and Y. Lecun, “Deep learning made easier by linear transformations in perceptrons,” in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 22, 2012, pp. 924–932.

[69] A. J. Meade Jr. and A. A. Fernandez, “Solution of nonlinear ordinary differential equations by feedforward neural networks,” *Mathematical and Computer Modelling*, vol. 20, no. 9, pp. 19–44, 1994.

[70] L. P. Aarts and P. van der Veer, “Neural network method for solving partial differential equations,” *Neural Processing Letters*, vol. 14, no. 3, pp. 261–271, 2001.

[71] S. Ferrari and R. F. Stengel, “Smooth function approximation using neural networks,” *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

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

[73] G. V. Puskorius, L. A. Feldkamp, and L. I. Davis Jr., “Dynamic neural network methods applied to on-vehicle idle speed control,” *Proceedings of IEEE*, vol. 84, no. 10, pp. 1407–1420, 1996.

[74] H. T. Siegelmann and E. D. Sontag, “Turing computability with neural nets,” *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[75] R. J. Williams and J. Peng, “An efficient gradient-based algorithm for on-line training of recurrent network trajectories,” *Neural Computation*, vol. 2, no. 4, pp. 490–501, 1990.

[76] R. J. Williams and D. Zipser, “A learning algorithm for continually running fully recurrent neural networks,” *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.

[77] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[78] J. Martens and I. Sutskever, “Learning recurrent neural networks with Hessian-free optimization,” in *Proc. International Conference on Machine Learning (ICML)*, 2011.

[79] K. Funahashi and Y. Nakamura, “Approximation of dynamical systems by continuous time recurrent neural networks,” *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

[80] L. Jin, P. N. Nikiforuk, and M. M. Gupta, “Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks,” *IEEE Transactions on Automatic Control*, vol. 40, no. 7, pp. 1266–1270, 1995.

[81] X.-D. Li, J. K. L. Ho, and T. W. S. Chow, “Approximation of dynamical time-variant systems by continuous-time recurrent neural networks,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 10, pp. 656–660, 2005.

[82] P. Tino, B. G. Horne, C. L. Giles, and P. C. Collingwood, “Finite state machines and recurrent neural networks – automata and dynamical systems approaches,” in *Neural Networks and Pattern Recognition*, Academic Press, 1998, pp. 171–220.

[83] K. Arai and R. Nakano, “Stable behavior in a recurrent neural network for a finite state machine,” *Neural Networks*, vol. 13, no. 6, pp. 667–680, 2000.

[84] I. Sutskever, J. Martens, and G. Hinton, “Generating text with recurrent neural networks,” in *Proc. International Conference on Machine Learning (ICML)*, Omnipress, 2011, pp. 1017–1024.

[85] A. Graves, “Generating sequences with recurrent neural networks,” *Computing Research Repository (CoRR)*, vol. arXiv: 1308.0850, 2013.

[86] S. Miyoshi, H.-F. Yanai, and M. Okada, “Associative memory by recurrent neural networks with delay elements,” *Neural Networks*, vol. 17, no. 1, pp. 55–63, 2004.

[87] Z. Zeng and J. Wang, “Design and analysis of high-capacity associative memories based on a class of discrete-time recurrent neural networks,” *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 38, no. 6, pp. 1525–1536, 2008.

[88] M. Stevenson, R. Winter, and B. Widrow, “Sensitivity of feedforward neural networks to weight errors,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 71–80, 1990.

[89] A. F. Murray and P. J. Edwards, “Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training,” *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 792–802, 1994.

[90] S. Cavalieri and O. Mirabella, “A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks,” *Neural Networks*, vol. 12, no. 1, pp. 91–106, 1999.

[91] J. P.-F. Sum, C.-S. Leung, and K. I.-J. Ho, “On-line node fault injection training algorithm for MLP networks: Objective function and convergence analysis,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 2, pp. 211–222, 2012.

[92] K.-C. Jim, C. L. Giles, and B. G. Horne, “An analysis of noise in recurrent neural networks: Convergence and generalization,” *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1424–1438, 1996.

