Chair of Electronic Design Automation

Design and Optimization of Emerging Systems for Biochemical Experiments and Neuromorphic Computing

Ying Zhu

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs
genehmigten Dissertation.

Vorsitzende/-r: Prof. Dr. Bernhard Wolfrum

Prüfende/-r der Dissertation:

1. Prof. Dr.-Ing Ulf Schlichtmann

2. Prof. Dr. Tsung-Yi Ho

Die Dissertation wurde am 02.12.2020 bei der Technischen Universität München
eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am
19.04.2021 angenommen.

Acknowledgments

Firstly, I would like to thank the International Graduate School of Science and Engineer (IGSSE), Technical University of Munich (TUM), which offered me the funding to proceed my research and multiple opportunities to join in the science and technology community.

Secondly but most importantly, I would like to express my sincere gratitude to my supervisor Prof. Ulf Schlichtmann. Thanks for the opportunity he offered to me so that I could research at the Chair of Electronic Design Automation (EDA) at TUM. In the past three years, he gave me precious instructions to improve my working skills and the ability for critical thinking. His serious attitude as a scientist also showed me a role model. Without his supports, I couldn't have the opportunities to learn from and work with many international outstanding researchers.

I would like to appreciate the effort and time from Prof. Bernhard Wolfrum and Prof. Tsung-Yi Ho who serve as the committee members.

I would like to express my sincere gratitude to Dr. Bing Li whom I closely cooperate with on the research topics - biochips and neuromorphic computing. As a senior researcher, he gave me a lot of invaluable advice on my research and career. The advice not only focused on the research topics we worked on but helped me learn how to discover cutting-edge research topics, develop novel ideas, design experiments, as well as present the research work, which inspired me to grow up as an independent researcher step by step. He also motivated me when I got depressed.

I appreciate the uncountable helps and supports in both research and life from Dr. Grace Li Zhang. Her devotion to research impressed me very much and her professionalism is my role model.

Simultaneously, due to the international cooperation, I truly appreciated the invaluable suggestions from Prof. Cheng Zhuo, Prof. Xunzhao Yin, Prof. Huaxi Gu, Prof. Yiyu Shi, Prof. Hailong Yao, Prof. Tsung-Yi Ho, and Prof. Robert Wille about our collaboration projects.

Furthermore, I would like to thank Dr. Helmut Graeb, Dr. Daniel Mueller-Gritschneder, Susanne Werner, Hans Ranke, Tobias Baur and other colleagues at the Chair of EDA, as well as Dr. Katrin Offe, Dr. Arne Schieder, Dr. Marco Barden, Jo-Anna Kuester and Sandra Schoengen with IGSSE for their kindly and patient support.

Last but not least, I give my deepest gratitude to my parents for their endless love.

Munich, May 2020

Ying Zhu

Contents

Chapter 1

Introduction

Interest has grown in recent years in using engineering to answer questions such as how humankind originated and evolved, why and how humankind acquired consciousness, and whether consciousness can be mimicked or even created. To answer these questions, researchers have long explored in fields ranging from traditional biological and chemical engineering to mathematics and computer science.

In several decades there has been much progress in biological sciences, ranging from genetic testing and pharmacogenomics to analysis and exploration of metabolism, enzyme and tissues, in which a large number of biochemical experiments are necessary. These experiments have been conducted using cumbersome devices such as tubes and droppers in traditional biochemical laboratories. This approach is complex and error-prone due to the need for human intervention. To improve the execution efficiency of experiments, the community of researchers and practitioners working in and around biochemistry has achieved a remarkable advancement by providing system-in-a-package solutions, where experiments can be completely performed within a compact system. Although this system integration has significantly improved experimental efficiency compared with traditional biochemical laboratories, only relatively simple experimental protocols can be processed automatically, and complex biochemical experiments, such as exhaustive diagnosis of diseases, still require human intervention [HCH17].

Humans are also using engineering to explore how the brain works. Early in

1960s, to mimic the nervous systems, scientists proposed artificial neural networks (ANNs) with the introduction of computers [Sch15,IL73]. However, due to the limited computing performance, ANNs were not commonly researched until the 21st Century when the integrated circuits were able to provide sufficient computing capacity. To further improve computing efficiency, engineers are developing central processing units (CPUs), graphic processing units (GPUs), field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) which can be enhanced by multicore computing, memory distribution optimization, and more. Even so, ANNs are still outperformed by human brains which consume less energy per operation and lower area costs [CMP13], because these computing systems still use the von Neumann architecture which separates the computing units and memory [Har14]. Therefore, automatic biochemical platforms and novel neuromorphic computing systems are necessary to make further progress in biochemical experiments and ANNs, respectively.

Flow-based microfluidic biochips are one of the most promising platforms used in biochemical and pharmaceutical laboratories due to their high efficiency and low costs. They have been applied to biochemical assays, such as DNA/RNA sequencing [Fai07, WHD$^+$13, LCH13], destruction and crystallization of protein molecules [XCP10], immunohybridization reactions [TMQ02], cell culture and analysis [YWR$^+$15]. Inside such biochips, fluids of nanoliter volumes are transported between devices for various operations, such as mixing and detection. The transportation channels and corresponding operation devices are controlled by microvalves driven by external pressure sources. Since assigning an independent pressure source to every microvalve would be impractical due to high costs and limited system dimensions, microvalves are switched on/off by by a control logic using time multiplexing. Existing control logic designs, however, still switch only a single control channel per operation, leading to low efficiency. Besides, due to the nano-scale size of control logic, manufacturing faults occur frequently.

For neuromorphic computing, the structure Computing in Memory (CIM) is introduced, using Memristors and Mach-Zehnder Interferometers (MZIs). A memristor is a kind of electric device whose conductance value can be changed by applying voltages. The memristor-based crossbar can realize rapid matrix-vector multiplication operations, which are one of the most time-consuming operations in neural networks (see Chapter 2). However, the memristor device is unstable and due to the nano-scale size process variations seriously affect the practical conductance values and further reduce the computing accuracy of memristor-based crossbars.

Simultaneously, in the optical domain, MZI-based Optical Neural Networks (ONNs) have emerged as a promising computing platform. In such ONNs, phases of light are modulated through MZIs to carry computation information, and MZIs are connected in a grid-like layout to implement multiplications and additions. These computations can be conducted at the speed of light and the results can be detected at a rate over 100 GHz [LV$^+$12], while the energy efficiency of such a chip can reach 10^5 times that of GPUs [YS$^+$17]. To use MZI-based ONNs in reality, though, there exist several technical challenges, including high power consumption, process variations, and thermal crosstalk.

1.1 Contributions of This Work

To improve the working efficiency of biochips, a novel control logic structure is presented, which is able to simultaneously switch multiple control channels. Moreover, the first fault-aware design in control logic by introducing backup control paths is proposed to maintain the correct functioning even when manufacturing defects occur. The automatic construction of the control logic is achieved using an integer linear programming (ILP) model with mixing multiplexing. Simulation results demonstrate that the proposed multi-channel switching mechanism reduces valve-switching times and lowers total logic cost, while improving fault tolerance for all control channels.

To solve the process variation impacts in memristors and maintain the inference accuracy of ANNs processed by memristor-based crossbars, a statistical training method is introduced to model process variations and noise as random variables and incorporate them into weights of neural networks during training. The cost function during training is modified to represent the probability of the correct output values, so that the resulting weights can maintain a good inference accuracy after being mapped onto memristors with process variations and noise. Global variation is compensated by scaling the target programming values according to the average of variations of memristors in a column.

In the MZI-based computing systems, we first demonstrate the problem that even with small variations in the phases of MZIs, the inference accuracy degrades significantly, thus requiring a systematic solution to enable ONNs in practical applications. The cost function during training is modified to reduce variations and thermal imbalance in MZIs. To map the phase values of MZIs obtained from training, we first calibrate process variations of MZIs and the phases of MZIs are then determined according to the individual characteristic curves. Thermal effects are modeled and compensated in advance when determining the exact power values applied onto MZIs to avoid undesirable phase drifts. Finally, online tuning is used to counter residual statistical variations and noise to improve inference accuracy further. Experimental results confirm that the proposed framework can effectively recover the inference accuracy of ONNs under process variations and thermal effects, thus enabling their adoption in accelerating neuromorphic applications.

1.2 Structure of This Dissertation

The structure of this dissertation is as follows. The background on biochips, neural networks, neuromorphic computing, and their state of the art and shortcomings are introduced in Chapter 2. A framework for biochip control logic is presented to improve the working efficiency and fault tolerance in Chapter 3. Chapter 4 describes the statistical training, testing and compensating framework to counter process variations and noise. In Chapter 5, a framework for ONNs considering process variations and thermal effects is described. The concluding Chapter 6 summarizes the results and points out directions for future work.

1.3 Summary

Biochemical experiments and existing chips with the von Neumann architecture are no longer a satisfying solution to conduct biochemical researches and neuromorphic computing, respectively, due to their low working efficiency and high energy consumption. Therefore, biochips and the novel CIM architectures are becoming increasingly interesting for researchers. However, to use these emerging systems is challenging due to immature design frameworks, manufacturing faults and variations, high power consumption, and thermal crosstalk. Therefore, a framework is proposed to automatically design a novel control logic which should improve the working efficiency and fault tolerance for biochips. A statistical training and compensation method incorporating process variation impacts on memristor-based crossbars is proposed to maintain the inference accuracy of ANNs processed through memristor-based crossbars. A modified training method is also designed for MZI-based ONNs to reduce power consumption and thermal hotspots, a special algorithm is introduced to test and reduce process variations, and a method is presented to eliminate thermal crosstalk. Finally, a tuning algorithm is proposed to reduce residual noise.

Chapter 2

Background and Shortcomings of Emerging Systems

Emerging systems are needed to improve their working efficiency for researchers working on biochemical experiments and neuromorphic computing. In this chapter, biochips, memristor-based neuromorphic systems and MZI-based neuromorphic systems are introduced. Their current research is reviewed and the technical shortcomings are presented.

2.1 Biochips for Automatic Biochemical Experiments

Biochips are an automatic and highly efficient platform for biochemical experiments. In this section, the basic structure and working principle of biochips are introduced, and the existing designs of the control logic, which is one of the core structures determining the working efficiency of biochips, are reviewed.

2.1.1 Introduction of Continuous Flow Biochips

A schematic of a biochip consists of a ***control layer*** and a ***flow layer*** which form the ***control channel*** and the ***flow channel***, respectively (see Fig. 2.1). The control channel is located on top of the flow channel, and at the intersections the channels are a

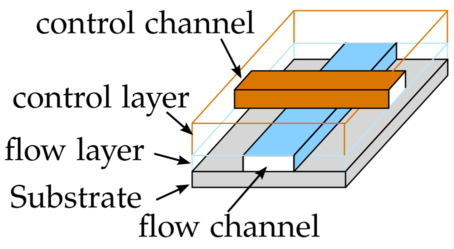


Figure 2.1: Schematic of flow-based microfluidic biochips [ZLH+18] ©2018 IEEE.

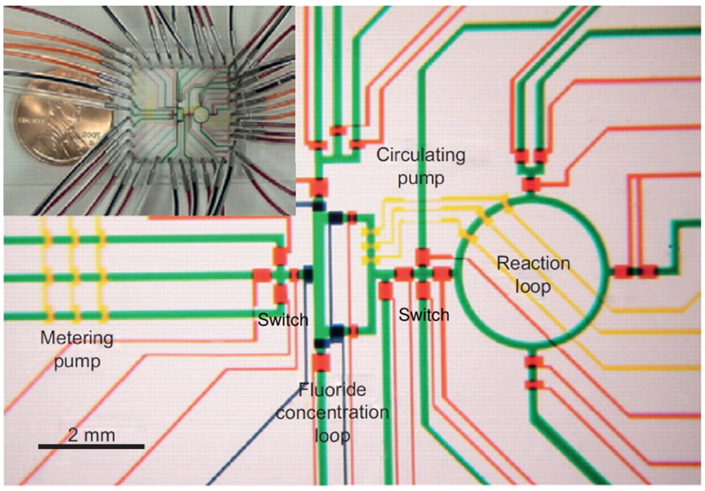


Figure 2.2: A microfluidic biochip with flow channels (green) and control channels (yellow and red) [LSE+05] ©AAAS 2005.

membrane fabricated with polydimethylsiloxane (PDMS), a kind of elastomer material [Fai07,HBM+12]. An air/fluid pressure through the control channel squeezes the flow channel to block the movement of flow samples. When the pressure in the control channel is released, the flow channel opens again for fluid transportation. In other words, the membrane is a valve, whose state is controlled by the pressure in the control channel. In the following, the valves in the flow core are called ***flow valves***, which share the same indices as the control channels. With the flow valves as the controlling units, complex biochips can be constructed, as shown in Fig. 2.2 [EiSWd13].

In such a chip, a large number of devices, e.g., mixers and detectors, are con-

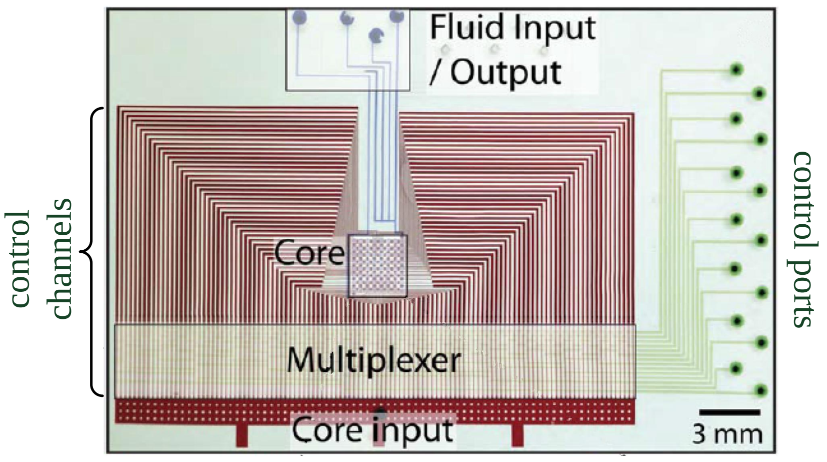


Figure 2.3: Structure of a complete biochip [FM11] ©The Royal Society of Chemistry 2011.

structed. These devices are connected by flow channels to transport intermediate experiment results. The transportation of these results is controlled by flow valves [HHG+19].

A major advantage of biochips is their large integration. Accordingly, the manufacturing process of biochips has taken a road similar to integrated circuits by etching microchannels on a substrate [AQ12]. Observing this similarity, the design automation community has started to propose methods and work flows to improve the design quality and efficiency. For example, the synthesis of biochip architectures has been addressed in [TLSH15a, TLHS15, TLL+16a, LTL+16, TLL+16b, LLY+17a, TLF+18a, TLF+18b, WZY+18, CHG+19, TLZ+19], the routing of flow channels in [LLC+14, HHCG19, GWY+17] and the storage, caching and transportation of fluid samples in [TLSH15b, LLY+17b]. Furthermore, test methods for defect detection after manufacturing have also been proposed in [HYHC14, LLB+17].

Compared with integrated circuits, biochips, however, exhibit some specific features. Besides flow channels that are used to transport fluid samples, flow valves need to be driven by external air/fluid-pressure patterns to change their states.

When executing an application, the patterns of air/fluid pressure in the control channels should be generated by a ***control logic***, which plays a critical role in a biochip, since it manages the overall execution of applications.

2.1.2 Traditional Control Logic for Continuous Flow Biochips

In Fig. 2.3 an example of a complete biochip from [FM11] is shown. The flow core of the biochip is located at the center for executing biochemical operations. The control channels surrounding the flow core, the multiplexer, the core input, as well as the pressure sources on the right-hand side together form a control logic to generate pressure patterns to switch flow valves in the flow core.

Due to the cost and the size of the mechanical components, it is not practical to assign each flow valve an independent pressure source. For example, in the design in Fig. 2.3, 114 flow valves in the flow core have been implemented. For executing applications, instead of using 114 pressure sources directly, which would be very cumbersome and expensive, only 15 pressure sources are used to generate pressure patterns, consisting of 14 control ports and one core input. The core input at the bottom provides one external pressure source that can be switched on or off. On the right, the control ports are connected to external pressure sources to create ***control patterns*** that specify which control channel can be connected to the core input. The multiplexer in the middle forms a multiplexing function to connect the channels to the core input according to these control patterns. Once a control channel is connected to the core input, its pressure value is updated to the same as that of the core input. Correspondingly, the open/closed state of the flow valve in the flow core driven by this control channel is also updated.

Fig. 2.4 explains the multiplexing function of the control logic to reduce the number of pressure sources. In this example, four complementary control ports x_1, $\overline{x}_1$, x_2, $\overline{x}_2$ are connected to pressure sources to control the connection of the control channels

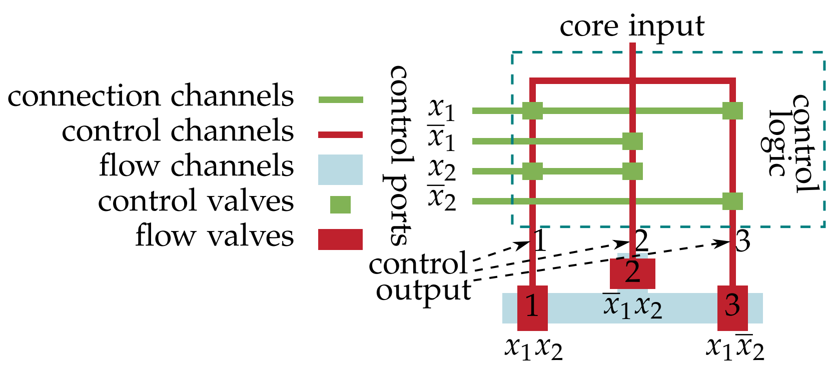


Figure 2.4: Control logic for multiplexing three control channels. Control ports x_1, $\overline{x}_1$, x_2, and $\overline{x}_2$ are connected to external pressure sources [ZLH+18] ©2018 IEEE.

that drive the three flow valves. In control logic design, the pressure values of the control ports are often complementary [FM11, MQ07]. At any time, only one of a pair of complementary ports can have a high pressure, so that the complementary control variables x_i and $\overline{x}_i$ can be implemented. Through the connection channels, these variables are used to control the valves built on top of the channels in the control logic, called ***control valves*** as shown in Fig. 2.4. The outputs of the control logic represent the states of the control channels, and are called ***control outputs***. The states of control ports and the control valves determine which control channel is to be connected to the core input to change the valves of the control outputs. For example, control channel 1 driving flow valve 1 is connected to control output 1, whose value is updated to the value of the core input when both x_1 and x_2 are set to logic '1'. In the following, the terms "control channel" and "control path" will be used interchangeably.

The combinations of control valves on the control paths form control patterns for channel multiplexing. For example, three control patterns x_1x_2, $\overline{x}_1x_2$, and $x_1\overline{x}_2$ are used in Fig. 2.4 to control the three channels. At any moment, only one of them can be true, so that only one control output can be connected to the core input for updating its pressure value. If the target pressure should be high, the pressure of

Table 2.1: Logic relationship corresponding to Fig. 2.4 [ZHL+19] ©2019 IEEE.

Control ports		Channel connected	State of the
x_1	x_2	to core input	connected channel
'1'	'1'	channel 1	=core input
'0'	'1'	channel 2	=core input
'1'	'0'	channel 3	=core input
'0'	'0'	–	–

the core input is activated; otherwise, the core input releases the pressure in the control channel.

With this mechanism, n complementary control ports can be used to multiplex $2^{n/2}$ control channels. That is to say, if the number of control channels is N, the required number of control ports is $2 * \lceil \log_2 N \rceil + 1$, where $\lceil x \rceil$ is the smallest number greater than or equal to x, and "$+1$" is the control port to core input. Note that in Fig. 2.4 three control channels exist, the required number of control ports is five, which is larger than the number of control channels. But with N increasing, the number of control channels N will also increase, the required number of control ports will increase slower than N. This reduction, however, is at the price of time consumption to switch the pressure states of control channels.

The function of the control logic is to change the pressure values, defined as ***channel states***, in the control channels so that flow valves can be switched to execute applications. As shown in Fig. 2.4, assume that at time t the channel states are "011", where '1' represents that the pressure in the corresponding control channel is high and '0' represents the pressure is low. At time $t+1$, assume that the states of the control channels need to be updated to "100". Since the control logic in Fig. 2.4 only allows one control channel to be connected to the core input at a moment, the state transitions need to be implemented using three switching operations, in which the control variables x_1 and x_2 are set to "11", "01" and "10", respectively. In this process, the three control channels are connected to the core input one after the other, activated by the control patterns x_1x_2, $\overline{x}_1x_2$, and $x_1\overline{x}_2$, respectively. Accordingly, the pressure of the core input should sequentially be set to '1', '0' and '0' to update the pressures in the control channels. For convenience, the time to update all control channels from their states at time t to their states at time t+1 is called a ***time slot***. Within a time slot, the states of several control channels may need to be changed by the control logic. Therefore, the state transition from time slot t to time slot $t+1$ may be divided into several ***time slices***, each of which represents an actuation of the control logic, costing one real time unit.

In the scenario where each flow valve is controlled by an independent pressure source, only one time slice is required to switch the pressure values for all control channels simultaneously. However, the control logic with the multiplexing structure usually costs more than one, e.g. three time slices in Fig. 2.4 from "011" to "100", leading to a tradeoff between design complexity and efficiency.

Recently, related research considering control channel optimization has started to appear. For example, the method in [HDHC17] minimizes pressure-propagation delay in control channels to reduce the response time of valves. The lengths of control channels are matched in [YHC15] to synchronize switching times of valves. These methods, however, mainly focus on the control channels that deliver air pressure to valves. The control logic to generate the required pressure patterns has not sufficiently been investigated yet. Up to now, only one method has been proposed to consider the reliability of control logic [WZY$^+$17], where the order of patterns that are required to control valves is adjusted to reduce the maximum number of switching times (or time slice) in the control logic. This method, unfortunately, still does not address the efficiency of generating the required pressure patterns.

2.2 Neuromorphic Hardware to Accelerate Neural Network Computing

Neuromorphic hardware is designed to accelerate neural networks. In this section, the background of neural networks and existing hardware with von Neumann architecture for neural network computing are presented. Due to the limit of memory bandwidth in the von Neumann architecture, to further accelerate neural network computing, CIM systems and novel memory units are introduced, where data is stored and processed simultaneously. In this dissertation, two CIM architectures, memristor-based crossbars in the electronic domain and MZI-based array in the optical domain, are optimized. The working principles of these architectures and technical challenges are described in this section.

2.2.1 Neural Network and Hardware Realization

The human nervous system is formed of neurons connected by myelin sheath through which electronic signals carrying information are transported and processed [KSJ+00]. Inspired by this, the basic structure of neural networks is designed as shown in Fig. 2.5, consisting of an input layer, a hidden layer and an output layer. The hidden layer can be extended into multiple layers in large neural networks.

The nodes in a neural network represent neurons labeled by $x_i^{[k]}$ or $a_i^{[k]}$, where k and i are the indices for the layer and the neuron in a layer or sublayer, respectively. In this dissertation, numbering of k and i starts from 0 to be consistent with the numbering in most programming languages. Generally, except for the input layer where the data are directly fetched into input neurons, each hidden layer and the output layer consist of two sublayers. The sublayer consisting of neurons labeled x

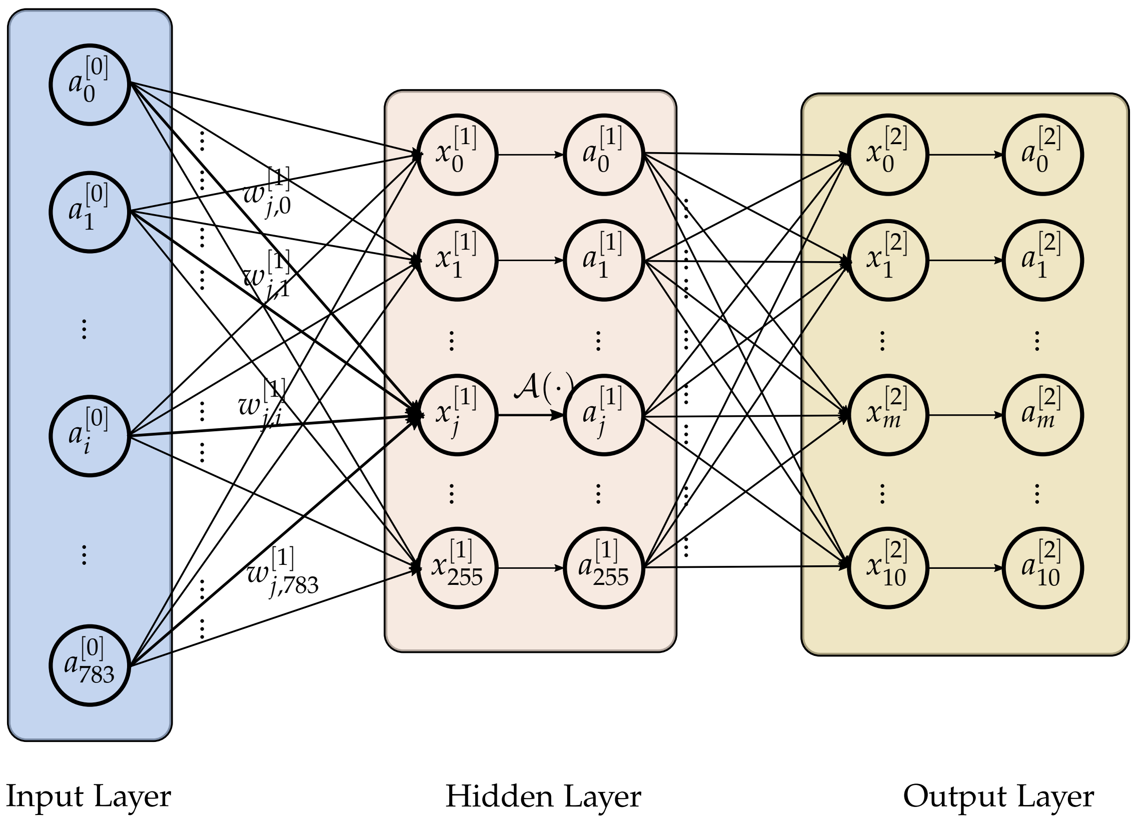


Figure 2.5: Basic structure of neural networks.

receives data from the previous layer after linear operations using

$$x_j^{[k]} = \sum_{i=0}^{N_{k-1}-1} w_{j,i}^{[k]} a_i^{[k-1]}, \tag{2.1}$$

where $w_{j,i}^{[k]}$ is the weight, representing the importance of the neuron i in layer $(k–1)$ to the neuron j in the layer k, see in Fig. 2.5 $x_j^{[1]} = \sum_{i=0}^{783} w_{j,i}^{[1]} a_i^{[0]}$; then the data is propagated through activation functions $\mathcal{A}(\cdot)$ to their target neurons a which make up the other sublayer by

$$a_j^{[k]} = \mathcal{A}(x_j^{[k]}). \tag{2.2}$$

Activation functions can include ReLU

$$a_j^{[k]} = \max(x_j^{[k]}, 0), \tag{2.3}$$

and sigmoid function

$$a_j^{[k]} = \frac{1}{1 + e^{-x_j^{[k]}}}. \tag{2.4}$$

Instead of the functions of a single fold x_j^k, they can also be the max pooling function

$$a_j^{[k]} = \max_{i \in \mathcal{I}} x_i^{[k]}, \tag{2.5}$$

where $\mathcal{I}$ is the range of neurons for comparisons, or the Softmax function

$$a_j^{[k]} = \frac{e^{x_j^{[k]}}}{\sum_{i=1}^{I} e^{x_i^{[k]}}}, \tag{2.6}$$

where I is the total number of neurons in the corresponding layer. A neural network is constructed by employing these linear and nonlinear operations.

Before putting them into applications, the weights in a neural network need to be determined by training. In the training process, training data are fetched to the input neurons, propagated through the network with multiplication, addition and activation functions and finally reach the outputs. This progress is defined as ***forward propagation***. The data at the outputs are compared with the expected values which are either 0 or 1 for classification. The difference is used to construct a cost function, which should be minimized to obtain weights that enable the neural network to achieve a satisfactory inference accuracy. A common cost function is Mean Square Error (MSE) as

$$\mathcal{C} = \frac{1}{M} \sum_{i=0}^{M-1} (\hat{a}_i - a_i^{[L-1]})^2 \tag{2.7}$$

where M is the number of neurons in the output layer, $L - 1$ is the index of the output layer, $a_i^{[L-1]}$ is the ith output of the neural network, and $\hat{a}_i$ is the expected value of the corresponding output.

Another common used cost functions is Cross Entropy expressed as

$$\mathcal{C} = -\frac{1}{M} \sum_{i=0}^{M-1} (\hat{a}_i \log(a_i^{[L-1]}) + (1 - \hat{a}_i) \log(1 - a_i^{[L-1]})) \tag{2.8}$$

with sigmoid as the activation function which results in values between 0 and 1 for $a_i^{[L-1]}$. In deep learning, the most frequently used activation function is Softmax, which can interpret outputs as a probability distribution, because it results in values in the interval $[0,1)$ for them and the sum of them equals 1. Accordingly, the cost function is Log-Likelihood Cost written as

$$\mathcal{C} = -\frac{1}{M}\sum_{i=1}^{M}(\hat{a}_i \log(a_i^{[L-1]})). \tag{2.9}$$

To minimize the cost function $\mathcal{C}$, the weights in the neural network, denoted as w_i, are adjusted according to the gradient of $\mathcal{C}$ with respect to the weights $\frac{\partial \mathcal{C}}{\partial w_i}$, namely, the ***gradient descent*** method [BV04], as

$$w_i \leftarrow w_i - \gamma \cdot \frac{\partial \mathcal{C}}{\partial w_i} \tag{2.10}$$

where γ is the learning rate and $\frac{\partial \mathcal{C}}{\partial w_i}$ can be calculated by the ***back-propagation*** algorithm [LBH15].

The back-propagation method calculates the gradients of $\mathcal{C}$ with respect to the parameters from the latest outputs layer, hidden layers to the input layers according to the chain rule, i.e., $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$, if $y = f(u)$ and $u = g(x)$ are differentiable functions [Ste99]. The gradient of $\mathcal{C}$ with respect to $a_m^{[k]}$, $\frac{\partial \mathcal{C}}{\partial a_m^{[k]}}$, where $k = L$–1, is calculated firstly with the present value of $a_m^{[k]}$ according to (2.7-2.9). Then, $\frac{\partial \mathcal{C}}{\partial x_j^{[k]}}$, the gradients of $\mathcal{C}$ with respect to $x_j^{[k]}$ can be calculated with the present value of $\frac{\partial \mathcal{C}}{\partial a_j^{[k]}}$ and $x_j^{[k]}$ by

$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial x_j^{[k]}} &= \frac{\partial \mathcal{C}}{\partial a_j^{[k]}} \cdot \frac{\partial a_j^{[k]}}{\partial x_j^{[k]}} \\ &= \frac{\partial \mathcal{C}}{\partial a_j^{[k]}} \cdot \frac{\partial \mathcal{A}(x_j^{[k]})}{\partial x_j^{[k]}}. \end{aligned} \tag{2.11}$$

After that, with the present values of $\frac{\partial \mathcal{C}}{\partial x_j^{[k]}}$, $a_i^{[k-1]}$ and $x_j^{[k]}$, we can obtain $\frac{\partial \mathcal{C}}{\partial w_{j,i}^{[k]}}$ and

$\frac{\partial \mathcal{C}}{\partial a_i^{[k-1]}}$ using

$$\begin{aligned}\frac{\partial \mathcal{C}}{\partial w_{j,i}^{[k]}} &= \frac{\partial \mathcal{C}}{\partial x_j^{[k]}} \cdot \frac{\partial x_j^{[k]}}{\partial w_{j,i}^{[k]}} \\ &= \frac{\partial \mathcal{C}}{\partial x_j^{[k]}} \cdot a_i^{[k-1]}\end{aligned} \tag{2.12}$$

and

$$\begin{aligned}\frac{\partial \mathcal{C}}{\partial a_i^{[k-1]}} &= \frac{\partial \mathcal{C}}{\partial x_j^{[k]}} \cdot \frac{\partial x_j^{[k]}}{\partial a_i^{[k-1]}} \\ &= \frac{\partial \mathcal{C}}{\partial x_j^{[k]}} \cdot w_{j,i}^{[k]}.\end{aligned} \tag{2.13}$$

By scanning k from L–1, L–2 to 1, we can obtain all weight gradients. With (2.10), after a sufficient number of iterations, the weights can be determined as $\mathcal{C}$ converges.

The neural network shown in Fig. 2.5 is a simple fully-connected neural network which can only realize very simple application. For example, it can infer MNIST [LCB10] with a 96% accuracy. However, to realize more complicated applications such as face identification, data mining, machine translation, neural networks are designed with tens or hundreds of hidden layers, over millions of weights and complex connections, such as convolutional neural networks (CNNs) [KSH12], residual neural networks (ResNets) [HZRS16], recurrent neural networks (RNNs) [MKB$^+$10]. Training these neural networks with the back-propagation method employing differential equations derived manually is complicated and error-prone. Therefore, to address the concerns, the community of neural networks develops several libraries, e.g., CNTK [Eta19], Caffe [JSD$^+$14], Keras [GP17], PyTorch [PGM$^+$19], TensorFlow [AAB$^+$15]. They can provide gradient calculations in back-propagation and neural network training automatically. TensorFlow is the library used in this dissertation because it can easily build data flow models, work on servers, edge devices, and the web, and support parallelization over multiple processors [AAB$^+$15].

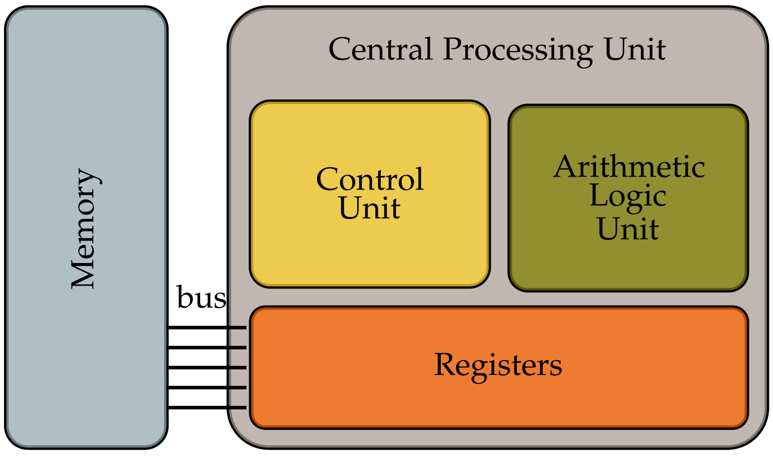


Figure 2.6: CPU Structure.

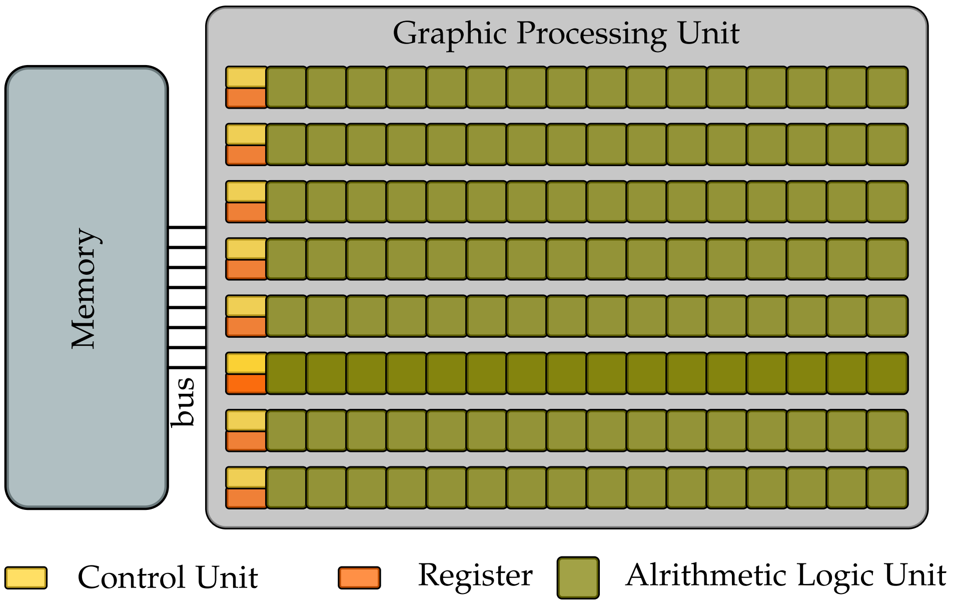


Figure 2.7: GPU Structure.

It has one of the largest open source communities as well as comprehensive documents, and is supported by the Google Brain Team [Dub17].

Although libraries can improve the working efficiency of researchers who work in high level design, computing efficiency and power consumption of training and inference for complicated neural networks are eventually decided by hardware, exactly where large amounts of data are stored and processed. As mentioned earlier,

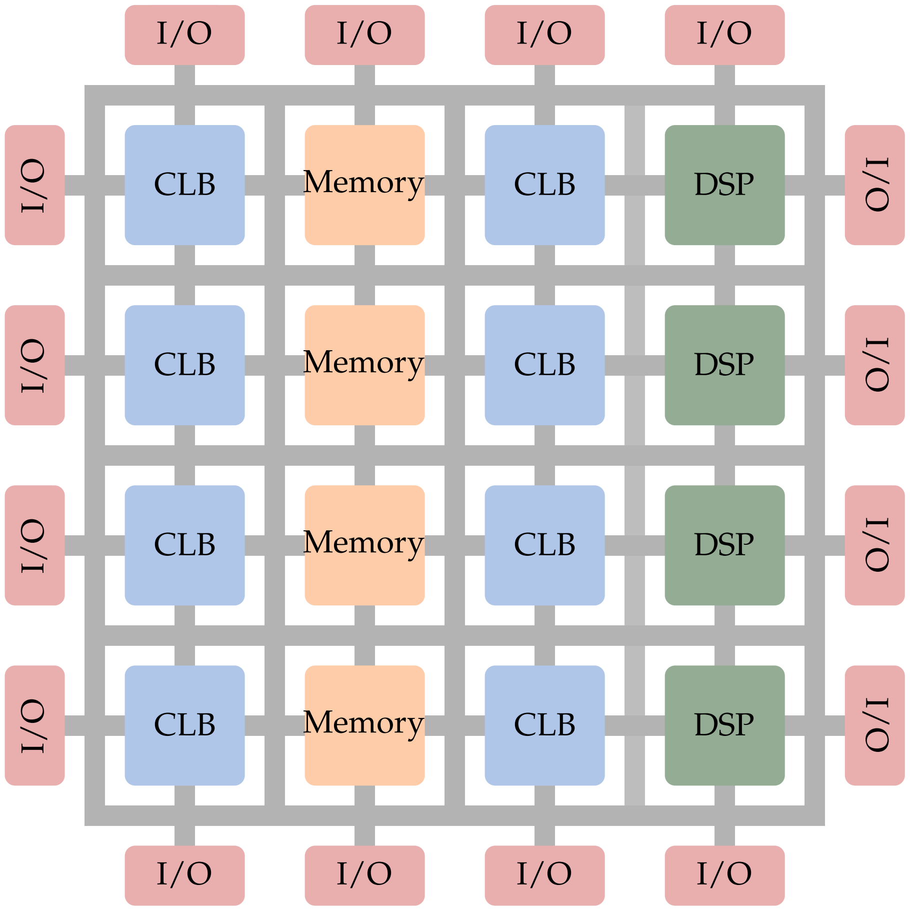


Figure 2.8: FPGA Structure.

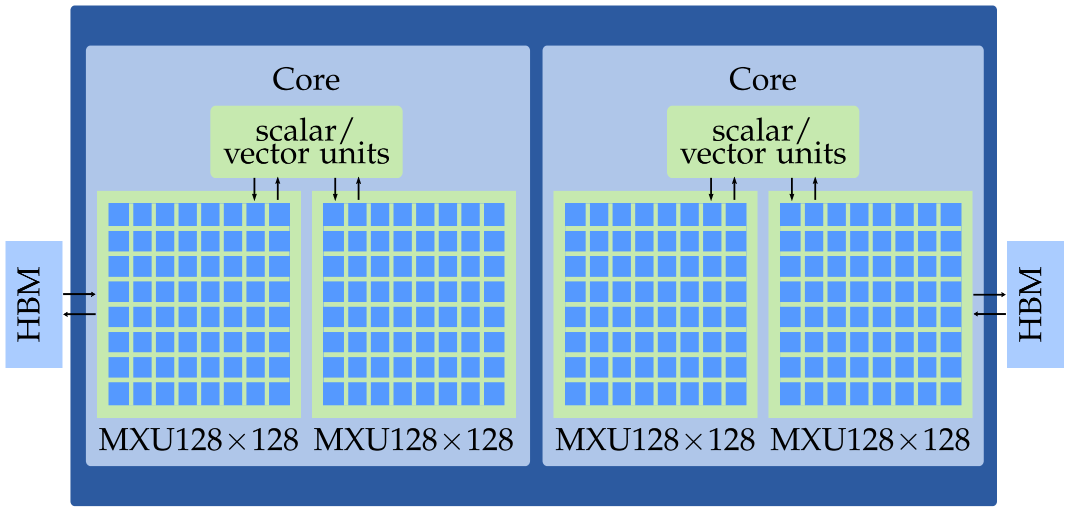


Figure 2.9: Schematic of TPU v3 - 4 chips, 2 cores per chip.

the concept of neural networks was proposed in the 1960s, but after that its development was blocked because the hardware capacity was low then until the increasing maturity of integrated circuit technologies enabled computers to operate with large amounts of computing at an acceptable cost. However, the "acceptable" cost does not mean high speed and low cost. Actually, to compute complicated neural networks in traditional computers requires too much time and power. Therefore, researchers have been devoting much time and effort to designing and optimizing hardware for AI.

Since the early days of computing, people have used CPUs (see Fig. 2.6 [Bro08]), which can realize complex operations and whose performance has been improved by technologies like multi-core processors and pipelines [BDRDR03]. However, the performance is eventually limited by memory bandwidth caused by the von Neumann architecture where data computing and storage are separated and too much data must be transported between arithmetic logic units (ALUs) and memory [McK04, XY19]. Because the main operations of neural networks are multiplication and addition [GZY^{+}19], GPUs are adept at them with high throughputs and large quantity of ALUs working in parallel as shown in Fig. 2.7 [Bro08]. GPUs have become the most widely used hardware in both training and inference for neural networks. However, GPUs consume too much power, which is a challenge for widely using both large cloud servers and customer devices. Besides, GPUs are also overqualified for applications that do not require high data throughputs, like video stream and image classification in edge computing [SSEM18]. FPGAs, see Fig. 2.8 [KTR^{+}08], have been proposed as a supplement for CPUs and GPUs in neural network computing. With hundreds to thousands of digital signal processing (DSP) units, configurable logic blocks (CLBs), many programmable input/output (I/O) buses and large internal memory, FPGAs with hardware-oriented neural network structure compression and hardware architecture design based on neural network structure can achieve over 10$\times$ greater speed and energy efficiency than the latest GPUs [LTA16, GZY+19]. To implement deep learning networks on FPGAs, challenges still exist with regard to data storage, memory bandwidth, and computational resources. Application-specific integrated circuit (ASIC) accelerators have also been utilized for neural network computing, e.g. custom-designed tensor processing units (TPUs) (see Fig. 2.9) by Google [Jou16]. But they have disadvantages with their development cycle, costs and flexibility.

Therefore, to reduce the access off-line memory and improve the integration density, engineers propose the CIM, a.k.a. computing in memory. These memory systems include but not limited in the following items:

Static Random Access Memory (SRAM)
SRAM is a memory system based on a 6-transistor (6-T) cell (see Fig. 2.10 (a)). By applying high voltage to WL, 1-bit data can be quickly written into or read from the SRAM cell through BL or BLB whose value is always opposite [ZWV17]. The SRAM cells are connected as an array (see Fig. 2.10 (b)) to realize multiplications for neural networks. In the array, data in the neural networks can be continuous values and applied as the voltage value output from digital-to-analog converters (DACs) into the wordlines; the weights are binary values and represented by the SRAM cell; the output currents in BL or BLB are the products of data and weights [ZWV17]. This array requires the neural network weights to be specially trained due to the binary limits and the structure to be custom designed or enlarged to maintain inference accuracy. To realize multiplication between arbitrary data and weights, [KGPS18] presented a method using multi SRAM cells to store weights and data and sequentially share charges after several time slots to obtain a multiplication result, both of which, however, are time-inefficient. In addition, SRAM cells are volatile, requiring power to maintain their values.