[93] Z. Wu, H. Su, J. Chu, and W. Zhou, “Improved delay-dependent stability condition of discrete recurrent neural networks with time-varying delays,” *IEEE Transactions on Neural Networks*, vol. 21, no. 4, pp. 692–697, 2010.

[94] Y. Shen and J. Wang, “Robustness analysis of global exponential stability of recurrent neural networks in the presence of time delays and random disturbances,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 87–96, 2011.

[95] J. Cao, “Global stability conditions for delayed CNNs,” *IEEE Transcations on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 48, no. 11, pp. 1330–1333, 2001.

[96] V. Singh, “Global robust stability of delayed neural networks: Estimating upper limit of norm of delayed connection weight matrix,” *Chaos, Solitons and Fractals*, vol. 32, no. 1, pp. 259–263, 2007.

[97] Y. He, M. Wu, and J.-H. She, “An improved global asymptotic stability criterion for delayed cellular neural networks,” *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 250–252, 2006.

[98] S. Arik, “New criteria for global robust stability of delayed neural networks with norm-bounded uncertainties,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 6, pp. 1045–1052, 2014.

[99] R. Yang, H. Gao, and P. Shi, “Novel robust stability criteria for stochastic Hopfield neural networks with time delays,” *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 39, no. 2, pp. 467–474, 2008.

[100] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, and S. Talay, “Large-scale FPGA-based convolutional networks,” in. in Machine Learning on Very Large Data Sets: Cambridge University Press, 2011.

[101] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2012, pp. 1044–1047.

[102] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. IEEE/ACM International Symposium on Microarchitecture (Micro)*, 2014, pp. 683–688.

[103] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 683–688.

[104] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. Great Lakes Symposium on VLSI*, ACM, 2015, pp. 199–204, ISBN: 978-1-4503-3474-7.

[105] 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 Conference (CICC)*, 2011, pp. 1–4.

[106] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

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

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

[109] Y. LeCun, F.-J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004, pp. 97–104.

[110] J. L. Holt and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 281–290, 1993.

[111] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 27–32, ISBN: 978-1-4503-2975-0.

[112] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *Computing Research Repository (CoRR)*, vol. abs/1502.02551, 2015.

[113] U. Lotric and P. Bulic, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing*, vol. 96, pp. 57–65, 2012.

[114] M. Horowitz, "Scaling, power and the future of CMOS," in *Proc. 20th International Conference on VLSI Design Held Jointly with 6th International Conference: Embedded Systems (VLSID)*, Washington, DC, USA: IEEE Computer Society, 2007, pp. 23–29.

[115] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, 1999.

[116] H. F. Hamann, A. Weger, J. A. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil, "Hotspot-limited microprocessors: Direct temperature and power distribution measurements," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 56–65, 2007.

[117] M. Janicki, J. H. Collet, A. Louri, and A. Napieralski, "Hot spots and core-to-core thermal coupling in future multi-core architectures," in *2010 26th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2010, pp. 205–210.

[118] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili, "Cooperative boosting: Needy versus greedy power management," in *Proc. 40th Annual International Symposium on Computer Architecture (ISCA)*, Tel-Aviv, Israel: ACM, 2013, pp. 285–296, ISBN: 978-1-4503-2079-5.

[119] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[120] S. Im and K. Banerjee, "Full chip thermal analysis of planar (2-D) and vertically integrated (3-D) high performance ICs," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2000, pp. 727–730.

[121] Y. Zhan and S. S. Sapatnekar, “High-efficiency green function-based thermal simulation algorithms,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1661–1675, 2007.

[122] J. Kung, I. Han, S. Sapatnekar, and Y. Shin, “Thermal signature: A simple yet accurate thermal index for floorplan optimization,” in *Proc. 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011, pp. 108–113.

[123] H. Chang and S. S. Sapatnekar, “Full-chip analysis of leakage power under process variations, including spatial correlations,” in *Proc. 42nd Design Automation Conference (DAC), 2005.*, 2005, pp. 523–528.