Flash Memory [MBGK+17]
Flash memory is based on the metal-oxide-semiconductor field-effect transistor (MOSFET). Its stored memory value is presented by the floating-gate charge, which

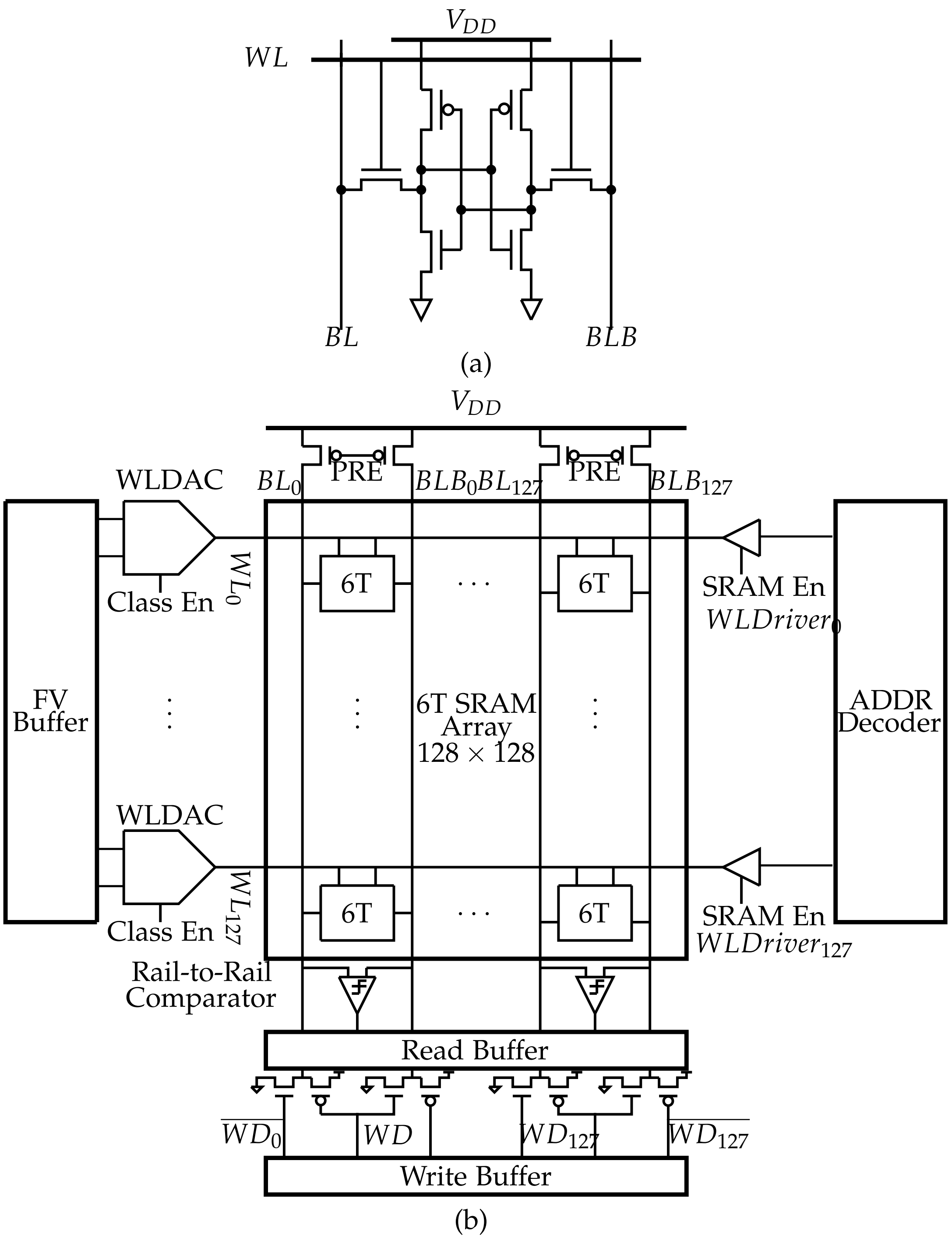


Figure 2.10: Schematic of SRAM-based CIM architecture. (a) 6-T SRAM. (b) Architecture of 6-T SRAM array [ZWV17] ©2017 IEEE.

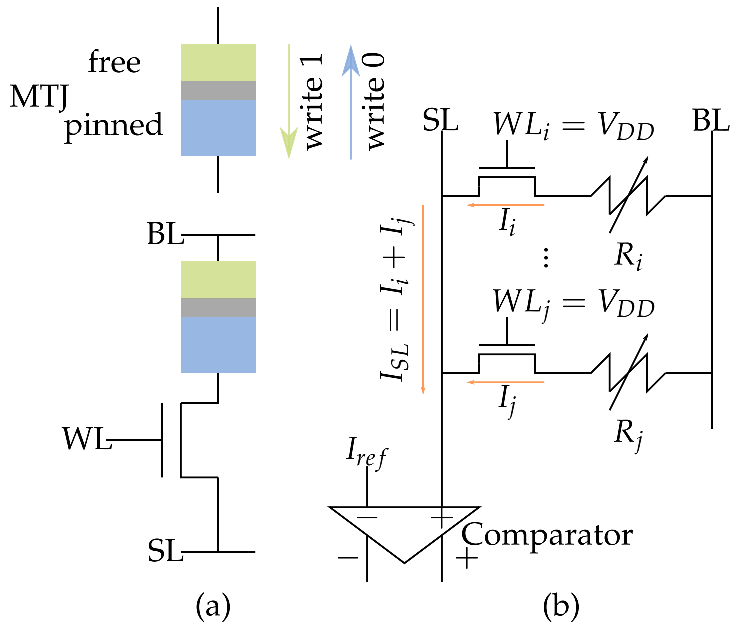
can be reduced by adding electrons to the floating gate, i.e., channel hot-electron injection (CHEI), or increased by removing electrons using Fowler-Nordheim (FN) tunneling. By reducing or increasing the floating-gate charge, MOSFET threshold voltages are increased or decreased, accordingly [DHMM96]. [MBGK$^+$17] presents an architecture to realize three-layer neural networks for MNIST, in which the multiplication is realized using the relation between MOSFET source-to-drain currents as the output and MOSFET gate voltages and threshold voltages as the input when MOSFETs are working in the linear region and subthreshold mode. However, this architecture is complicated requiring peripheral floating transistors and subtraction amplifiers for robustness and reliability.

Magnetic Random Access Memory (MRAM)

An MRAM bit-cell consists of an access transistor and a magnetic tunnel junction (MTJ), see Fig. 2.11 (a). By applying voltages with different directions to MTJ, its states can be switched to a low resistance R_P representing logic '1' or a high resistance R_{AP} representing logic '0'. MRAM can be rapidly read from or written into and is non-volatile, i.e., no power is needed to maintain data. With the structure in Fig. 2.11 (b) and introducing different reference currents, logic operations AND, OR, $NAND$, NOR and XOR can be realized [JRRR17]. Based on these operations, single-bit in-memory addition and multiplication can be achieved. However, multi-bit addition and multiplication are still time-inefficient.

Electronic Phase Change Memory (EPCM)

The core element of PCM is the phase change material, which can be continuously changed between the amorphous phase and the crystalline phase using Joule heating. Different phases result in different conductance or resistance values of the EPCM [SLGB$^+$18]. When the materials are sandwiched between two electrodes and a crossbar is constructed (see Fig. 2.12), the output current from each column is the summation of the products between the input voltages and the conductances of the EPCM in the corresponding column, which can quickly realize matrix-vector



(R_i, R_j)	I_{SL}
(R_p, R_p)	I_{P-P}
(R_p, R_{AP})	I_{P-AP}
(R_{AP}, R_p)	I_{AP-P}
(R_{AP}, R_{AP})	I_{AP-AP}

(c)

Figure 2.11: Basics for MRAM-based Computing Unit [JRRR17]. (a) Applying voltages to MRAM in different directions can write ′0′ or ′1′ in MRAM. (b) CIM in MRAM can be realized by comparing the total output current of corresponding MRAMs and a reference current. (c) Output currents of two MRAMs in different states ©2017 IEEE.

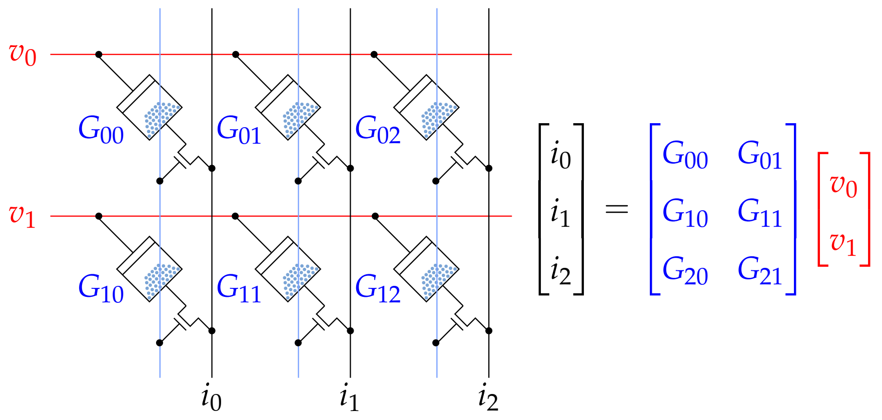

v_0
v_1
G_{00}
G_{01}
G_{02}
G_{10}
G_{11}
G_{12}
i_0
i_1
i_2
$\begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} G_{00} & G_{01} \\ G_{10} & G_{11} \\ G_{20} & G_{21} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$

Figure 2.12: Computing structure based on EPCM in the electrical domain. Reproduced from [SLGB+18] with the permission of AIP Publishing.

multiplications. However, this material is easily impacted by $1/f$ noise, aging, as well as nonlinearity and randomness in crystallization, which further reduce the computing accuracy of EPCM-based crossbars [SLGB+18].

Optical Phase Change Memory (OPCM)

The optical domain contains OPCM. Through heating, the material can switch between the crystalline and amorphous phase continuously too, which leads to different optical transmittances. The material is magnetron sputtered on the optical waveguide to construct a computing structure as in Fig. 2.13 [FYW+19]. The input data are carried by different wavelength lights. The data values are represented by the values of optical power. These optical signals are collected firstly by the collector in Fig. 2.13, and then the power of every optical signal is equally distributed to the corresponding optical rings. This process is controlled by the coupling efficiency and ring radii. After that, the optical signals are transported through the phase change material, and the output optical signal power is the product of the corresponding input signal and the transmittance of OPCM. Finally, the optical signal power is collected again to sum the products. Therefore, the matrix-vector multiplication is realized.

Because optical signals with the same wavelength interfere with each other resulting in amplitude superimposition, the optical ring cannot distinguish between signals on the same wavelength or on two wavelengths whose distance is the free spectral range (FSR) [BDHVV$^+$12], which makes the collection and distribution unavailable. Accordingly, every input data requires an independent wavelength optical signal in a limited range. Note the available wavelength range is determined by both the FSR and the optical waveguide functional range. Furthermore, the wavelength of the optical source carrying input data is not an ideal value but around which the optical power exists on the wavelengths. Therefore, signal information is distributed on carrying lights with wavelengths around the ideal value and crosstalks exist between two optical signals and increase as the wavelengths of the optical signals become closer to each other. Therefore, the number of available wavelengths is limited, and the structure cannot serve large neural networks.

Due to the structure size, power consumption, computing speed as well as scalability, memristor-based crossbars (see Section 2.2.2) and optical MZI arrays (see Section 2.2.3) to realize CIM for neural networks are focused in this dissertation.

2.2.2 Emerging System with Memristor-based Crossbar

In this subsection, the memristor is introduced, how the memristor-based crossbar executes the matrix-vector multiplication is described, what degrades the computing accuracy of the crossbar is explained, and current research that reduces the degradation is presented.

In 1971, Leon Chua noted that there are three basic two-terminal circuit elements resistor R, capacitor C and inductor L, which are defined in terms of the relationships between two of the four fundamental circuit variables, namely, the current i, the voltage v, the charge q and the flux φ. For these four variables, the number of relationships between two of them is six. Five of them are known with three definition relations $R = \mathrm{d}u/\mathrm{d}i$, $C = \mathrm{d}q/\mathrm{d}v$ and $L = \mathrm{d}\varphi/\mathrm{d}i$ and two time-related

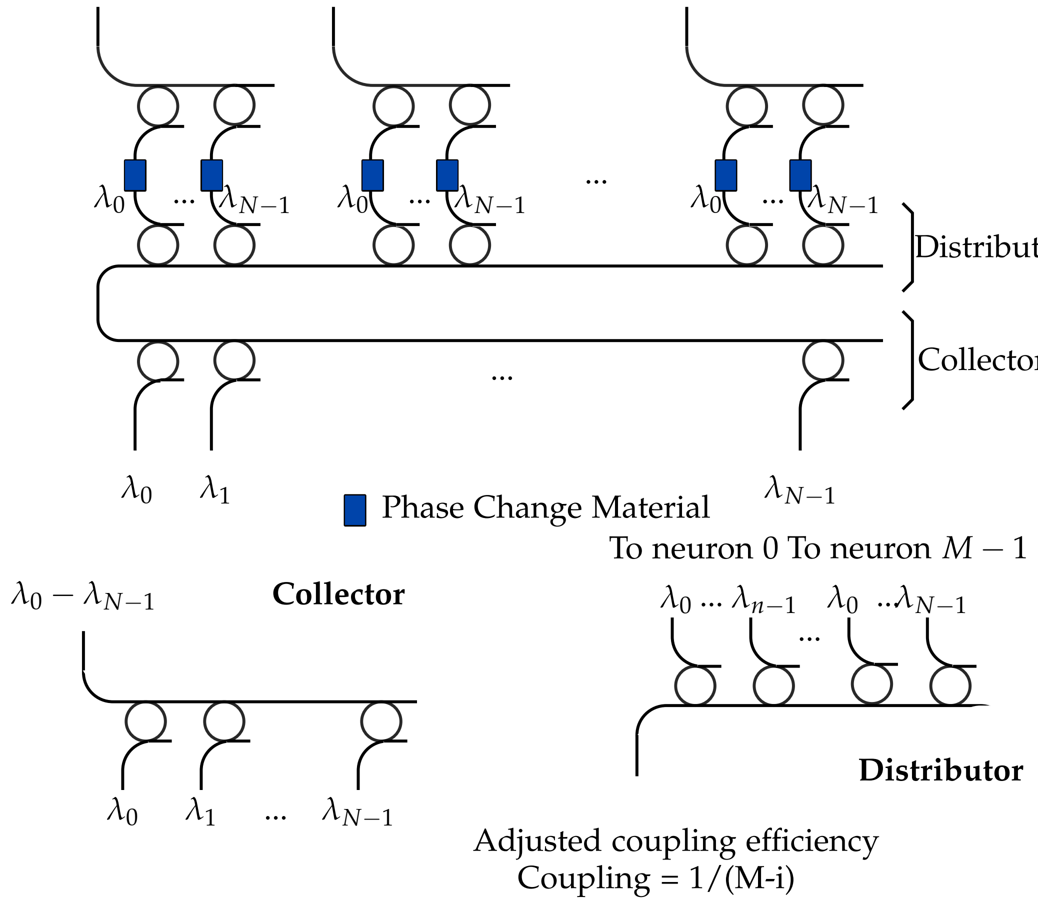

λ_0 ... λ_{N-1}
λ_0 ... λ_{N-1}
...
λ_0 ... λ_{N-1}
Distributor
Collector
...
λ_0 λ_1
🟦 Phase Change Material
λ_{N-1}
To neuron 0 To neuron $M-1$
$\lambda_0 - \lambda_{N-1}$
Collector
λ_0 ... λ_{n-1} λ_0 ..λ_{N-1}
...
λ_0 λ_1 ... λ_{N-1}
Distributor
Adjusted coupling efficiency
Coupling = 1/(M-i)

Figure 2.13: Computing structure based on OPCM in the optical domain [FYW+19] ©2019, Springer Nature.

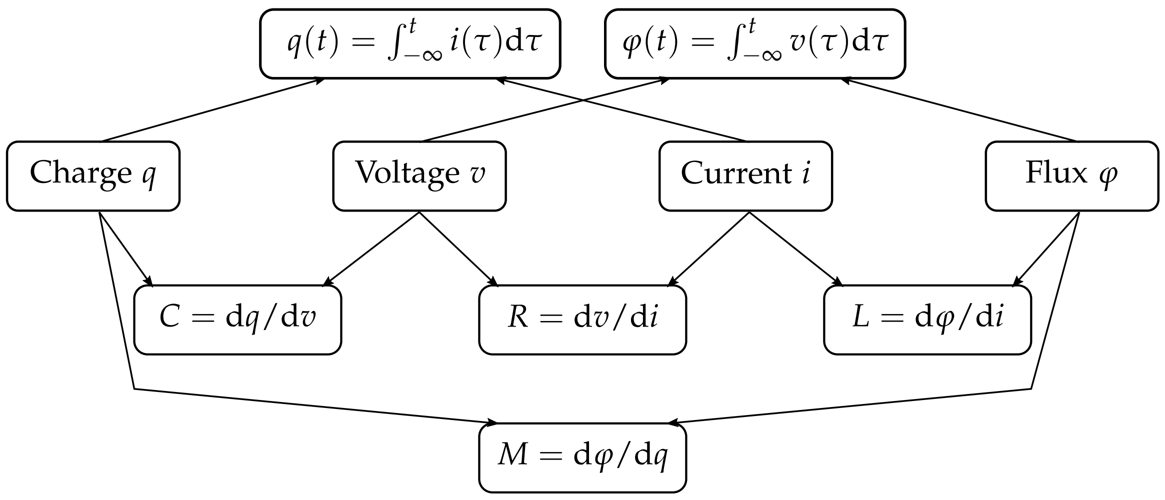

$q(t) = \int_{-\infty}^{t} i(\tau)\mathrm{d}\tau$
$\varphi(t) = \int_{-\infty}^{t} v(\tau)\mathrm{d}\tau$
Charge q
Voltage v
Current i
Flux φ
$C = \mathrm{d}q/\mathrm{d}v$
$R = \mathrm{d}v/\mathrm{d}i$
$L = \mathrm{d}\varphi/\mathrm{d}i$
$M = \mathrm{d}\varphi/\mathrm{d}q$

Figure 2.14: Relationships in the four fundamental two-terminal circuit elements, i.e., resistor, capacitor, inductor and memristor.

relations $q(t) = \int_{-\infty}^{t} i(\tau)\mathrm{d}\tau$ and $\varphi(t) = \int_{-\infty}^{t} v(\tau)\mathrm{d}\tau$ (see Fig. 2.14). One is undefined, which is the relationship between φ and q. In [Chu71], Chua inferred it as

$$M = \mathrm{d}\varphi/\mathrm{d}q = v/i, \tag{2.14}$$

where M is the fourth basic element ***memristor***. The memristor resistance can be interpreted by the slope of $\varphi - q$ curve. If the $\varphi - q$ curve is nonlinear, memristor resistance can vary along the $\varphi - q$ curve (see an example in Fig. 2.15 [KSYC10]). Therefore, memristor resistance can be programmed or tuned to a specific value by changing the flux φ or the charge q, namely, applying voltage v or current i to memristor terminals for a period of time. This process is defined as ***programming*** [SKK10, KSYC10, HSL$^+$16].

The real memristor devices are fabricated with a thin film of titanium dioxide in 2008 by HP Labs [SSSW08] (see Fig. 2.16). In the memristor, part of the titanium dioxide is doped so that its resistance is low and the undoped part is insulated with high resistance. When a bias V is applied, the oxygen vacancies in the doped part will be driven into the undoped part, lowering the memristor resistance. In con-

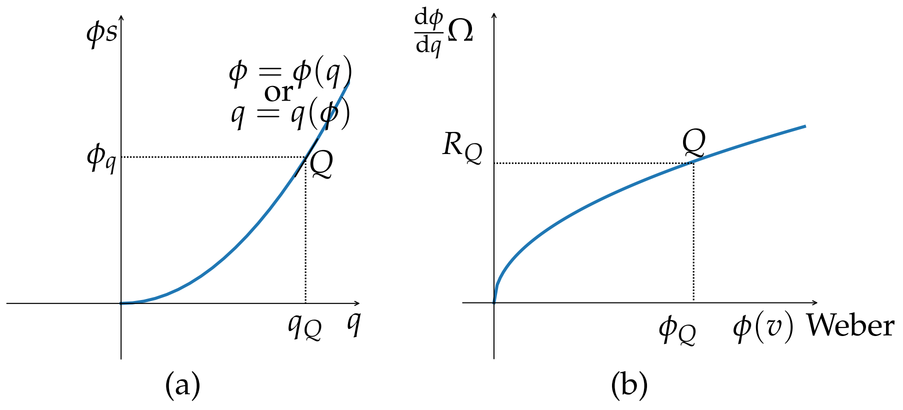


Figure 2.15: An example of $\phi - q$ curve for memristors. (a) The nonlinear curve between the flux ϕ and the charge q. (b) The slop of the curve in (a) [KSYC10] ©2010 IEEE.

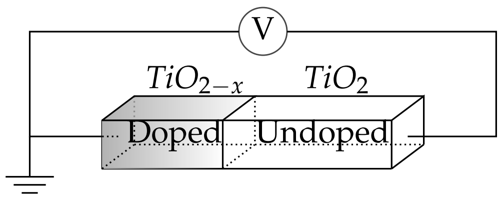


Figure 2.16: Diagram for memristor [SSSW08] ©2008, Springer Nature.

trast, a reversed bias V will drive the oxygen vacancies from the undoped part to the doped part, increasing the memristor resistance [NCXX10,Chu11,SW13]. This feature enables memristors to function as nanoelectronic memories and computer logic and further realize neuromorphic computing [ZSL18,Mit19].

The memristor-based crossbar is a crossbar where memristors sit between the horizontal wordlines and the vertical bitlines at the crossing points (see Fig. 2.17 [LYY+15]). Assuming a voltage vector, $\mathbf{U} = (U_1, U_2, ..., U_i, ..., U_M)$ is applied to the memristors on the jth column through the horizontal wordlines in the crossbar in Fig. 2.17. A current $I_j = \sum_{i=1}^{M} U_i g_{ij}$ can then be detected at the bottom of column i, where g_{ij} is the conductance of the memristor in the ith row and the jth column. Accordingly, the outputs of all columns produce the results of a matrix-vector mul-

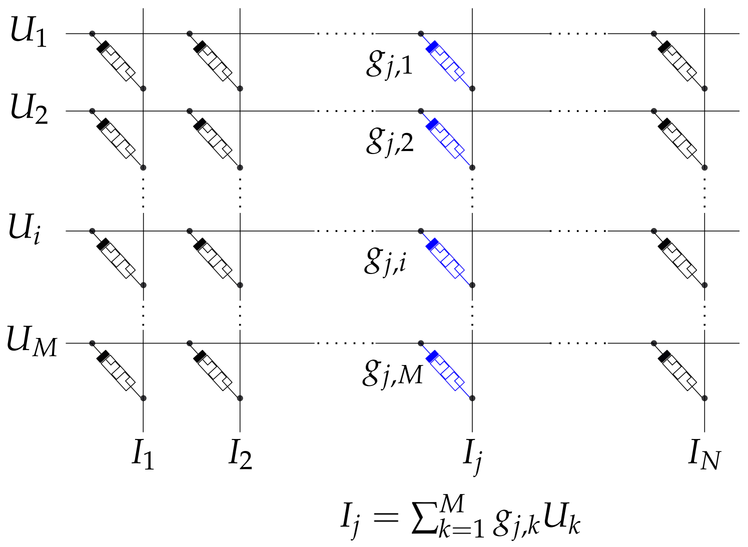


Figure 2.17: Memristor crossbar architecture [ZZW+20] ©2020 IEEE.

tiplication,

$$\begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_j \\ \vdots \\ I_N \end{bmatrix} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,i} & \cdots & g_{1,M} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,i} & \cdots & g_{2,M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ g_{j,1} & g_{1j2} & \cdots & g_{j,i} & \cdots & g_{j,M} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ g_{N,1} & g_{N,2} & \cdots & g_{N,i} & \cdots & g_{N,M} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_i \\ \vdots \\ U_M \end{bmatrix}. \tag{2.15}$$

In the matrix-vector multiplication executed by a crossbar, the matrix is represented by the conductance values of memristors. In neuromorphic computing, these conductance values correspond to the weights of a neural network after training, and should be programmed into the memristors before the crossbar is used to accelerate computing. Assuming that the ith weight w_i in the neural network is mapped to the conductance g_i of the ith memristor, this mapping can be described as

$$g_i = \frac{g_{max} - g_{min}}{w_{max} - w_{min}}(w_i - w_{min}) + g_{min} = \alpha w_i + \beta, \tag{2.16}$$

$$\alpha = \frac{g_{max} - g_{min}}{w_{max} - w_{min}}, \tag{2.17}$$

$$\beta = g_{min} - \frac{g_{max} - g_{min}}{w_{max} - w_{min}} w_{min} \tag{2.18}$$

where g_{max} and g_{min} are the maximum and minimum conductance values in the crossbars, respectively; w_{max} and w_{min} are the maximum and minimum weights, respectively [ZZL+19].

Since matrix-vector operations are realized by the electrical nature of the devices in a crossbar, the computation efficiency of such a platform can potentially significantly outperform the conventional von Neumann architectures. Accordingly, memristor-based crossbars have been implemented in hardware to accelerate neural networks [HGL+18], and methods to integrate them into existing computing systems have been explored [LWW+14, SDCG+15].

However, in reality, programming memristors is challenging, because process variations make memristors differ from each other and noise affects the programming accuracy as well [CYL14]. After manufacturing, the variations cause the physical and electrical properties of memristors to differ from each other [NCXX10]. Therefore, the same programming voltage applied to different memristors changes their conductance values differently. Consequently, the conductance values of memristors deviate from their nominal values after training. Since process variations are statistical, these conductance deviations are also statistical.

A straightforward way to overcome the impacts from process variations is to program memristors individually with many reading-programming cycles. But this method is too time-consuming for crossbars in large-volume industrial production. To counter process variations and noise, [COT00] trains the weights using the Monte Carlo simulation to minimize the expected value of the cost function. [LLC+15] minimizes the corner cases of the statistical cost function and adjusts the mapping between the weights and conductance values of the memristors. In addition, [CLC+17] applies iterative training and remapping to reduce the weight variance for a given crossbar. These methods, however, are either time-consuming or require heavy on-chip tests and redundant memristors to counter large variations. More recently, the concept of statistical neural networks has been proposed in [WXXS19, WXX+19]: while the canonical forms there are used to model the correlation in the inputs, the weights remain constant.

2.2.3 Emerging System with Optical Mach-Zehnder Interferometer Array

In this subsection, basics of optical signal transformation and the working mechanism of MZIs to perform multiply-accumulate operations are introduced first. Afterwards, the grid-like ONN architecture for implementing neural networks is described and design challenges of such ONNs are explained. The existing literature on ONNs is also reviewed in the end.

In ONNs, data is carried by the optical signal which is expressed by the trigonometric function

$$L = A\cos(\omega t + kz + \theta), \tag{2.19}$$

where A is the amplitude of the optical signal, ω is the frequency, t is the time, k is the wavenumber which is equal to $2\pi/\lambda$, z is the distance between the observation point and the initial point, and θ is the initial phase. $P = \omega t + kz + \theta$ is the total phase of the optical signal. When the amplitude A and phase P are known, the optical signal state can be obtained. Different data can be represented with different As and then propagated through the optical components which are fabricated in waveguides [YS+17].

MZIs are the fundamental optical components for computation. The structure of MZI is illustrated in Fig. 2.18(a). An MZI consists of two direction couplers, also

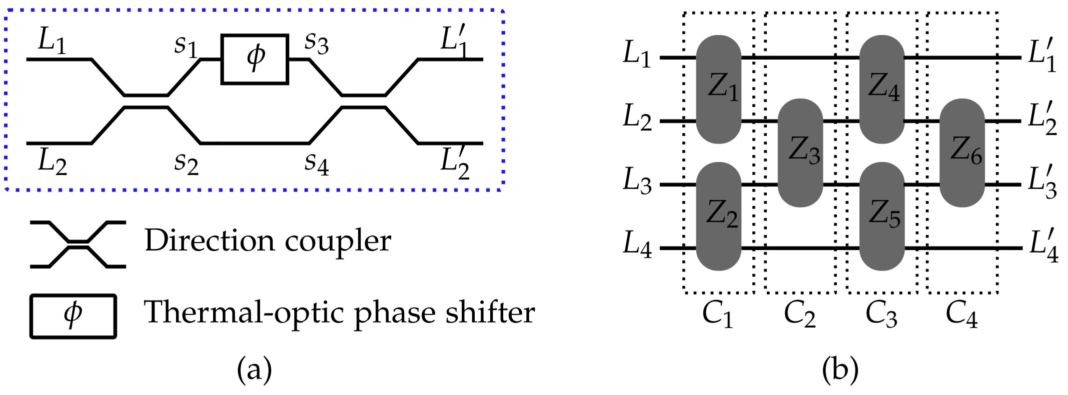


Figure 2.18: MZI and the architecture of ONN [ZZL+20] ©2020 IEEE. (a) The structure of MZI. (b) The grid-like architecture of MZI array to perform 4×4 multiplication. This architecture is composed of four columns of MZIs (C_1–C_4). Each component Z_i, $i = 1, ..., 6$, in this array represent an entire MZI shown in (a).

called beam splitters, and one thermal-optical phase shifter, whose phase can be configured by tuning its temperature with an applied power. One MZI can implement the multiplication of a 2×2 matrix and a vector of two elements using a mechanism based on interference of light, requiring light signals of the same wavelength.

Assume two input light signals L_1 and L_2 with the same frequency, namely, the same ω and k, are connected to the inputs of the MZI in Fig. 2.18(a). Their trigonometric expressions are

$$L_1 = A_1 \cos(\omega t + kz + \theta_1), \tag{2.20}$$

$$L_2 = A_2 \cos(\omega t + kz + \theta_2), \tag{2.21}$$

where A_1 and A_2 are their amplitudes, θ_1 and θ_2 are their initial phases, respectively. To simplify the following expressions, the part ωt is omitted since the frequencies of light signals are the same, and the part kz is omitted under the assumption that the observation points of light signals are at the same distance from their beginning points. For mathematical operation simplification, in the optical domain, the complex expression is more commonly used than the trigonometric expression (see Appendix A).

The light signals at the direction coupler inputs can be revised as

$$L_1^c = A_1 e^{j\theta_1}, \tag{2.22}$$

$$L_2^c = A_2 e^{j\theta_2}, \tag{2.23}$$

where j is the fundamental imaginary number, i.e., $j = \sqrt{-1}$. In the direction coupler, the light signals L_1^c and L_2^c are propagated through their waveguide branches. When the waveguides branches are closing, part of the light will be coupled to the other's branches and phase shifts will be introduced in the light signals. The coupler ratio and phase shifts are decided by the physical parameters of the two branches, e.g., the length and the distance of the parallel branches [SL12]. Generally, a symmetric 50:50 direction coupler is used here, which means half of the energy from one light signal at the input is transmitted to the other's branch after the coupler, i.e. the amplitudes of the signals become $\frac{1}{\sqrt{2}}$ of their original amplitudes. In addition, the diagonal transmission of the light signals is appended with a phase shift $\frac{\pi}{2}$ [YS+17]. Therefore, after passing through the first direction coupler in Fig. 2.18(a), the two complex signals s_1 and s_2 can be expressed as

$$s_1 = \frac{A_1}{\sqrt{2}} e^{j\theta_1} + \frac{A_2}{\sqrt{2}} e^{j(\theta_2 + \frac{\pi}{2})} = \frac{A_1}{\sqrt{2}} e^{j\theta_1} + \frac{jA_2}{\sqrt{2}} e^{j\theta_2}, \tag{2.24}$$

$$s_2 = \frac{A_1}{\sqrt{2}} e^{j(\theta_1 + \frac{\pi}{2})} + \frac{A_2}{\sqrt{2}} e^{j\theta_2} = \frac{jA_1}{\sqrt{2}} e^{j\theta_1} + \frac{A_2}{\sqrt{2}} e^{j\theta_2}, \tag{2.25}$$

where $e^{j\frac{\pi}{2}} = j$ according to Euler's formula. Therefore, the relation between $[L_1^c, L_2^c]^T$ and $[s_1, s_2]^T$ can be written as

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & j \\ j & 1 \end{bmatrix} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix}. \tag{2.26}$$

The thermal-optical phase shifter in Fig. 2.18(a) adds a phase shift ϕ to s_1. Therefore, $[s_3, s_4]^T$ can be obtained using

$$\begin{bmatrix} s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} e^{j\phi} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}. \tag{2.27}$$

Since the second direction coupler transforms $[s_3, s_4]^T$ in a way similar to (2.26), the relation between $[L_1^c, L_2^c]^T$ and $[L_1'^c, L_2'^c]^T$ can thus be expressed as

$$\begin{aligned} \begin{bmatrix} L_1'^c \\ L_2'^c \end{bmatrix} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & j \\ j & 1 \end{bmatrix} \begin{bmatrix} e^{j\phi} & 0 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & j \\ j & 1 \end{bmatrix} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix} \\ &= je^{\frac{j\phi}{2}} \begin{bmatrix} \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \\ \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \end{bmatrix} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix} \\ &= \mathbf{T} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix} \end{aligned} \tag{2.28}$$

with

$$\mathbf{T} = je^{\frac{j\phi}{2}} \begin{bmatrix} \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \\ \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \end{bmatrix}, \tag{2.29}$$

where $L_1'^c$ and $L_2'^c$ are the complex representations of L_1' and L_2' at the outputs of the MZI in Fig. 2.18(a). Therefore, the relation between the inputs and the outputs of an MZI can be expressed by the transformation matrix $\mathbf{T}$, which is a unitary matrix so that its conjugate transpose $\mathbf{T}^*$ is its inverse [FMW+19].

The conversion of inputs to the outputs in (2.28) indicates that an MZI implements a matrix-vector multiplication operation in the complex domain. This property can thus be used to accelerate computations in neural networks taking advantage of the nature of light modulation.

To implement the multiplication for multiple inputs, MZIs can be connected to form an ONN as the triangular array [RZBB94a] or the grid-like architectures [CHM+16], which allows every input signal to be propagated to every output. In this dissertation, the grid-like architecture is used to implement ONNs because the grid-like architecture costs less area than the triangular array in fabrication to realize an $N \times N$ multiplication [CHM+16]. Fig. 2.18(b) illustrates the grid-like architecture to perform a 4×4 matrix-vector multiplication for four input light signals L_1, L_2, L_3 and L_4. The whole transformation of this architecture can be expressed as the multiplication of the transformation matrices of all columns, as

$$\mathbf{T} = \mathbf{T}_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1} \tag{2.30}$$

for the ONN in Fig. 2.18(b). The transformation matrices of the columns can be formed from the transformation matrices of MZIs. For example, $\mathbf{T}_{C_1}$ and $\mathbf{T}_{C_2}$ can be expressed as

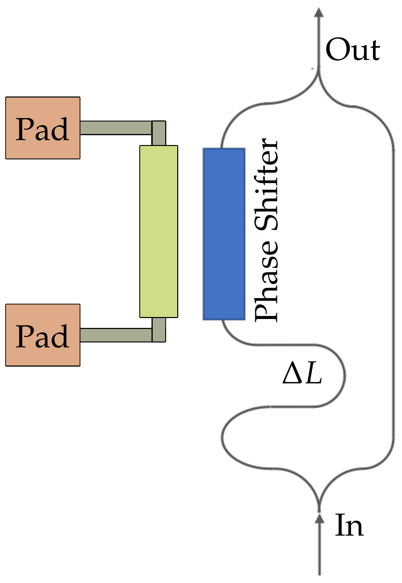
$$\mathbf{T}_{C_1} = \begin{bmatrix} \mathbf{T}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_2 \end{bmatrix}, \quad \mathbf{T}_{C_2} = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{T}_3 & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix} \tag{2.31}$$

where $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ are the 2×2 transformation matrices of the first three MZIs. All these matrices are unitary so that their conjugate transpose are equal to their inverse [CHM+16, RZBB94b].

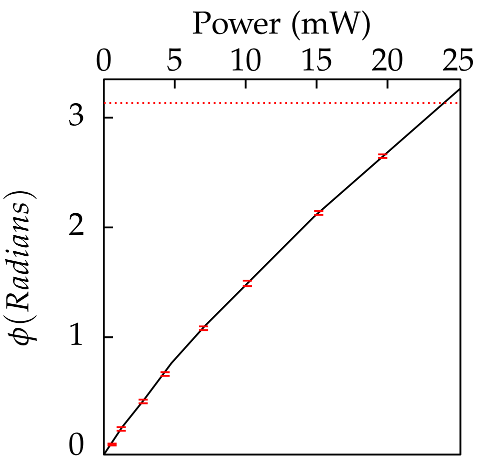
To implement ONNs, MZI phases should be programmed to expected values, which is realized by supplying different power to heat thermal-optical phase shifters. In theory [HMM+14], the relation between the phase shift $\Delta\phi$ and the supply power P is expressed as

$$\Delta\phi = \frac{2\pi L}{\lambda}\frac{dn}{dT}\frac{\tau}{H}P, \tag{2.32}$$

where L is the device length, λ is the free-space wavelength, $\frac{dn}{dT}$ is the thermo-optic coefficient, τ is the thermal time constant, H is heat capacity and P is adjusted by applying different voltages or different pulse widths per unit time to the heaters [RB17]. Fig. 2.19(a) is a layout to test the thermal-optical phase shift and Fig. 2.19(b) gives the curve of phase shift vs. power. When power is removed from the phase shifter, it will return its default state. Therefore, to maintain the phases of thermal-optical shifters requires power supplies [YS+17].



(a)



(b)

Figure 2.19: Thermo-optical phase shifter. (a) Schematic for heating the thermo-optical phase shifter. (b) Average phase shift of three devices vs. power. Adapted with the permission from [HMM^{+}14] ©The Optical Society.

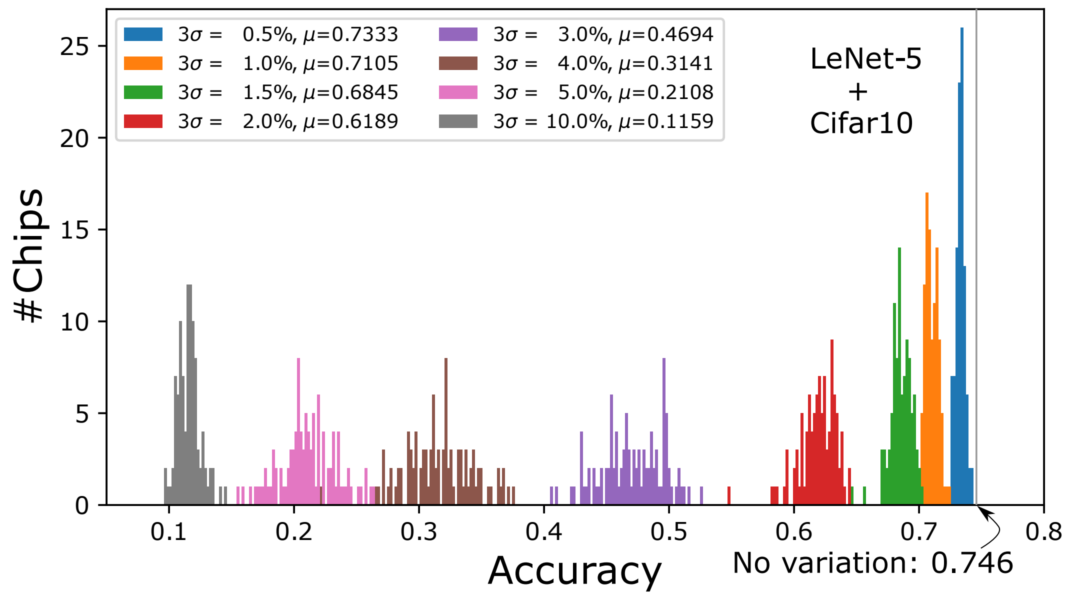


Figure 2.20: Degradation of inference accuracy of LeNet-5 on Cifar10 under process variations in MZI phases [ZZL+20] ©2020 IEEE.

In ONNs, one of the primary challenges originates from the matrix in (2.29), where the results of the trigonometric functions $\sin\frac{\phi}{2}$ and $\cos\frac{\phi}{2}$ instead of the MZI phases ϕ directly are used for matrix-vector multiplication. In mapping a neural network onto an ONN, however, only the phases of MZIs can be adjusted. Due to manufacturing process variations and thermal effects between neighboring MZIs, this adjustment inevitably suffers inaccuracy, so that the final phases of MZIs may deviate from the target values. These deviations are distorted by the $\sin(\cdot)$ and $\cos(\cdot)$ functions in the transformation matrices of MZIs such as (2.28), thus causing a significant degradation of the resulting inference accuracy. For example, Fig. 2.20 compares the inference accuracy of LeNet-5 using Cifar10 [KNH14] under different variations. Even with 0.5%–1% random variations in the MZI phases, the inference accuracy already exhibits an obvious drop. When variations become relatively large, no meaningful inference accuracy can be achieved anymore and the ONN becomes unusable. Therefore, it is critical to address process variations and thermal effects in ONNs before they can be adopted for practical applications.

To configure the phase of an MZI corresponding to the result of training, the temperature of the MZI needs to be tuned accordingly. This can be further realized by applying a given amount of power to the device. Fig. 5.3 illustrates the functions of phase changes with respect to applied power p for five exemplary MZIs under process variations extracted from [HMM+14]. According to this comparison, it can be observed that the initial phases of MZIs already differ from each other due to process variations, even if no power is applied (p=0). In addition, the same amount of power applied on MZIs also generates different phase changes. For example, in Fig. 5.3, each of p_1, p_2 and p_3 causes different phase changes on different MZIs. The largest phase deviation occurs as the target phase becomes large and thus requires a large amount of power for phase configuration.

To maintain inference accuracy of ONNs, effects of process variations need to be extracted so that the applied power for phase configuration can be adjusted accordingly. In addition, since the phases of MZIs are configured by changing their temperatures by applying power, neighboring MZIs affect each other's temperature, leading to further phase deviation. This effect also needs to be addressed to maintain inference accuracy. Furthermore, training can also be investigated to explore the potential of reducing process variations and thermal imbalance in advance.

In [YS+17], a fully optical architecture with MZIs connected in a grid-like layout [CHM+16] is fabricated to implement multi-layer neural networks. In this architecture, a weight matrix from software training is first decomposed with SVD into three matrices and mapped to MZIs afterwards. To reduce the number of MZIs, in [ZLL+19], one of the decomposed matrices is replaced by a sparse tree structure and the MZI phases are trained directly to achieve a high inference accuracy, and the network structure is further pruned by an FFT-based representation in [GZF+20]. Furthermore, other connection architectures have also been explored, such as the FFTNet architecture [BBA07] and the triangular mesh architecture [RZBB94b]. In addition, ONN designs also include optoelectronic implementation of reservoir computing [BM+13, BMF+18] and photonic spiking processing [RKF+09, TNSP14], demonstrating the potential of optical computing for neural networks.

Since ONNs rely on MZIs for computation and trigonometric functions are involved in their functional representation, the inference accuracy of these networks is very sensitive to process variations and thermal effects of MZIs. Unfortunately, this sensitivity has not been analyzed and addressed up to now, thus hindering the adoption of this high-performance implementation of neural networks in practice.

2.3 Summary

In this chapter, biochips, memristor-based crossbars and MZI-based arrays are introduced as typical emerging systems to improve efficiency for biochemical experiments and neural network computing. Their challenges and the state of the art are reviewed. The working efficiency of biochips is generally affected by their control logic which used to be a binary tree multiplexing structure. To further improve efficiency, the control logic requires an application-specific design. In addition, the control logic with manufacturing faults needs to be addressed.

To accelerate neural network computing, memristor-based crossbars and MZI arrays are proposed to break the von Neumann architecture bottleneck. However, both of them are vulnerable to process variations and noise. MZI arrays are also impacted by power consumption and thermal crosstalk. Frameworks are needed to solve these problems or reduce their impacts.

Chapter 3

Multi-Channel and Fault-Tolerant Control Multiplexing for Flow-Based Microfluidic Biochips

As mentioned previously, the up-to-date control logic for biochips offers only the single-channel switching function, which is also vulnerable to manufacturing faults. In this chapter, a multi-channel switching control logic is presented to improve working efficiency and accompanied with backup paths to improve the fault tolerance. To automatically generate the novel control logic, a three-step framework is presented: 1. Compress the switching patterns by mixing-multiplexing; 2. Distribute control patterns to achieve minimum switching times; 3. Establish an ILP model to construct the new control logic. Six applications are used to test the validation of the framework. The simulation results show an improvement in working efficiency and a resource-saving in biochips.[1] [2]

[1]©2018 IEEE. Reprinted, with permission, from Y. Zhu and B. Li and T. Ho and Q. Wang and H. Yao and R. Wille and U. Schlichtmann, Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips, 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 11/2018.

[2]©2019 IEEE. Reprinted, with permission, from Y. Zhu and X. Huang and B. Li and T. Ho and Q. Wang and H. Yao and R. Wille and U. Schlichtmann, MultiControl: Advanced Control Logic Synthesis for Flow-Based Microfluidic Biochips, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 11/2019.

3.1 Proposed Multiplexing Mechanisms for Control Logic Design and Problem Formulation

The control logic design described in Section 2.1.2 is very effective in reducing the number of pressure sources. However, flow valves are switched sequentially in this scheme by activating control channels individually. During the state transition from time slot t to time slot $t+1$, the execution of an application on the biochip is paused. If the number of valves whose states need to be updated is large, the execution time of the application can be prolonged. This disadvantage is due to the fact that only one output can be updated in a time slice. To solve this problem, a new design scheme that allows multiple control outputs to be activated simultaneously will be introduced in the following to improve the efficiency of the control logic.

In addition, the existing control logic design is also sensitive to manufacturing defects. If a control channel cannot be opened properly, the corresponding flow valve cannot be switched anymore, potentially leading to a complete chip failure. This reliability issue is addressed in the proposed new design scheme with duplicated control paths, which are constructed together with control paths for multi-channel switching to improve design efficiency.