[124] “ASTM: Standard test method for thermal transmission properties of thermally conductive electrical insulation materials,” ASTM International, Tech. Rep. D 5470-12, 2012.

[125] A. Chowdhury, B. Guenin, C. Woo, S. Kim, and S. Lee, “The effect of die attach layer delamination on the thermal performance of plastic packages,” in *Proc. 48th Electronic Components and Technology Conference (ECTC)*, 1998, pp. 1140–1147.

[126] J. Abdul, Y. Wang, N. Guo, A. U. Rehman, and K. C. Chan, “Ultrasonic evaluation of the silicon/copper interfaces in IC packaging,” *IEEE Transactions on Electronics Packaging Manufacturing (TEPM)*, vol. 26, no. 3, pp. 221–227, 2003.

[127] J. Long, S. O. Memik, G. Memik, and R. Mukherjee, “Thermal monitoring mechanisms for chip multiprocessors,” *ACM Trans. Archit. Code Optim.*, vol. 5, no. 2, 9:1–9:33, Sep. 2008.

[128] R. Cochran and S. Reda, “Spectral techniques for high-resolution thermal characterization with limited sensor data,” in *Proc. 46th ACM/IEEE Design Automation Conference*, 2009, pp. 478–483.

[129] Y. Zhang, A. Srivastava, and M. Zahran, “Chip level thermal profile estimation using on-chip temperature sensors,” in *Proc. IEEE International Conference on Computer Design (ICCD)*, 2008, pp. 432–437.

[130] S. Sharifi and T. S. Rosing, “Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 10, pp. 1586–1599, 2010.

[131] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, “An integrated quad-core opteron processor,” in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 102–103.

[132] J. Shor, K. Luria, and D. Zilberman, “Ratiometric BJT-based thermal sensor in 32nm and 22nm technologies,” in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2012, pp. 210–212.

[133] A. Nowroz, G. Woods, and S. Reda, “Improved post-silicon power modeling using AC lock-in techniques,” in *Proc. 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011, pp. 101–107.

[134] W. Yueh, K. Z. Ahmed, and S. Mukhopadhyay, “Field programmable thermal emulator (FPTE): An all-silicon test structure for thermal characterization of integrated circuits,” in *IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2014, pp. 66–71.

[135] J. Kung, M. Cho, S. Yalamanchili, and S. Mukhopadhyay, “On-line real-time temperature and power estimation of an IC using time-domain thermal filters,” in *Proc. IEEE Electrical Performance of Electronic Packaging and Systems (EPEPS)*, 2013, pp. 199–202.

[136] M. A. P. Pertijs, A. Niederkorn, X. Ma, B. McKillop, A. Bakker, and J. H. Huijsing, “A CMOS smart temperature sensor with a 3σ; inaccuracy of ±0.5°C from -50°C to 120°C,” *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 40, no. 2, pp. 454–461, 2005.

[137] P. Chen, C. C. Chen, T. K. Chen, and S. W. Chen, “A time domain mixed-mode temperature sensor with digital set-point programming,” in *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, 2006, pp. 821–824.

[138] T. Yang, S. Kim, P. R. Kinget, and M. Seok, “16.4 0.6-to-1.0V 279μm^2, 0.92μW temperature sensor with less than $+3.2/-3.4$°C error for on-chip dense thermal monitoring,” in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2014, pp. 282–283.

[139] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, “Modeling within-die spatial correlation effects for process-design co-optimization,” in *Proc. 6th International Symposium on Quality of Electronic Design (ISQED)*, Washington, DC, USA: IEEE Computer Society, 2005, pp. 516–521, ISBN: 0-7695-2301-3.

[140] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” in *Proc. 27th Annual International Symposium on Computer Architecture (ISCA)*, 2000, pp. 83–94.

[141] S. Gupta and F. N. Najm, “Power modeling for high-level power estimation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 18–29, 2000.

[142] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[143] Nangate, Sunnyvale, CA, *45nm open cell library*, `http://www.nangate.com/`.

[144] Synopsys, Mountain View, CA, *Design compiler user guide*, 2010.

[145] Cadence, San Jose, CA, *Encounter digital implementation system menu reference*, 2011.

[146] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, "Using Krylov methods in the solution of large-scale differential-algebraic systems," *SIAM Journal on Scientific Computing*, vol. 15, no. 6, pp. 1467–1488, 1994.

[147] T. Han, Y. Han, *et al.*, "Solving large scale nonlinear equations by a new ODE numerical integration method," *Applied Mathematics*, vol. 1, no. 03, p. 222, 2010.

[148] J. Michalakes and M. Vachharajani, "GPU acceleration of numerical weather prediction," *Parallel Processing Letters*, vol. 18, no. 4, pp. 531–548, 2008.

[149] K. M. Cuomo and A. V. Oppenheim, "Circuit implementation of synchronized chaos with applications to communications," *Physical Review Letters*, vol. 71, no. 1, p. 65, 1993.

[150] N. Foster and R. Fedkiw, "Practical animation of liquids," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, 2001, pp. 23–30.

[151] N Chakrabarti and G. Lakhina, "Collisional Rayleigh-Taylor instability and shear-flow in equatorial Spread-F plasma," *Annales Geophysicae*, vol. 21, pp. 1153–1157, 2003.

[152] L. Li, "A low order acceleration scheme for solving the neutron transport equation," Master's thesis, Massachusetts Institute of Technology, 2013.

[153] R. Bialecki, A. J. Kassab, and A. Fic, "Proper orthogonal decomposition and modal analysis for acceleration of transient FEM thermal analysis," *International Journal of Numerical Methods in Engineering*, vol. 62, no. 6, pp. 774–797, 2005.

[154] Z. Nagy, Z. Vörösházi, and P. Szolgay, "Emulated digital CNN-UM solution of partial differential equations," *International Journal of Circuit Theory and Applications*, vol. 34, no. 4, pp. 445–470, 2006.

[155] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 43, 2015, pp. 92–104.

[156] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, 2015, pp. 161–170.

[157] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, ACM, 2015, pp. 199–204.

[158] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, IEEE, 2013, pp. 13–19.

[159] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR, abs/1502.02551*, vol. 392, 2015.

[160] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Intl. Symp. on Computer Architecture (ISCA)*, IEEE, 2016.

[161] S. Bandini, G. Mauri, G. Pavesi, and C. Simone, "Computing with a distributed reaction-diffusion model," in *International Conference on Machines, Computations, and Universality*, Springer, 2004, pp. 93–103.

[162] A. L. Hodgkin and A. F. Huxley, "The dual effect of membrane potential on sodium conductance in the giant axon of Loligo," *The Journal of physiology*, vol. 116, no. 4, p. 497, 1952.

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

[164] Hybrid Memory Cube Consortium, *Hybrid memory cube specification 1.0*, 2013.

[165] *Nangate FreePDK15 Open Cell Library*, `http://www.nangate.com/?page_id=2328`.

[166] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI*, IEEE, 2014, pp. 290–295.

[167] Z. Vörösházi, A. Kiss, Z. Nagy, and P. Szolgay, "Implementation of embedded emulated-digital CNN-UM global analogic programming unit on FPGA and its application," *International Journal of Circuit Theory and Applications*, vol. 36, no. 5-6, pp. 589–603, 2008.

[168] Á. Rodríguez-Vázquez, R. Domínguez-Castro, F. Jiménez-Garrido, S. Morillas, J. Listán, L. Alba, C. Utrera, S. Espejo, and R. Romay, "The Eye-RIS CMOS vision system," in *Analog circuit design*, Springer, 2008, pp. 15–32.

[169] A. Savich, M. Moussa, and S. Areibi, "A scalable pipelined architecture for real-time computation of MLP-BP neural networks," *Microprocessors and Microsystems*, vol. 36, no. 2, pp. 138–150, 2012.