3.1.1 Multi-Channel Switching

In Fig. 2.4, only three flow valves are driven by the control logic, though the combinations of pressure sources are capable of generating four control patterns. Consider the scenario that channel states are switched from "011"→"100". The control logic individually switches the second and the third channel from '1' to '0'. Therefore, it is possible to combine the last two operations. Besides the three control patterns used in Fig. 2.4, there is still the fourth control pattern $\overline{x}_1\overline{x}_2$ available, which can be used to switch the channel 2 and 3 together, as shown in Fig. 3.1(a).

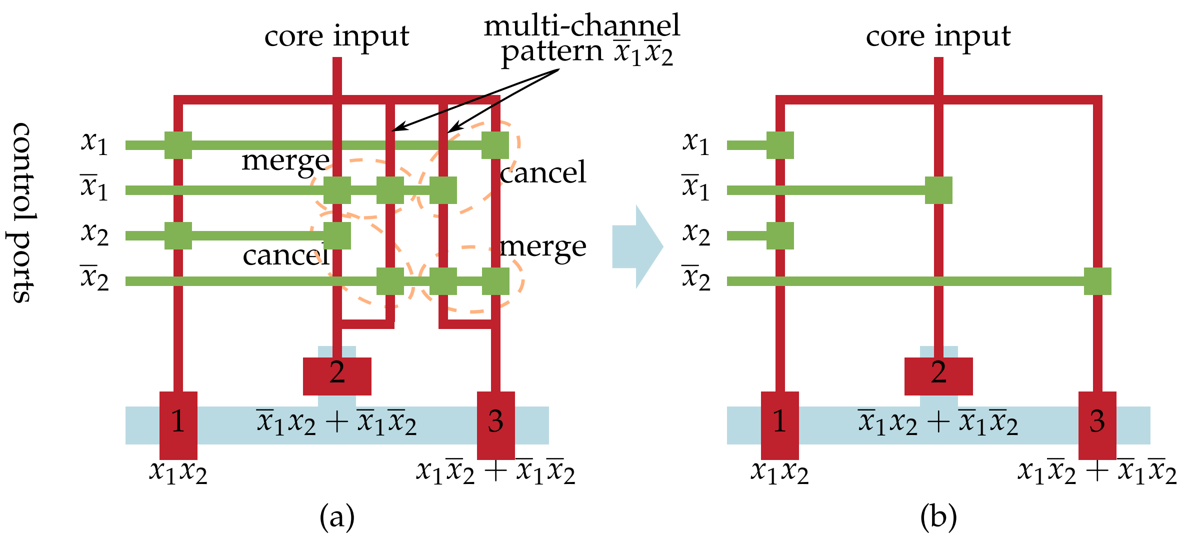

core input
multi-channel pattern $\overline{x}_1\overline{x}_2$
control ports
x_1
$\overline{x}_1$
x_2
$\overline{x}_2$
merge
cancel
cancel
merge
1
2
3
$\overline{x}_1x_2+\overline{x}_1\overline{x}_2$
x_1x_2
$x_1\overline{x}_2+\overline{x}_1\overline{x}_2$
(a)
core input
x_1
$\overline{x}_1$
x_2
$\overline{x}_2$
1
2
3
$\overline{x}_1x_2+\overline{x}_1\overline{x}_2$
x_1x_2
$x_1\overline{x}_2+\overline{x}_1\overline{x}_2$
(b)

Figure 3.1: Control logic with multi-channel switching. (a) Additional control pattern $\overline{x}_1\overline{x}_2$ is used to update control channels 2 and 3 simultaneously. (b) Simplified control logic after valve merging and canceling [ZLH+18] ©2018 IEEE.

We call this "multi-channel switching". In this augmented design, both channels 2 and 3 are directly connected to the core input through the newly added control paths , which is named as the **Direct Connection** (DC) in this dissertation . Consequently, in the transition from "011"→"100", the number of time slices can be reduced by 1. However, in the DC control logic, the number of control valves on every control path is fixed to the number of the complementary control port pairs, e.g., on every path in Fig. 3.1(a) exist two control valves. The control valve number increases with the number of multi-channel switching paths increasing, leading to a high cost. Therefore, a method is needed to reduce the control valve number for a new control logic.

In Fig. 3.1(a), flow valve 3 is driven by two control paths. At the bottom of these two paths, the two control valves are connected to the same control port $\overline{x}_2$. Therefore, they can be merged to save one valve. The two control valves at the top of these two control paths are complementary, since they are connected to x_1 and $\overline{x}_1$. Therefore, no matter what value x_1 has, at least one of the two control paths to flow valve 3 opens on the condition that $\overline{x}_2$ is set to '1'. Accordingly, the two valves at the top of the two control paths to flow valve 3 can be canceled. The merging and canceling operations can also be applied to the control channels to flow valve 2. Consequently, the control logic can be simplified as shown in Fig. 3.1(b), where only one control valve is required in each of the control paths to the control outputs 2 and 3. This merging and canceling process is actually the simplification of the Boolean logic $\overline{x}_1 x_2 + \overline{x}_1\overline{x}_2 = \overline{x}_1$ and $x_1\overline{x}_2 + \overline{x}_1\overline{x}_2 = \overline{x}_2$. The $+$ sign means that either control path can drive the corresponding flow valve sufficiently. In Fig. 3.1(b) the number of valves has been reduced from 10 to 4 compared with Fig. 3.1(a). Compared with the original control logic in Fig. 2.4, the number of valves has also been reduced from 6 to 4, while the multi-channel switching function is still implemented.

In the simplified design in Fig. 3.1(b), the flow valves can still be switched individually, because the individual control patterns $\overline{x}_1 x_2$ and $x_1\overline{x}_2$ are still valid for channels 2 and 3, respectively. For example, the control pattern $x_1\overline{x}_2$ connects only the control channel 3 to the core input, while the other two channels are still closed. Consider a more complex scenario of channel states "011"→"100" →"001"→"110". The transition "100" →"001" requires two time slices for channels 1 and 3, while channel 2 does not need to be updated. The transition "001"→"110" still requires three time slices, since the channels 1 and 2 cannot be updated simultaneously. Consequently, the total number of time slices required by the flow valves can be calculated as the sum of time slices in the time slots, i.e., 2+2+3=7, which is less than the time slices 8 required in the original design in Fig. 2.4, where only single-channel switching is possible.

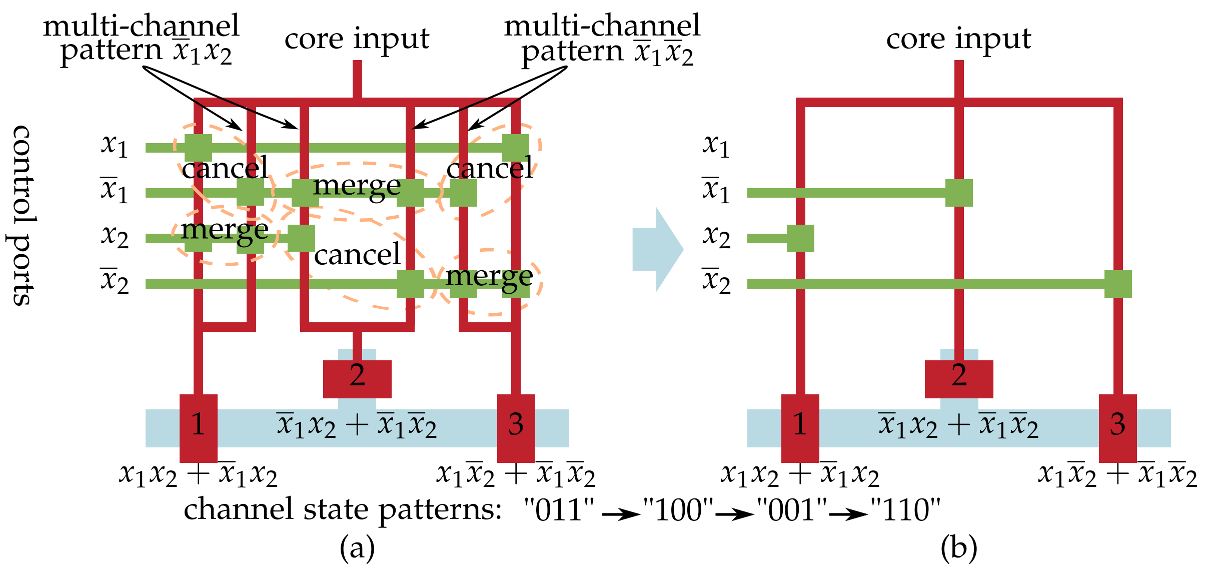

multi-channel pattern $\overline{x}_1 x_2$
core input
multi-channel pattern $\overline{x}_1\overline{x}_2$
control ports
x_1
$\overline{x}_1$
x_2
$\overline{x}_2$
cancel
merge
cancel
merge
cancel
merge
1
2
3
$\overline{x}_1 x_2 + \overline{x}_1\overline{x}_2$
$x_1 x_2 + \overline{x}_1 x_2$
$x_1\overline{x}_2 + \overline{x}_1\overline{x}_2$
channel state patterns: "011" → "100" → "001" → "110"
(a)
(b)

Figure 3.2: Control logic reduction by alternate multi-channel switching. (a) Control pattern $\overline{x}_1 x_2$ is used to update control channels 1 and 2 simultaneously and control channel 2 has no individual control pattern. (b) Simplified control logic [ZLH+18] ©2018 IEEE.

3.1.2 Logic Reduction by Alternate Multi-Channel Switching for Given Applications

In the case in Fig. 3.1(b), the control logic cannot be reduced anymore, since all the spare control patterns have been used. This design still maintains the ability to update each control channel individually, as well as to update the states of the channels 2 and 3 simultaneously. The maintained single-switching ability guarantees that this control logic is capable of generating states of control channels for any applications.

If the application of the biochip is given, the state transitions become known. In a sequence of transitions such as "011"→"100"→"001" →"110", it can be observed that the control channel in the middle is always updated together with another one, either the first or the last. This phenomenon indicates that it is not necessary to assign channel 2 an individual control pattern. Instead, the original control pattern

$\overline{x}_1 x_2$ in Fig. 3.1(a) can be spared to implement multi-channel switching between channels 1 and 2, as shown in Fig. 3.2(a).

In Fig. 3.2(a), control channels 1 and 3 receive individual control patterns $x_1 x_2$ and $x_1 \overline{x}_2$, respectively. The control channel 2, however, can only be switched together with either channel 1 by $\overline{x}_1 x_2$ or channel 3 by $\overline{x}_1 \overline{x}_2$. This loss of generality makes this control logic design suitable only for a given application. But the control logic itself can be simplified and the switching times of valves in executing the application can be reduced.

After the merging and canceling operations are applied to the case in Fig. 3.2(a), only three control valves are left in the design, as shown in Fig. 3.2(b). The logic of the control patterns can be verified from the multi-channel control patterns as $x_1 x_2 + \overline{x}_1 x_2 = x_2$ for channel 1, $\overline{x}_1 x_2 + \overline{x}_1 \overline{x}_2 = \overline{x}_1$ for channel 2, and $x_1 \overline{x}_2 + \overline{x}_1 \overline{x}_2 = \overline{x}_2$ for channel 3. Furthermore, the number of control ports is also reduced by one, since x_1 is not required anymore, leading to a further decrease of the complexity of the biochip platform.

For the state transitions of the control channels "011"→"100"→ "001"→"110", the further simplified control logic requires only 2+2+2=6 time slices, since channel 2 always shares the new value with another channel. A special case is in the transition "100"→"001", where the state of channel 2 does not change. Therefore, the function of the chip is independent of whether the state of the second channel is updated or not, similar to a "don't care" channel in logic design. In the design in Fig. 3.2(b), the pattern $\overline{x}_1 x_2$ takes advantage of this phenomenon for multi-channel switching. Since the number of control valves in the control logic has also been reduced significantly, this comparison confirms that the newly introduced multi-channel switching concept can improve the execution efficiency of the control logic and reduce the resource usage at the same time.

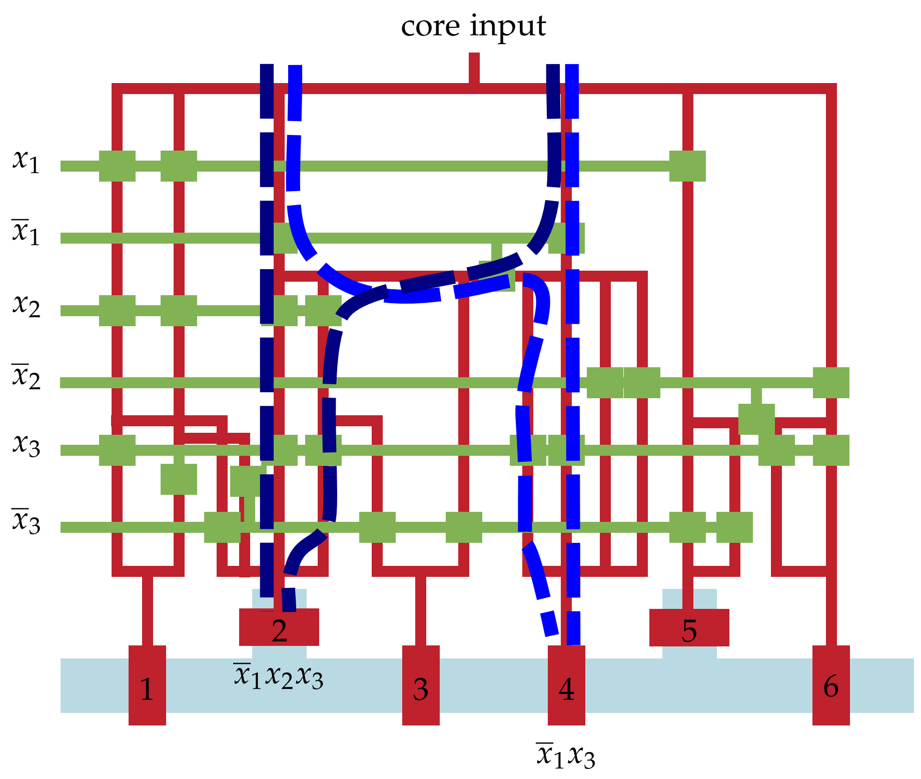


Figure 3.3: Fault tolerance in control logic [ZLH+18] ©2018 IEEE.

3.1.3 Fault Tolerance in Control Layer

In Fig. 3.2(b), there is only one valve and one control path to a control output. During manufacturing, there might be defects in the control logic. If a control valve cannot be closed, the core input is always connected to the control channel, leading to a failed flow valve in the biochip. To tackle this problem, a control valve can be duplicated and inserted in series to the original control valve, similar to the solution in [HGR+17]. On the other hand, if a control valve cannot be opened or a control path is blocked, there is no path to connect the core input to the control output to update its state. A simple strategy to solve this problem is to duplicate all the channels and valves and insert them in parallel to the original channels and valves. This method, however, may lead to an unnecessarily complicated design and large resource usage.

Fig. 3.3 shows another example of control logic generated by the proposed method, where the control paths along control valves to control outputs 2 and 4 are highlighted. In this case, the control pattern $\overline{x}_1x_2x_3$ activates these two outputs simultaneously, forming a multi-channel switching pair. Furthermore, to each of these control outputs, there are two independent paths through the control logic. If one of these paths is blocked due to a manufacturing defect, the other path still maintains the correct function of the control logic.

Compared with the straightforward strategy to duplicate the control logic for fault tolerance, the control paths in Fig. 3.3 share control valves, e.g., the two valves connected to $\overline{x}_1$, leading to a reduction of resource usage. To design such a control logic with efficient multi-channel switching and resource sharing for fault-tolerance, these features need to be considered together in a general framework.

3.1.4 Problem Formulation

Based on the new mechanisms discussed above, the control-logic design considering control multiplexing and fault-tolerance can be formulated as follows:

Input: The states of all flow valves/control channels at every moment in a given biochemical application.

Output: An optimized control logic supporting multi-channel switching and fault tolerance.

Objective: (1) Minimize the number of time slices for channel switching, (2) minimize the number of control valves, and (3) minimize the total control-channel length.

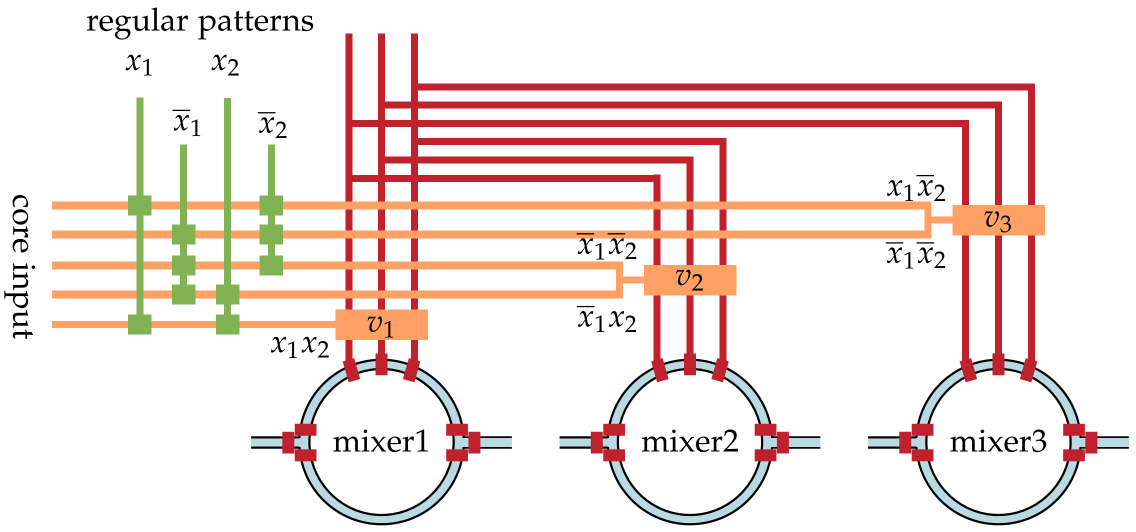
3.2 A General Framework for Control Multiplexing and Fault Tolerance

To generate a control logic supporting multi-channel switching and fault tolerance, a general framework including two major steps is adopted in this section. First, the given control channel states are converted to channel switching patterns. Then control channels that can be enabled simultaneously are identified to reduce the total number of time slices. In the second step, control channels are constructed to meet the multi-channel switching and fault-tolerance requirements and thus generate the final control logic.

3.2.1 Switching States Compression by Mixing Multiplexing

The complexity of control logic is affected by the flow-valve states to be generated. Generally more valve states lead to more control channels and valves. In practice, a large number of valve states are actually generated by mixers. As shown in Fig. 3.4 (a), each of the three mixers has three flow valves at the top to create a circular flow for peristalsis mixing. This function requires these valves to be switched with a high frequency within a given time period. The flow-valve states need to repeat a given pattern series, e.g., "010"→"011"→"001"→"101"→"100"→"110" [MQ07]. Assume that each of the three mixers in Fig. 3.4 (a) is activated by this pattern series once, but at a different time. An exemplary flow-valve switching states are thus shown as in Fig. 3.4 (b), where the bold patterns highlight the valves need to be switched (in total 21 states should be switched in this example).

In a mixer, the flow valves for peristalsis are only used to create a circular flow with the given pattern series, no matter from which pattern the series starts. In other words, the switching series can be rotated, as long as the whole pattern series is repeated. To compress switching times, we take advantage of this feature



(a)

mixer1	mixer2	mixer3
000	000	000
000	000	000
010	000	000
011	000	000
001	000	000
101	**010**	000
100	**011**	000
110	**001**	000
000	**101**	**010**
000	**100**	**011**
000	**110**	**001**
000	**000**	**101**
000	000	**100**
000	000	**110**
000	000	**000**
000	000	000

(b)

regular patterns	$v_1v_2v_3$
000	111
000	111
010	**011**
011	011
001	**011**
101	**001**
100	001
110	**001**
010	**100**
011	100
001	**100**
101	**110**
100	110
110	**110**
000	**111**
000	111

(c)

Figure 3.4: Switching states compression by mixing multiplexing. (a) Structure of mixing multiplexing. (b) Switching states when peristalsis valves in mixers are controlled separately. (c) Switching states with mixing multiplexing [ZHL+19] ©2019 IEEE.

by driving all mixers with the same peristalsis patterns, as shown in Fig. 3.4 (a). The three control ports at the top of this structure provide a repeating regular pattern series for all the peristalsis valves. The real connection of these ports to the peristalsis valves in the mixers are controlled by newly introduced valves v_1, v_2 and v_3. Therefore, the switching states in Fig. 3.4 (b) can be converted into Fig. 3.4 (c), where v_1, v_2 and v_3 are opened with a low pressure in their control channels when the mixers start, and they are closed with a high pressure when the mixers stop. Since the regular patterns are shared by all mixers and switch very often, they are generated by external pressure sources directly. The control logic only needs to generate the corresponding states for v_1, v_2 and v_3. Compared with the original direct control of peristalsis valves in Fig. 3.4 (b), the control logic only needs to produce 5 switching activities, fewer than a fourth of the switching activities in Fig. 3.4 (b).

To implement mixing multiplexing, the original switching states are examined. For each mixer, a new valve is created on the control paths to its peristalsis valves. These valves are considered control valves and the original peristalsis valves are removed from the control patterns. Consequently, the control logic can generate the control patterns such as x_1x_2, $\overline{x}_1x_2$ and $x_1\overline{x}_2$ to control v_1, v_2 and v_3, respectively. In addition, mixers can be activated by multi-channel switching. For example, mixer2 and mixer3 in Fig. 3.4(c) can be activated by the control pattern $\overline{x}_1\overline{x}_2$ simultaneously. This concept of mixing multiplexing is a pre-processing step for control logic construction, and it scales well as the number of mixers in the chip increases, since for each mixer only the control patterns of one valve need to be generated, which only mark the starting and stopping time of the mixer and thus do not switch often.

3.2.2 Computation of Multi-Channel Switching Scheme

As discussed in Section 3.1.2, the number of time slices of the control logic and the resource usage can be reduced significantly if the control channel states required for the application are considered. These states are written as a ***state matrix*** $\tilde{P}$, whose rows represent the states of all control channels at different moments. For example, for the states of the transitions "011"→"100"→"001" →"110", $\tilde{P}$ is written as

$$\tilde{P} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \tilde{Y} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & X & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{3.1}$$

In a transition such as "011"→"100", the second and the third control channels need to be connected to the core input with its pressure value set to '0'. Afterwards, the first control channel needs to be connected to the core input and the pressure value of the core input should be set to '1'. In both cases, it is the responsibility of the control logic to connect the corresponding control channels to the core input. These requirements to the control logic can be represented by a ***switching matrix*** $\tilde{Y}$ derived from the state matrix $\tilde{P}$. In this matrix, a '1' represents that the corresponding control channel is connected to the core input and its state is updated to the same as that of the core input; a '0' indicates no update of the corresponding control channel. Therefore, these rows are called ***switching patterns***. As an example, the switching matrix of $\tilde{P}$ in (3.1) is also shown as $\tilde{Y}$. Note that in the transition "100"→"001", when the first channel is updated to '0', the second channel can be updated together with the first channel, or it can be ignored since its state does not change. Accordingly, a don't care 'X' appears. In reality, multiple '1's in a row in $\tilde{Y}$ may not be updated simultaneously, in case this specific multi-channel combination is not implemented. Therefore, such a row needs to be split into time slices so that the corresponding channels are updated by several operations. To reduce the overall number of time slices, the multi-channel combinations need to be selected carefully.

In a general case, assume that the switching matrix is written as

$$\tilde{Y} = \begin{bmatrix} Y_0 \\ Y_1 \\ \cdots \\ Y_{M-1} \end{bmatrix} = \begin{bmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N-1} \\ \cdots & \cdots & \ddots & \cdots \\ y_{M-1,0} & y_{M-1,1} & \cdots & y_{M-1,N-1} \end{bmatrix} \tag{3.2}$$

where $y_{i,j}$ is a constant taking one of the values '0', '1' or 'X'. M is the number of transitions in which at least one channel should be switched. N is the number of control channels.

As discussed above, a row in $\tilde{Y}$ may contain multiple '1's that cannot be implemented simultaneously. Consequently, the corresponding time slot of switching these control channels needs to be split into several time slices. The objective is that the overall number of time slices implementing the switching matrix $\tilde{Y}$ is reduced. To fulfill this objective, the potential multi-channel switching combinations need to be examined.

For N control channels, there are $2^N - 1$ possible combinations to form multichannel scheme, defined by the ***multiplexing matrix*** $\tilde{X}$ with N columns, as

$$\tilde{X} = \begin{bmatrix} X_0 \\ X_1 \\ ... \\ X_{2^N-2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3.3}$$

where each row represents a possible combination of control channels to form the multi-channel switching. If an item $x_{i,j}$ in $\tilde{X}$ is '1', the corresponding control channel is included in the multi-channel switching combination.

Since the objective of multi-channel switching is to select proper combinations of rows in $\tilde{X}$ to implement the switching matrix $\tilde{Y}$, a ***selection matrix*** $\tilde{T}$ of M rows and $2^N - 1$ columns is defined as follows

$$\tilde{T} = \begin{bmatrix} t_{0,0} & t_{0,1} & \cdots & t_{0,2^N-2} \\ t_{1,0} & t_{1,1} & \cdots & t_{1,2^N-2} \\ \cdots & \cdots & \ddots & \cdots \\ t_{M-1,0} & t_{M-1,1} & \cdots & k_{M-1,2^N-2} \end{bmatrix} \tag{3.4}$$

where the i-th row defines which rows in $\tilde{X}$ are selected to implement the switching pattern in the ith row of $\tilde{Y}$ in (3.2).

For example, Fig. 3.5 shows the multiplexing matrix and a feasible selection matrix corresponding to the switching matrix in (3.1). There are a total of seven channel combinations and four of them are selected to implement the switching patterns defined in $\tilde{Y}$. Note that the last switching pattern in $\tilde{Y}$, i.e., '110', is split into two time slices to update the states of the first two control channels. More specifically, the channel combinations '100' and '010' are selected to update the states of the two channels sequentially.

$$\tilde{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \tilde{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.5}$$

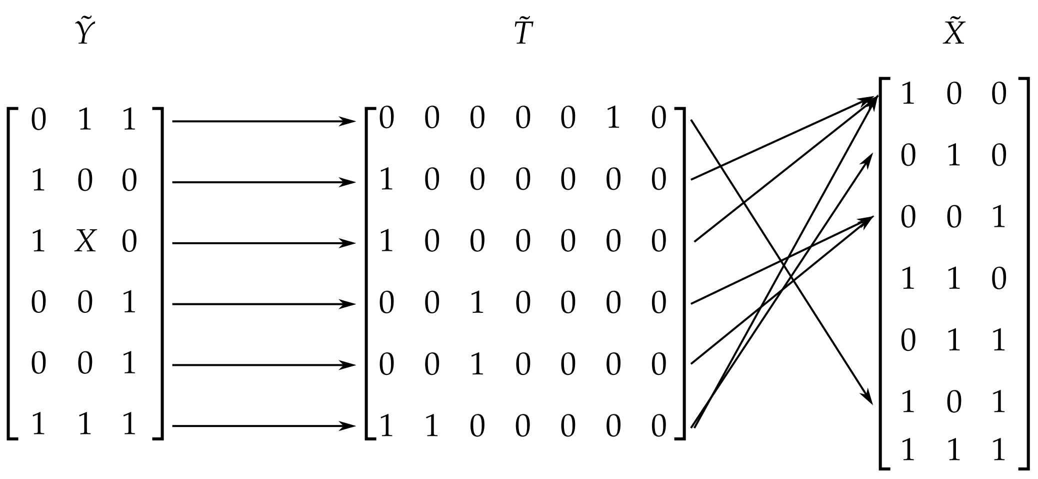


Figure 3.5: Multiplexing matrix and a feasible selection matrix corresponding to the switching matrix in (3.1) [ZHL+19] ©2019 IEEE.

In a row in (3.2), if an item $y_{i,k}$ is '1', meaning that this control channel must be activated once, it must be covered by at least one of the rows in $\tilde{X}$ that has a '1' at the corresponding column. This constraint can be expressed as

$$\sum_{j=0}^{j=2^N-2} t_{i,j}x_{j,k} \begin{cases} \geq 1, & y_{i,k}=1 \\ =0, & y_{i,k}=0 \end{cases} \tag{3.6}$$
$$\forall i = 0, \ldots M-1,\ k = 0, \ldots N-1$$

where $x_{i,j}$ and $y_{i,k}$ are given constants. $t_{i,j}$ are 0-1 variables whose values are determined by a solver.

In a control logic, the maximum number of allowed control patterns is usually given or constrained by the number of external pressure sources as a constant $Q_{cw} = 2^{\lceil log_2 N \rceil}$ and usually $Q_{cw} \ll 2^N - 1$. Accordingly, for each row in $\tilde{X}$, a 0-1 variable l_i is defined to indicate whether the corresponding combination is selected. The total number of selected combinations should be no larger than Q_{cw}, constrained as

$$\sum_{i=0}^{2^N-2} l_i \leq Q_{cw}. \tag{3.7}$$

If row j in $\tilde{X}$ is not selected so that $l_j = 0$, all the selection variables in column j in

$\tilde{T}$ must be set to 0, constrained as

$$t_{i,j} \leq l_j, \quad \forall i = 0, \ldots M-1, \; j = 0, \ldots 2^N - 2. \tag{3.8}$$

Since a row in $\tilde{T}$ represents which multi-channel switching combinations from $\tilde{X}$ are selected to implement the switching patterns in the corresponding row in $\tilde{Y}$, the number of '1's in this row in $\tilde{T}$ represents the required number of time slices. To minimize the total number of slices, the following optimization problem can be solved

$$\text{minimize} \quad \sum_{i=0}^{M-1} \sum_{j=0}^{2^N-2} t_{i,j} \tag{3.9}$$

$$s.t. \quad (3.6), \; (3.7), \; (3.8). \tag{3.10}$$

After solving (3.9)–(3.10), the combinations of channels to implement multi-channel switching are determined by the values of l_i. The values of $t_{i,j}$ specify how the switching patterns in $\tilde{Y}$ can be realized by these control patterns to reduce the overall switching times.

For an application, the number of rows in the switching matrix $\tilde{Y}$ might be large, making (3.9)–(3.10) very difficult to solve. In practice, many rows in the switching matrix $\tilde{Y}$ might be equal. For example, a typical application contains many mixing operations, which use only a few switching patterns repeatedly. In the proposed method, these rows are merged and the number of merged rows are multiplied with $t_{i,j}$ in (3.9) to reduce the scale of the problem. Another deployed technique to reduce the scale of (3.9)–(3.10) is that in the multiplexing matrix, the maximum number of channels that are allowed to switch simultaneously is constrained to a given number. This is acceptable because the case that a large number of channels are updated simultaneously is not common in reality. In the experiments, this maximum number is set to 3.

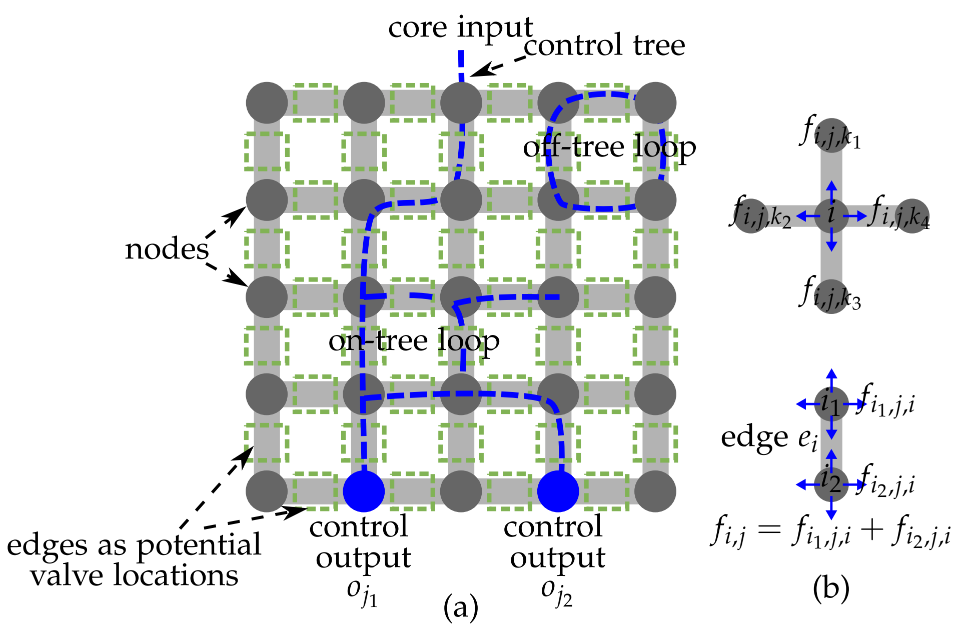


Figure 3.6: Routing grid. (a) A control tree constructed on a routing grid. (b) Modeling variables representing flow volumes and directions for control tree construction [ZLH+18] ©2018 IEEE.

3.2.3 Control Logic Construction on a General Routing Grid

After determining the multi-channel switching scheme, the values of the l_i carry the information which control channels should be connected to the core input together. Since the states of control channels are the same as those at the outputs of the control logic, it is then the task of the control logic to generate correct patterns at its outputs to drive control channels.

When multiple control outputs are updated by a control pattern, a control path should be constructed for each of them. Since these paths can also share segments with each other, a ***control tree*** is constructed, as illustrated in Fig. 3.6. In reality, the control tree can be very complex, even with many branches dangling in the middle of the control tree.

In the proposed method, a general routing grid is used as virtual guide to construct control trees. Such a grid is composed of a set of horizontal and vertical edges, and edges join other edges at nodes. On this routing grid, a path can be viewed as a series of consecutive connected edges. On each edge, a control valve can be built. If no valve is built, but the edge still appears in the final control logic, it then always connects the two nodes at its ends. If in the end an edge does not appear in the control logic, the two nodes at the ends of this edge are always not connected directly. An example routing grid is shown in Fig. 3.6 (a), where o_{j_1} and o_{j_2} are control outputs.

For an edge e_i on the routing grid, a 0-1 variable e_i^{exist} is used to indicate whether the edge appears in the control logic. In addition, a 0-1 variable v_i^{exist} is used to indicate whether a control valve v_i is built on the edge e_i. If a control valve appears on the control tree to drive the control output o_j, it must be switched open when o_j is activated. This open/closed state is denoted by the 0-1 variable $v_{i,j}^{state}$. The relation between these variables can be written as

$$v_{i,j}^{state} \le v_i^{exist} \le e_i^{exist}, \quad \forall e_i \in \mathcal{E}, c_j \in \mathcal{C} \tag{3.11}$$

where $\mathcal{E}$ is the set of all edges on the routing grid, and $\mathcal{C}$ is the set of all control patterns.

To construct a control tree in the control logic, the connection state of an edge should be defined first. In two scenarios, an edge connects the two nodes at its ends: 1) an edge appears in the control logic but there is no valve built on it; 2) a valve is built on an edge that appears in the control logic, and the valve is open due to the control port driving it. Such an edge is called a ***connection edge*** in the following. If the 0-1 variable $c_{i,j}$ is used to indicate whether the edge e_i connects the nodes at its two ends when the jth control pattern is applied, $c_{i,j}$ can be constrained as

$$c_{i,j} = e_i^{exist} - v_i^{exist} + v_{i,j}^{state}. \tag{3.12}$$

With the connection states of edges defined above, the control tree can thus be described accordingly. To construct a control tree, the idea is to use the concept of a flow from the core input. The flow fills the edges it passes and only reaches the control outputs that should be activated.

For each node n_i in the routing a grid, a flow value $f_{i,j,k}$ is defined with respect to control pattern c_j and the kth edge that is incident to n_i directly, as shown in Fig. 3.6(b). If a flow enters a node, the flow value is negative. If it leaves a node, its flow is positive. Since a node only connects edges but cannot store a flow volume, for node n_i that does not correspond to the core input or a control output that should be activated, the relation between the flow variables can be written as

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} = 0 \tag{3.13}$$

where $\mathcal{E}_i$ is the set of edges incident to node n_i directly.

For each edge e_i, the variable $f_{i,j}$ is defined to represent whether the edge stores one unit of the flow, and is determined by the flows entering the edge from the two nodes at its ends and could be one only when the edge is open. The relation is expressed as

$$f_{i,j} = \sum_{n_k \in \mathcal{N}_i} f_{k,j,i} \leq c_{i,j} \tag{3.14}$$

where $\mathcal{N}_i$ is the set of nodes at the ends of edge e_i, and $f_{k,j,i}$ is a flow value to e_i.

To form a control tree from the core input to control output o_j, the nodes in-between must function as connecting points. A 0-1 variable $n_{i,j}$ is defined to represent whether node n_i is in the tree or not. For a node in the tree, at least an edge incident to it should be filled by the flow. The connection condition for node n_i is

$$n_{i,j} \leq \sum_{e_k \in \mathcal{E}_i} f_{k,j} \leq 4 \cdot n_{i,j} \tag{3.15}$$

where $\mathcal{E}_i$ is the set of edges connecting to n_i.

Since the core input needs to provide sufficient flow to fill the edges in the control tree and the flow must only reach the current control output o_j, the following constraints can be established:

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} > 0, \quad n_i = n_{core} \tag{3.16}$$

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} < 0, \quad n_i \text{ represents an opened output } o_j. \tag{3.17}$$

For the outputs that are closed, the nodes representing them should not appear on the control tree. Since an edge that is in the connection state makes its two nodes share the same status, $n_{i,j}$ needs to be constrained as

$$n_{i,j} = 0, \ \forall n_i \text{ representing a closed output} \tag{3.18}$$

$$c_{i,j} - 1 \le n_{k_1,j} - n_{k_2,j} \le 1 - c_{i,j}, \ \forall e_i \in \mathcal{E} \tag{3.19}$$

where $n_{k_1,j}$ and $n_{k_2,j}$ are the two nodes of e_i, and $\mathcal{E}$ is the set of all edges.

The constraints above can be used to generate a control tree shown in Fig. 3.6(a). These constraints do not prohibit an on-tree loop such as that in Fig. 3.6(a) from happening. The existence of on-tree loops, however, does not affect the correct function of the control logic. The off-tree loop in Fig. 3.6(a) is excluded by the constraints (3.11)–(3.19). However, these constraints are only sufficient conditions to construct a control tree. Off-tree loops can still appear, provided that they do not activate the current control output.

The flow value $f_{i,j}$ defines whether an edge is required to control an output. If a valve appears on the edge, its connection to the control ports needs to be determined, so that it can be switched correctly by an external pressure source. Assume there are N_p control ports. Since a control valve can be connected to any of these N_p ports, for the control valve v_i, N_p 0-1 variables $p_{i,1}, p_{i,2}, \dots p_{i,N_p}$ are defined. The variable $p_{i,k}$ represents whether control valve v_i is connected to the kth control port. Since a control valve can only be controlled by one port when a control valve exists on an edge, these variables are constrained as

$$\sum_{k=1}^{N_p} p_{i,k} = v_i^{exist}. \tag{3.20}$$

For the jth control tree, corresponding to the jth control pattern, assume the states of the control ports are denoted by 0-1 variables $s_{j,1}, s_{j,2}, \dots s_{j,N_p}$. For the control valve v_i, its state corresponding to the jth control pattern is written as $v_{i,j}^{state}$. Since all the valves controlled by the same control port must have the same state in a control pattern, the valve states can be constrained as

$$p_{i,k} - 1 \le v_{i,j}^{state} - s_{j,k} \le 1 - p_{i,k}, \quad \forall e_i \in \mathcal{E},\ k \in \{1, \dots N_p\} \tag{3.21}$$

where $\mathcal{E}$ is the set of all edges.

In a control logic, the control patterns should be different in activating different control outputs or their combinations. Therefore, when regarded as binary numbers, the values of the control patterns are different from each other. This condition can be specified as

$$B_j = 2^0 \cdot s_{j,1} + 2^1 \cdot s_{j,2} + \dots + 2^{N_p - 1} \cdot s_{j,N_p} \tag{3.22}$$

$$B_{j_1} - B_{j_2} \le -1 + My,\ \forall j_1 \ne j_2 \tag{3.23}$$

$$B_{j_1} - B_{j_2} \ge 1 - (1 - y)M,\ \forall j_1 \ne j_2 \tag{3.24}$$

where M is a large number, y is an intermediate 0-1 variable, where $y = 1$ if and only if $B_{j1} > B_{j2}$ and $y = 0$ if and only if $B_{j1} < B_{j2}$.

The constraints described in this section are very general. To implement multi-channel switching, a control tree needs to activate multiple control outputs simultaneously. Accordingly, these active outputs can simply be enabled by adding constraints similar to (3.17), so that the control tree drives multiple control outputs at the same time.

To implement backup paths for fault tolerance, a path needs to be identified from a control tree first. For example, in Fig. 3.6, the on-tree loop does not need to be duplicated, because the direct path between the core input and the control output is already sufficient for state updating. To identify a path in the control tree, a model similar to that used to identify the control tree can be deployed. In constructing the control tree, the edges are chained one after another. To constrain a path instead of a tree, the only change to be made is that a node in the routing grid that represents neither the core input nor the control output is only allowed to connect exactly two edges, in contrast to (3.15), where more than two edges can be connected to a node to allow more freedom to the patterns of control ports. For fault tolerance, two identified control paths that are backup to each other should not share any parts on the routing grid. This constraint can be specified as that the edges covered by the paths should not overlap, so that the variables indicating that an edge belonging to these paths should be 1 only for one of the fault-tolerant paths. These constraints are similar to those for constructing control trees discussed above, and are not discussed in detail due to limited space.

With the constraints defined, the control logic can be constructed by creating a control tree for each control pattern and solving the resulting ILP problem as

$$\text{minimize} \quad \sum_{i=0}^{|\mathcal{E}|-1} v_i^{exist} \tag{3.25}$$

$$s.t. \quad (3.11)\text{–}(3.24). \tag{3.26}$$

To improve the efficiency of the formulation (3.25-3.26), two heuristic techniques have been applied. First, the control ports are only allowed to control the valves in the rows and columns that are neighboring to them in the routing grid. Second, the routing grid is partitioned into sub-blocks and the formulation (3.25-3.26) is applied to each sub-block to solve the problem hierarchically. In addition, pressure degradation in control trees has also be considered, which is not explained in detail due to space limit.

3.3 Simulation Results

The proposed method was implemented in C/C++ and tested on a PC with 2.4 GHz CPU and 32GB memory. We demonstrate the results of three real-life biochemical

Table 3.1: Details of benchmarks used in this paper [ZHL+19] ©2019 IEEE.

Benchmarks (#M_x,#F_v,#C_s,#S_p,#I_p)	
RA30: (2,19,10221,13408,86)	R0: (1,22,5000,6684,153)
mRNA: (3,37,5361,1403,52)	R1: (2,27,6000,8013,244)
CPA: (3,25,2941,1409,92)	R2: (3,48,127000,9372,325)

applications that are CPA (Colorimetric Protein Assay) used in RA30 chip from [LLY+17a], IVD (Int-Vitro Diagnostics) applied in CPA chip from [LLY+17a], and mRNA chip from [MAQ06]. In addition, three randomly generated sequences of switching patterns R0, R1, and R2 are tested to demonstrate the characteristics of the proposed method further.

The details of aforementioned benchmarks are listed in Table 3.1, where #M_x is the numbers of mixers used in the applications and #F_v is the number of flow valves/control channels. The numbers of states of flow valves in executing the corresponding applications are reported in #C_s and the numbers of switching patterns corresponding to the rows of the switching matrix $\tilde{Y}$ in (3.2) are reported in #S_p. After merging equivalent rows of switching patterns as described in Section 3.2.2, the numbers of independent patterns used in (3.9)–(3.10) are reported in #I_p. In our simulations, the control logic itself has two layers to implement the multiplexing function. In the flow part, the flow channels and control channels also form a two-layer structure. Therefore, the whole chip can be considered as two two-layer blocks interfaced by the control outputs.

Verification of mixing-multiplexing control architecture

In Section 3.2.1, we presented a mixing-multiplexing control architecture (MMCA) by switching the states of flow valves in mixers in a centralized manner, and thus further improve the efficiency of a control logic. To verify the effectiveness of mixing multiplexing, we compare the design results between MMCA with the ar-

Table 3.2: Validation of the proposed multi-channel switching mechanism [ZHL$^+$19] ©2019 IEEE.

Bench	Control architecture with individual mixing							Control architecture with mixing multiplexing						
	#T_s	#T_m	Imp (%)	#N_c	#A_p	#C_p	Time (s)	#T_s	#T_m	Imp (%)	#N_c	#A_p	#C_p	Time (s)
RA30	27025	15247	43.6	19	17	10	1307.0	27025	10080	62.7	19	12	11	559.5
CPA	4198	1742	58.5	25	22	10	2146.7	4198	1065	74.6	25	13	11	846.3
mRNA	4464	1597	64.2	37	20	12	3159.5	4464	1055	76.4	37	18	13	2303.6
R0	6891	6799	1.3	22	26	10	3981.4	6891	5090	26.1	22	20	13	2465.5
R1	14334	9776	31.8	27	28	10	4002.9	14334	6179	56.9	27	24	13	3942.3
R2	26058	11781	54.8	48	51	12	10171.2	26058	7481	71.3	48	46	15	9541.8
Average	—	—	42.4	—	—	—	—	—	—	61.3	—	—	—	—

chitecture in which mixers are controlled individually (IMCA). Fig. 3.7 shows the comparison results on the total number of valve-switching times. Overall MMCA achieves a 25%–39% switching times reduction across all the benchmarks, with an average reduction of 34%. This significant reduction of switching times will further improve the execution efficiency of biochips. Moreover, as shown in Fig. 3.8, compared with IMCA, the total number of applied control patterns is also reduced by 10%–29% in the benchmarks, with an average reduction of 21%. This result implies that MMCA has a greater potential to realize a large-scale multi-channel control.