[170] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–4.

[171] *MNIST database*, `http://yann.lecun.com/exdb/mnist/`.

[172] C. Szegedy, W. Liu, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Computing Research Repository (CoRR)*, vol. abs/1409.4842, 2014.

[173] Z. C. Lipton, D. C. Kale, C. Elkan, and R. C. Wetzel, "Learning to diagnose with LSTM recurrent neural networks," *Computing Research Repository (CoRR)*, vol. abs/1511.03677, 2015.

[174] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Proc. Second International Conference on Human Behavior Understanding*, ser. HBU'11, Amsterdam, The Netherlands, 2011, pp. 29–39, ISBN: 978-3-642-25445-1.

[175] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *Proc. International Conference on Machine Learning (ICML)*, 2015.

[176] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech and Language*, vol. 30, no. 1, pp. 61–98, 2015.

[177] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[178] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local SVM approach," in *Proc. 17th International Conference on Pattern Recognition (ICPR)*, vol. 3, 2004, pp. 32–36.

[179] A. Nagendran, D. Harper, and M. Shah, *New system performs persistent wide-area aerial surveillance*, SPIE Newsroom.

[180] M. Zhang and A. A. Sawchuk, "USC-HAD: A daily activity dataset for ubiquitous activity recognition using wearable sensors," in *Proc. ACM Conference on Ubiquitous Computing*, Pittsburgh, Pennsylvania: ACM, 2012, pp. 1036–1043, ISBN: 978-1-4503-1224-0.

[181] I. Laptev, "On space-time interest points," *Int. J. Computer Vision*, vol. 64, no. 2-3, pp. 107–123, Sep. 2005.

[182] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: New features and speed improvements," in *Deep Learning Workshop at Neural Information Processing Systems (NIPS)*, 2012.

[183] P. Szolgay, I. Szatmari, and K. Laszlo, "A fast fixed point learning method to implement associative memory on CNNs," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 44, no. 4, pp. 362–366, 1997.

[184] T. Kozek, T. Roska, and L. O. Chua, "Genetic algorithm for CNN template learning," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 6, pp. 392–402, 1993.

[185] M. Browne and S. S. Ghidary, "Convolutional neural networks for image processing: An application in robot vision," in *Australian Joint Conference on Artificial Intelligence*, 2003, pp. 641–652.

[186] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, Dec. 2010.

[187] 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 tracking," *IEEE Transactions on Neural Networks*, vol. 20, no. 9, pp. 1417–1438, 2009.

[188] L. O. Chua and L. Yang, "Cellular neural network: Theory," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257–1272, 1988.

[189] ——, "Cellular neural networks: Applications," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1273–1290, 1988.

[190] D. Feiden and R. Tetzlaff, “Cellular neural networks for motion estimation and obstacle detection,” *Advances in Radio Science*, vol. 1, pp. 143–147, 2003.

[191] R. Hegde and N. R. Shanbhag, “Soft digital signal processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 6, pp. 813–823, 2001.

[192] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar, “Reducing power by optimizing the necessary precision/range of floating-point arithmetic,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 273–286, 2000.

[193] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: Imprecise adders for low-power approximate computing,” in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414.

[194] L. O. Chua and T. Roska, *Cellular neural networks and visual computing: Foundations and applications*. 1st ed. Cambridge University Press, 2002, pp. 258–266.

[195] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[196] K. Karacs, G. Y. Cserey, A. Zarandy, P. Szolgay, C. S. Rekeczky, L. Kek, V. Szabo, G. Pazienza, and T. Roska. (2010). Software library for cellular wave computing engines.

[197] Q. Gao, P. Forster, K. R. Mobus, and G. S. Moschytz, “Fingerprint recognition using CNNs: Fingerprint preprocessing,” in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, 2001, pp. 433–436.

[198] R. Bolle and S. Pankanti, *Biometrics, personal identification in networked society: Personal identification in networked society*, A. K. Jain, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 1998, ISBN: 0792383451.