Validation of multi-channel switching mechanism

As discussed previously, in the traditional single-channel switching mode, the '1's in a switching matrix must be updated individually, leading to a low-efficiency control system. To validate the efficiency of the proposed multi-channel switching mechanism, we compare the synthesis results of two switching modes in both IMCA and MMCA in Table 3.2.

In IMCA, the total numbers of time slices in the single-channel switching mode are reported in the column $\#T_s$ in Table 3.2. With multiple-channel switching, these numbers are reduced significantly in most cases, as shown in the column $\#T_m$. The reduction of these switching times can reach up to 64.2%, as shown in the column Imp.

The numbers of control patterns used in the control logic are shown in the column $\#A_p$, which are larger than the numbers of control channels in the column $\#N_c$ due to the additional control patterns for multi-channel switching for cases R0, R1 and R2, while being slightly smaller in cases RA30, CPA and mRNA since several flow valves in these cases always activate simultaneously with other valves so that their control patterns are shared. It can be observed that with a limited increase of the number of control patterns, a significant reduction of switching times (42.4% on average) from $\#T_s$ to $\#T_m$ can be achieved. Moreover, the number of control ports used in the control logic are reported in the column $\#C_p$.

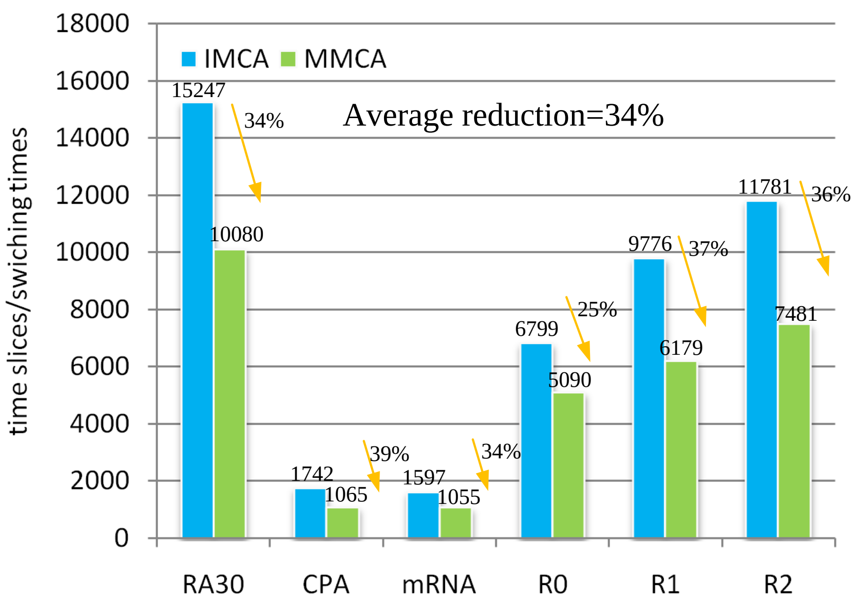


Figure 3.7: Comparison on the number of valve-switching times [ZHL+19] ©2019 IEEE.

In MMCA, compared with the single-channel switching mode, the proposed multi-channel switching mechanism achieves a 26.1%–76.4% time-slice reduction, with an average reduction of 61.3%. Furthermore, in all benchmarks, the numbers of control patterns used in the control logic are fewer than the numbers of control channels, this is achieved by a slight increase of the number of control ports. This result, on one hand, demonstrates the high efficiency of our multi-channel switching scheme, and meanwhile further confirms the effectiveness of the proposed MMCA.

The CPU time to generate the control logic by the proposed method is reported in the columns *Time*. It can be seen that all results can be generated within an acceptable time.

In addition, in determining multi-channel switching patterns, the maximum number of control channels that can be switched together is bound to a given number to increase the implementation efficiency. The reduction of valve-switching times with different bounds is shown in Fig. 3.9. As expected, the reduction increases

Table 3.3: Comparison on the cost of control logic [ZHL+19] ©2019 IEEE.

Bench	Control architecture with individual mixing						Control architecture with mixing multiplexing					
	DC		ILP		Imp (%)		DC		ILP		Imp (%)	
	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$
RA30	270	628	137	443	49.26%	29.46%	184	444	75	323	59.24%	27.25%
CPA	340	790	179	629	47.35%	20.38%	280	652	104	464	62.86%	28.83%
mRNA	510	1152	212	949	58.43%	17.62%	420	986	213	820	49.29%	16.84%
R0	515	1213	274	948	46.80%	21.85%	415	975	172	740	58.55%	24.10%
R1	490	1148	330	942	32.65%	17.94%	555	1308	332	1110	40.18%	15.14%
R2	1014	2317	812	2292	19.92%	1.10%	1188	2723	1170	2669	1.52%	1.98%

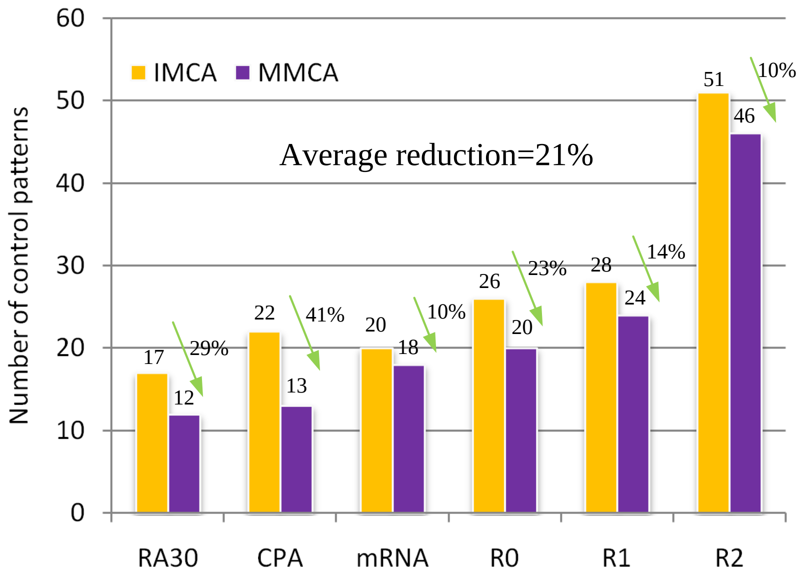


Figure 3.8: Comparison on the number of applied control patterns [ZHL+19] ©2019 IEEE.

as the number of channels that can be switched together increases. However, a further increase from 3 to 4 does not lead to significant performance improvement, justifying the bound set in our method. For case R2 the reduction even decreases slightly due to the heuristics introduced in the proposed formulation to improve computing efficiency.

Validation of control logic construction

We compare the ILP method with the DC method [MQ07] in terms of the cost of the final control logic, including the number of control valves and the total channel lengths (see Table 3.3). In IMCA, the proposed method achieves a 19.92%–58.43% control-valve reduction and a 1.10%–29.46% channel-length reduction across all benchmarks. In MMCA, the number of control valves and the total channel lengths can also be reduced by 45.27% and 19.02% on average, respectively. In other words, the ILP method reduces the control logic cost from the DC method.

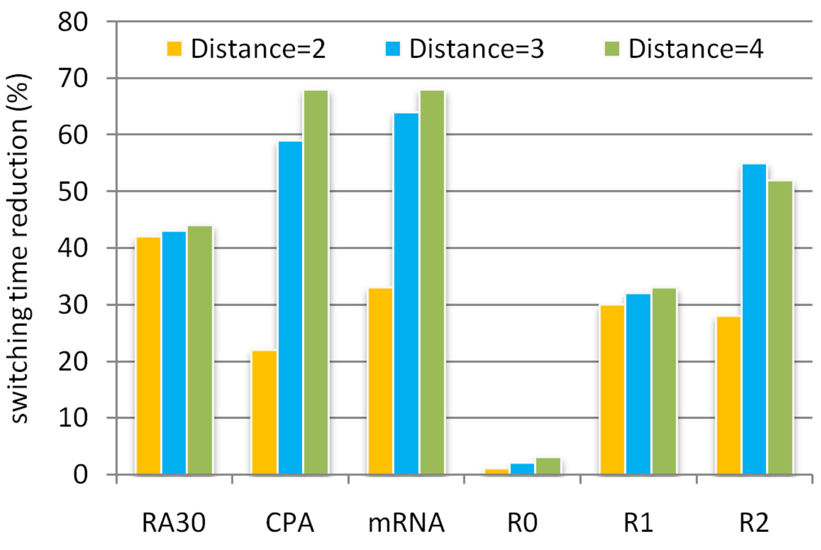


Figure 3.9: Reduction of total channel switching times under different multiplexing distances [ZHL+19] ©2019 IEEE.

3.4 Summary

We have studied the control-logic design problem, considering both control multiplexing and fault-tolerance in flow-based microfluidic biochips, and presented a systematic method to efficiently solve it. By introducing the multi-channel switching and mixing multiplexing mechanisms, the time slices required for switching valves can be reduced significantly. Moreover, independent backup paths have also been introduced to improve the reliability of automatically generated control logic. Using these concepts, a general framework based on an ILP formulation is proposed. Simulation results have shown that our method can generate a control logic with high efficiency, low cost, and fault tolerance within a short time.

Chapter 4

Statistical Training for Memristor-based Crossbars Considering Process Variations and Noise

Memristor-based crossbars, as one of the most promising neuromorphic computing structures, whose working principle has been demonstrated in Section 2.2.2, are vulnerable to process variations and noise. In this chapter, a statistical training method is proposed to maintain the inference accuracy with weights with variations. Firstly, it decomposes process variations into independent statistical variables via Principal Component Analysis (PCA) in Section 4.1. Then the statistical training method introduces the independent statistical variables into the traditional training framework, whose modified operations in forward propagations and cost functions are explained in Section 4.2 and 4.3, respectively. After that, the proposed method is supplemented with the global variation test and compensation to further maintain inference accuracy in Section 4.4. Results are presented in Section 4.5.[1]

[1]©2020 IEEE. Reprinted, with permission, from Y. Zhu and G. L. Zhang and T. Wang and B. Li and Y. Shi and T. Ho and U. Schlichtmann, Statistical Training for Neuromorphic Computing using Memristor-based Crossbars Considering Process Variations and Noise, 2020 Design, Automation Test in Europe Conference Exhibition (DATE), 03/2020.

4.1 Principal Component Analysis (PCA) of Process Variations and Canonical Form of Weights

As mentioned previously, process variations are inherent in the manufacturing process [AR+16]. They consist of global variation and local variations. The former is shared by all memristors and the latter are specific to individual memristors. In the manufacturing process, local variations are correlated [XZH07], and the shared global variation also increases the correlation between memristors. The overall correlation between the memristors can be expressed using a covariance matrix **R** [XZH07].

This matrix can be decomposed using PCA [Seb09] to facilitate the computation during training, as

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T \tag{4.1}$$

where $\mathbf{\Sigma} = diag(\lambda_1, \lambda_2, ..., \lambda_N)$ contains the eigenvalues of **R**, and $\mathbf{V} = [V_1, V_2, ..., V_N]$ contains the corresponding eigenvectors which are orthogonal to each other.

Assume that the variations of all the memristors are written together as **D**. After the decomposition in (4.1), **D** can be expressed as

$$\mathbf{D} = \mathbf{V} \cdot \mathbf{\Sigma}^{0.5} \cdot \mathbf{B} \tag{4.2}$$

where $\mathbf{B} = [B_1, B_2, \ldots, B_N]$ are independent random variables. The representation in (4.2) expresses the variations on memristors as linear combinations of independent random variables. To reduce computation complexity, the independent random variables corresponding to small eigenvalues can also be discarded without affecting the modeling accuracy significantly.

Due to process variations, the actual conductance values programmed into memristors vary from their nominal values. Assume the conductance values of all the memristors are expressed as $\mathbf{G}$. The actual conductance values considering process variations can be expressed as

$$\mathbf{G} = \mathbf{G}_0 + \mathbf{F}(\mathbf{G}_0, \mathbf{D}) \approx \mathbf{G}_0 + \mathbf{f}(\mathbf{G}_0) \cdot \mathbf{D} \tag{4.3}$$

where $\mathbf{G}_0$ represents the nominal conductance values. $\mathbf{F}(\mathbf{G}_0, \mathbf{D})$ is a function representing the relation between process variations and the change of conductance, which can be approximated by $\mathbf{f}(\mathbf{G}_0) \cdot \mathbf{D}$ due to the relatively small value of the process variations compared with the nominal values. $\mathbf{f}(\mathbf{G}_0)$ can be characterized from device measurement directly. Together with (4.2), (4.3) can be transformed into

$$\mathbf{G} = \mathbf{G}_0 + \mathbf{f}(\mathbf{G}_0) \cdot \mathbf{V} \cdot \mathbf{\Sigma}^{0.5} \cdot \mathbf{B}. \tag{4.4}$$

Besides process variations, programming noise also causes random drifting of the conductance values [CYL14]. To incorporate this effect, (4.4) can be modified as

$$\mathbf{G} = \mathbf{G}_0 + \mathbf{f}(\mathbf{G}_0) \cdot \mathbf{V} \cdot \mathbf{\Sigma}^{0.5} \cdot \mathbf{B} + \mathbf{h}(\mathbf{G_0})\mathbf{N} \tag{4.5}$$

where $\mathbf{h}(\mathbf{G}_0)$ represents the impact of noise on the conductance values of memristors and $\mathbf{N}$ contains independent random variables specific to certain memristors.

According to (4.5) and the relation between the weights and the conductance values in (2.16), the weights can be expressed as linear combinations of $\mathbf{B}$ and $\mathbf{N}$, since $\mathbf{G_0}$ and all the coefficients of $\mathbf{B}$ and $\mathbf{N}$ are known. Therefore, a weight under process variations and noise can be expressed in the canonical form [VRK$^+$06] as

$$w_i = w_{i,0} + \sum_{k=1}^{N} w_{i,k} B_k + w_{i,n} N_i \tag{4.6}$$

where $w_{i,0}$ is the nominal value of the weight, $w_{i,k}$ and $w_{i,n}$ are constant coefficients. B_k are random variables independent from each other but shared by all the weights. N_i is the pure random variable individual to each weight.

4.2 Statistical Forward Propagation

Since the weights have been expressed as linear combinations of random variables to incorporate process variations and noise, the training of neural networks should also be adapted, in which the multiplication, addition, activation functions as well as the cost function should be modified accordingly.

4.2.1 Statistical Multiplication

In a neural network such as shown in Fig. 2.5, the input to a neuron in the first layer is constant. This input needs to be multiplied with the weight on the connection to the next neuron. Since the weight is now expressed in the canonical form (4.6), this multiplication can be realized by multiplying the input constant to all the coefficients in (4.6). The result of this multiplication is still in the canonical form. At an output neuron in Fig. 2.5, or a neuron in the next hidden layer if it exists, the multiplication needs to be performed again. The input values from the previous layer to this neuron are the results of previous computation operations, so that they are already in the canonical form (4.6). Consequently, the multiplication should be performed between two expressions in the canonical form. Assume the input is denoted as a_j and the weight to multiply is denoted as w_i, The multiplication can be performed as

$$\begin{aligned}
a_k =& a_j \cdot w_i \\
=& (a_{j,0} + \sum_{k=1}^{N} a_{j,k}B_k + a_{j,n}N_j)(w_{i,0} + \sum_{k=1}^{N} w_{i,k}B_k + w_{i,n}N_i) \\
=& a_{j,0}w_{i,0} + \sum_{k=1}^{N}(a_{j,0}w_{i,k} + a_{j,k}w_{i,0})B_k + a_{j,0}w_{i,n}N_i + a_{j,n}w_{i,0}N_j \\
&+ \sum_{k=1}^{N}\sum_{l=1}^{N} a_{j,k}w_{i,l}B_kB_l + \sum_{k=1}^{N} a_{j,k}w_{i,n}B_kN_i + \sum_{k=1}^{N} a_{j,n}w_{i,k}N_jB_k + a_{j,n}w_{i,n}N_jN_i \\
\approx& a_{j,0}w_{i,0} + \sum_{k=1}^{N}(a_{j,0}w_{i,k} + a_{j,k}w_{i,0})B_k + \sqrt{(a_{j,0}w_{i,n})^2 + (a_{j,n}w_{i,0})^2}N_k.
\end{aligned} \tag{4.7}$$

The multiplication above produces terms of the second order and complicates the further propagation of the data across the neural network. To reduce computational complexity, we only keep the first-order terms and approximate $a_{j,0}w_{i,n}N_i + a_{j,n}w_{i,0}N_j$ using $\sqrt{(a_{j,0}w_{i,n})^2 + (a_{j,n}w_{i,0})^2}N_k$ by matching their variances [VRK+06], where N_k is a new independent random variable. Consequently, the result of a multiplication can also be represented in a canonical form and propagated further through the neural network using the same implementation of the computation operations.

4.2.2 Statistical Addition

At a neuron, the results of multiplication operations from different input neurons should be added together. Assume that the results of two multiplications are x_i and x_j, their sum can be computed as

$$x_k = x_i + x_j = (x_{i,0} + x_{j,0}) + \sum_{m=1}^{N}(x_{i,m} + a_{j,m})B_m + \sqrt{x_{i,n}^2 + x_{j,n}^2}N_k \tag{4.8}$$

where N_k is a new independent random whose coefficient is determined by matching variance with $x_{i,n}N_i + x_{j,n}N_j$.

4.2.3 Statistical Activation Function Transformation

At a neuron, the result of multiplication and addition with canonical form (4.6) which is also denoted as

$$x_i = x_{i,0} + \sum_{l=1}^{N} x_{i,k} B_k + x_{i,n} N_{x_i} = x_{i,0} + X_i \tag{4.9}$$

needs to be processed by an activation function f which can be expanded using Taylor series at the mean value $x_{i,0}$, and approximated by discarding terms with an order higher than 1, as

$$\begin{aligned} a_i = f(x_i) =& f(x_{i,0}) + f'(x_{i,0}) X_i + \sum_{l=2}^{\infty} \frac{f^{(l)}(x_{i,0}) X_i^l}{l!} \\ \approx& f(x_{i,0}) + f'(x_{i,0}) X_i. \end{aligned} \tag{4.10}$$

Softplus operation Inside a neural network, the active function used at neurons is often ReLU $y = max(x, 0)$. However, this function is nondifferentiable at 0, a condition that is needed to expand this function into Taylor series to allow the propagation of the canonical form (4.9) through the neural network. Therefore, we choose the softplus $f(x) = \log(1 + e^x)$, also called SmoothReLU, as the activation function.

According to (4.10), the softplus function can be approximated as

$$\begin{aligned} a_i = f(x_i) =& \log(1 + e^{x_{i,0}}) + \frac{e^{x_{i,0}}}{1 + e^{x_{i,0}}} X_i + \sum_{l=2}^{\infty} \frac{f^{(l)}(x_{i,0})}{l!} X_i^l \\ \approx& \log(1 + e^{x_{i,0}}) + \frac{e^{x_{i,0}}}{1 + e^{x_{i,0}}} X_i. \end{aligned} \tag{4.11}$$

Note that this linearization of the softplus function adapts itself according to the mean value of the input data to allow useful information to be propagated to the next layer. Fig. 4.1 (a), Fig. 4.1 (b) and Fig. 4.1 (c) present the examples of input distributions, output distributions after the actual activation softplus function and the linearized approximated softplus functions with different mean values. The examples show that the linearized approximated distributions are similar to the actual outputs so that the linearized approximation method is effective.

Sigmoid operation At the outputs of the neural network, sigmoid function $f(x) = 1/(1+e^{-x})$ may be applied to generate classification results. Similar to softplus operation, this function can be expanded into Taylor series and approximated as

$$\begin{aligned} a_i = f(x_i) =& \frac{1}{1+e^{-x_{i,0}}} + \frac{e^{-x_{i,0}}}{(1+e^{-x_{i,0}})^2} X_i + \sum_{l=2}^{\infty} \frac{f^{(l)}(x_{i,0})}{l!} X_i^l \\ \approx& \frac{1}{1+e^{-x_{i,0}}} + \frac{e^{-x_{i,0}}}{(1+e^{-x_{i,0}})^2} X_i. \end{aligned} \tag{4.12}$$

4.3 Statistical Probability Included Cost Function

As shown in (2.8), the cross entropy is used to guide the adjustment of the weights. If the expected value of an output is equal to 1, i.e., $\hat{a}_i = 1$, the actual value of this output a_i is expected to be close to 1. If a_i deviates much from 1, $\mathcal{C}$ in (2.8) quickly becomes large, so that the weights that contribute to this deviation are punished, as indicated by (2.10). This mechanism works similarly in the case when the output $\hat{a}_i$ is equal to 0.

When process variations and noise are considered, the actual value at an output is represented in the canonical form (4.6), so that the comparison between the actual value and the expected value, which is either 1 or 0 for classification, becomes statistical. To signify that the whole distribution of the output should be shifted toward the expected value, we use the mean value μ_{a_i} of the distribution to replace a_i in the original cost function (2.8). In addition, we punish an output if its distribution is not close to the expected value. If the expected value $\hat{a}_i$ is 1 but a_i incorrectly drifts to 0, the probability $P(a_i \leq 0.5)$ becomes large. Therefore, we also include

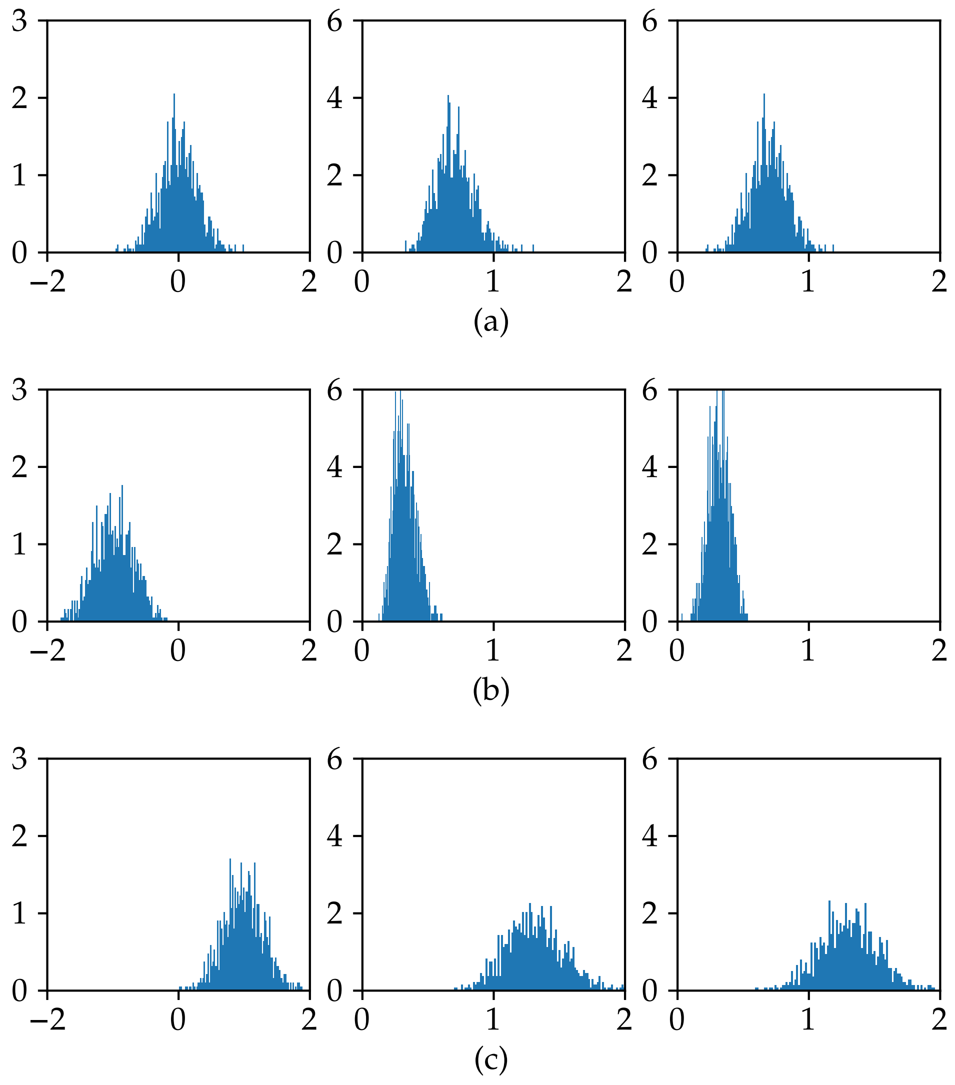


Figure 4.1: Distributions comparisons, in which the first figure is the input distribution, the second is the actual distribution after Softplus function, the third is the distribution after the linearized approximated function. (a) The input nominal value is 0. (b) The input nominal value is a negative value. (c) The input nominal value is a positive value.

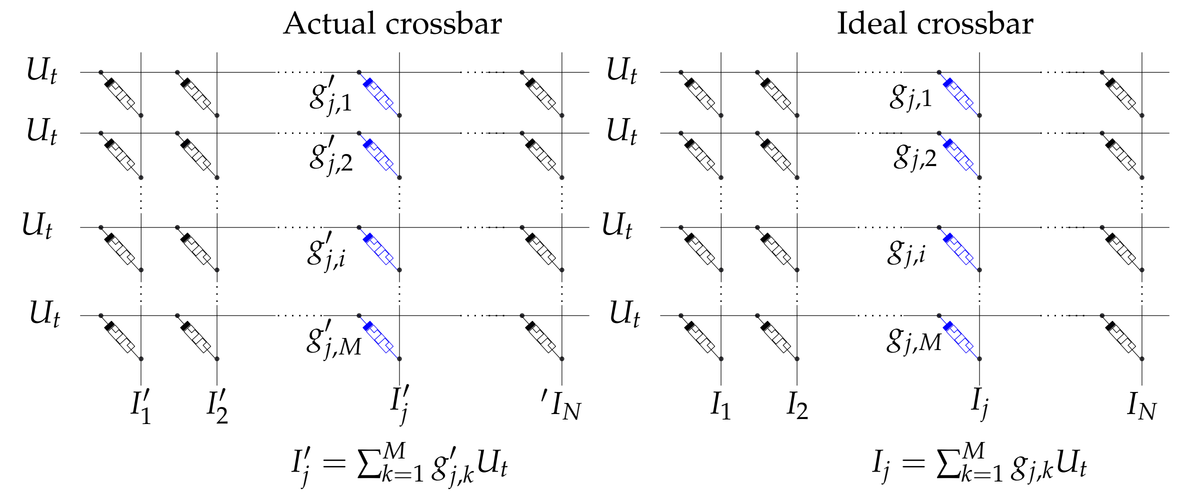


Figure 4.2: Compare the outputs between the crossbar with variations and the ideal crossbar to test and compensate for the global variation.

this probability into the cost function. Similarly, the case that the expected value equal to 0 is also adapted. Consequently, the cost function are transformed as

$$\mathcal{C} = \sum_{i=1}^{M} (-\hat{a}_i P^p(a_i \leq 0.5) \log(\mu_{a_i}) - (1-\hat{a}_i) P^p(a_i \geq 0.5) \log(1-\mu_{a_i})) \tag{4.13}$$

where p is a power used to magnify the influence of the probability.

4.4 Compensation for Global Variation

As explained in Section 4.1, global variation is shared by all the memristors, so that this variation contributes a large part of the correlation between the process variations. Consequently, the variations in the conductance values of the memristors also exhibit a tendency to vary into the same direction. To capture this tendency, memristors in a column in the crossbar as shown in Fig. 4.2 can be tested by applying a test voltage U_t to all the wordlines. The current measured at the output of the jth column can be written as

$$I'_j = (g'_{1,j} + g'_{2,j} + ... + g'_{M,j}) \cdot U_t \tag{4.14}$$

where $g'_{1,j}, g'_{2,j}, ..., g'_{M,j}$ are the actual conductance values of the memristors in the jth column. Due to process variations and noise during programming, these conductance values deviate from the values determined by weights after training. However, since the random variations experienced by the memristors can cancel each other out when the conductance values are added together, this sum can be used to evaluate the overall effect of global variation.

Since the current with the ideal conductances for these memristors can be expressed as

$$I_j = (g_{1,j} + g_{2,j} + ... + g_{M,j}) \cdot U_t \tag{4.15}$$

the current deviation caused by global variation on the memristors in this column can be expressed as a ratio $R = I'_j / I_j$. When programming memristors, we then adjust the programming voltage to offset the conductance by $1/R$ to compensate global variation in advance to improve the inference accuracy.

4.5 Simulation Results

The proposed framework was implemented using TensorFlow [AAB+15] and tested with an Intel 3.6 GHz CPU and an NVIDIA GeForce GTX1080Ti graphics card. The standard deviations of the distribution of process variations and noise were set to 25% and 5% of the nominal values [AR+16, CYL14]. The covariance matrix was generated using the method in [XZH07] and the global variation was set to 60% of the total process variations.

We simulated 2000 chips using Monte Carlo simulation and tested them with one-layer and two-layer fully-connected neural networks. The results are shown in Table 4.1, where DT is the conventional training method with deterministic weights without considering process variations and noise, VT is the vortex training method [LLC+15] and ST is the method proposed in this paper. In this experiment,

Table 4.1: Accuracy of SNN with different training methods [ZZW+20] ©2020 IEEE.

	DT			VT			ST		
NN	Acc_0	$\mu(Acc)$	$\sigma(Acc)$	Acc_0	$\mu(Acc)$	$\sigma(Acc)$	Acc_0	$\mu(Acc)$	$\sigma(Acc)$
FC1	0.91	0.87	0.10	0.91	0.89	0.05	0.91	0.90	0.03
FC2	0.97	0.32	0.33	0.92	0.38	0.32	0.92	0.92	0.01

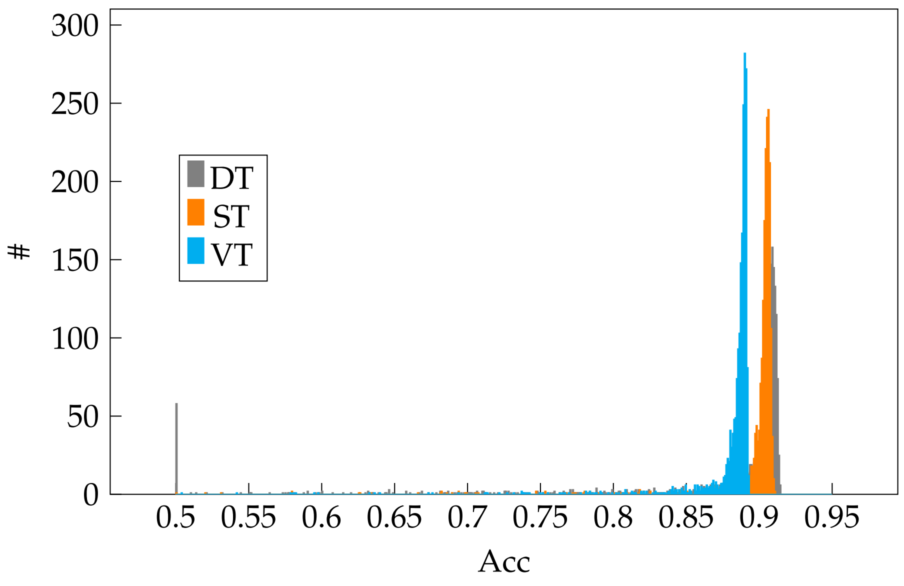


Figure 4.3: Accuracy distributions [ZZW+20] ©2020 IEEE.

the power p in (4.13) was set to 2. NN stands for neural network, FC1 and FC2 are one-layer and two-layer fully-connected neural networks, respectively. Acc_0 is the accuracy of the ideal case when memristors are programmed to conductance values equal to the target values mapped from weights, $\mu(Acc)$ and $\sigma(Acc)$ are the mean and standard deviation of the inference accuracy of the simulated 2000 chips, respectively.

As shown in Table 4.1, the proposed ST method has a larger mean value and a smaller standard deviation in the inference accuracy compared with VT and DT, indicating that the simulated chips can achieve a good accuracy with much fewer

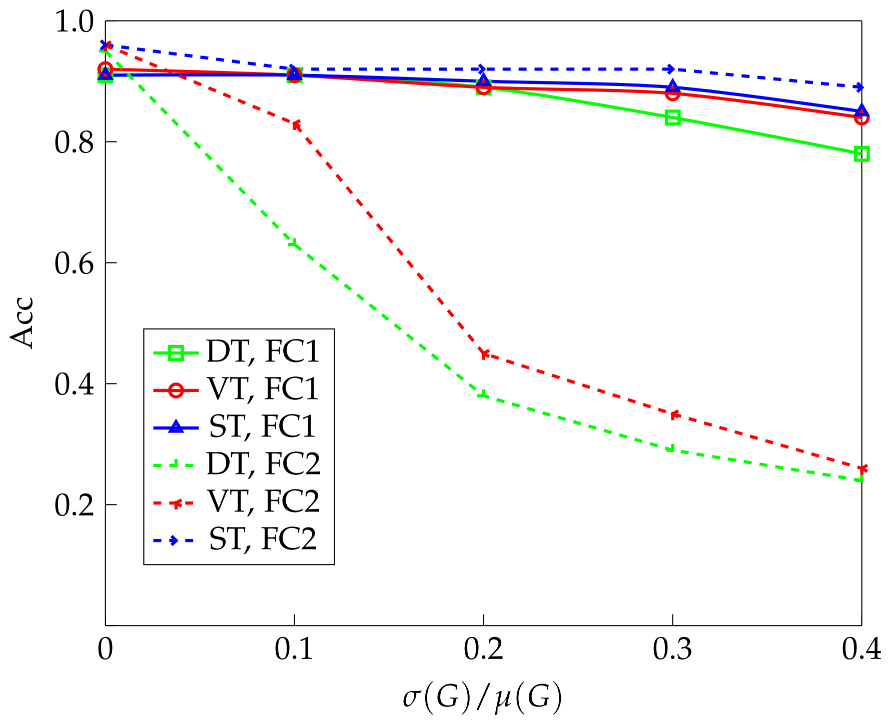


Figure 4.4: Robustness comparison [ZZW+20] ©2020 IEEE.

failing outliers. The distribution of the inference accuracy of the simulated chips tested with $FC1$ is shown in Fig. 4.3, where the results of ST have more chips concentrated toward high accuracy.

To verify the robustness of the proposed method, we tested different ratios of the standard deviation to the mean value of process variations, denoted as σ_G/μ_G, using FC1 and FC2. The results are illustrated in Fig. 4.4, where the y-axis shows the mean value of the inference accuracy of the simulated chips. According to this comparison, the proposed ST method maintains a stable accuracy as process variations increase, while the other methods suffer a significant accuracy loss.

4.6 Summary

In this chapter, a statistical training method for neural networks has been proposed. The inference accuracy can be maintained, even as process variations become large, by modeling process variations and noise as random variables and applying global variation compensation.

Chapter 5

Countering Variations and Thermal Effects for Accurate Optical Neural Networks

In this chapter, software training for ONNs is described firstly, in which phases of MZIs are trained directly to achieve a high inference accuracy. During training the phases of MZIs are also reduced, since the deviation of phases from the expected values due to variations is small when the MZIs are tuned to small phases, as shown in Fig. 5.3, where the resulting phases of MZIs at p_1, p_2 and p_3 spread from the nominal curve with different amounts. In addition, thermal imbalance is decreased to reduce phase drift of MZIs during training. To configure the phases of MZIs to the target values determined from training, we introduce a calibration procedure to determine the effects of process variations. Thermal effects among MZIs are also modeled and the power values to configure phases of MZIs are adjusted in advance to counter these effects. Finally, variation residues and noise are compensated with characterized online tuning. The flow of the proposed framework is illustrated in Fig. 5.1. [1]

[1]©2020 IEEE. Reprinted, with permission, from Y. Zhu and G. L. Zhang and B. Li and X. Yin and C. Zhuo and H. Gu and T. Ho and U. Schlichtmann, Countering Variations and Thermal Effects for Accurate Optical Neural Networks, 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 11/2020.

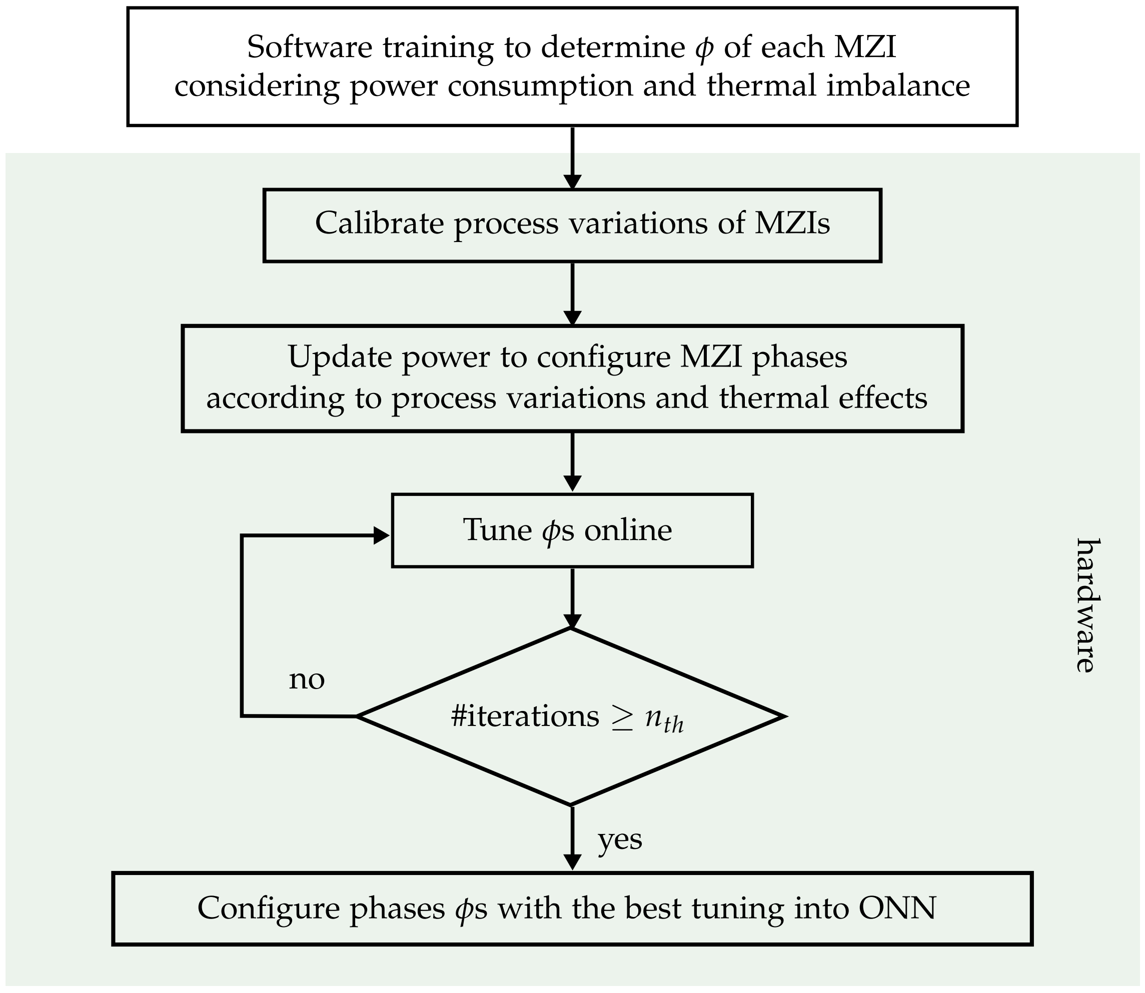

Software training to determine ϕ of each MZI considering power consumption and thermal imbalance
Calibrate process variations of MZIs
Update power to configure MZI phases according to process variations and thermal effects
Tune ϕs online
no
#iterations $\geq n_{th}$
yes
Configure phases ϕs with the best tuning into ONN
hardware

Figure 5.1: Work flow of the proposed framework [ZZL+20] ©2020 IEEE.

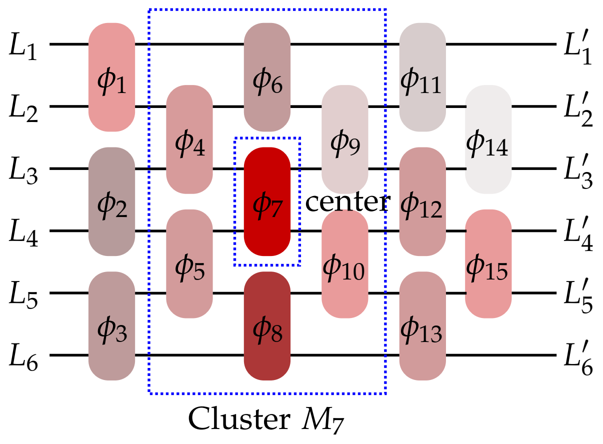


Figure 5.2: Thermal imbalance in an ONN. The darker the color is, the hotter the corresponding MZI is [ZZL+20] ©2020 IEEE.

5.1 Software Training for ONNs

In an ONN, the phases of MZIs carry computation information. Input data go through the ONN and are multiplied with the transformation matrix similar to (2.29) and (2.30). In the ONN, the items in the matrix are trigonometric functions of the phases of MZIs. During training, the phases of MZIs are updated with a cost function, defined as follows

$$Cost = -\sum_{i=1}^{\eta} \hat{y}_{(i)} log(y_{(i)}) = C(\boldsymbol{\phi}) \tag{5.1}$$

where $y_{(i)}$ is the value of the ith output of the neural network, and $\hat{y}_{(i)}$ is the target value of this output. η is the number of outputs. $C(\boldsymbol{\phi})$ is the cross entropy function with respect to the phases $\boldsymbol{\phi}$ of all the MZIs in the ONN. A lower cost usually results in a higher accuracy. To decrease the cost function, the phases of MZIs in the neural network are updated iteratively, as

$$\phi_i = \phi_i - \mathcal{LR} \cdot \frac{\partial C(\boldsymbol{\phi})}{\partial \phi_i} \tag{5.2}$$

where $\mathcal{LR}$ denotes the learning rate.

In an ONN, the phase of an MZI is tuned by changing its temperature. As shown in Fig. 5.3, the smaller the phase of an MZI is tuned, the smaller the deviation of this phase from the expected value becomes. Therefore, during training, we add a regularization term to the cost function to reduce the overall changed phase of MZIs, as

$$Cost = C(\boldsymbol{\phi}) + \alpha \cdot \sum_{i=1}^{N} \phi_i \tag{5.3}$$

where $C(\boldsymbol{\phi})$ is the cross entropy defined in (5.1), N is the number of MZIs in the whole ONN, and α is the penalty parameter to reduce the phase values of MZIs during training.

In an ONN, the MZIs have different temperatures after configuration. If MZIs with significant temperature difference are physically close to each other, their temperatures also drift due to thermal imbalance. For example, in Fig. 5.2, ϕ_7 and ϕ_8 have large phases and thus high temperatures, though other MZIs have small phases due to the appended regularization term in (5.3). These MZIs with small phases can be heated by ϕ_7 and ϕ_8 and their phases drift accordingly.

To address the problem above, we reduce thermal imbalance by narrowing the temperature gap between an MZI and its neighbors. In this technique, we choose each MZI in the ONN and its surrounding neighbors to form a cluster M_i, as illustrated in Fig. 5.2. The size of the cluster can be adjusted to include more MZIs according to different ONN structures. The average phase of the MZIs in such a cluster is computed as $\sum_{\phi_j \in M_i} \phi_j / |M_i|$, where M_i is the set of MZIs in the cluster and $|M_i|$ is the number of the MZIs in M_i. Afterwards, the difference between the phase of the MZI and this average is minimized to reduce thermal imbalance. Accordingly, the cost function in (5.3) is modified as

$$Cost = C(\boldsymbol{\phi}) + \alpha \cdot \sum_{i=1}^{N} \phi_i + \beta \cdot \sum_{i=1}^{N} |\phi_i - \sum_{\phi_j \in M_i} \phi_j / |M_i|| \tag{5.4}$$

where β is the penalty parameter to reduce thermal imbalance. After this training, the target phases of MZIs are determined and certain amount of power should be applied onto the MZIs to achieve these phases.

5.2 Hardware Implementation for ONNs

After the phases of MZIs are determined using software training, they should be configured into the thermal-optic phase shifters in the MZIs. In this section, we first introduce the proposed calibration procedure to extract process variations in Section 5.2.1. Afterwards, we compensate the phases of MZIs to counter thermal effects in Section 5.2.2. Finally, the residual variations and noise are compensated with a characterized online tuning in Section 5.2.3. With these steps, the effects of process variations and thermal impact can be reduced further.

5.2.1 Extraction of Process Variations with Calibration

To configure the MZI phases accurately, the effects of process variations on individual MZIs need to be extracted. However, MZIs in an ONN cannot be measured directly. Instead, only the final outputs of the ONN can be observed. To solve this problem, we propose to deploy a technique based on differential testing. The concept of this method is illustrated in Fig. 5.4, where the column C_4 is under test. This network contains four columns of MZIs. Their transformation matrices are written as $\mathbf{T}_{C_1}$, $\mathbf{T}_{C_2}$, $\mathbf{T}_{C_3}$, and $\mathbf{T}_{C_4}$.

In the test procedure, we first apply four input patterns 1000, 0100, 0010 and 0001 to the inputs of the ONN. These input patterns together form an identity matrix $\mathbf{I}$. The outputs corresponding to these inputs are written together as a matrix $\mathbf{M}$. As discussed in Section. 2.2.3, this output matrix is the multiplication of the transformation matrix of the ONN and the input data. Therefore, with the identity

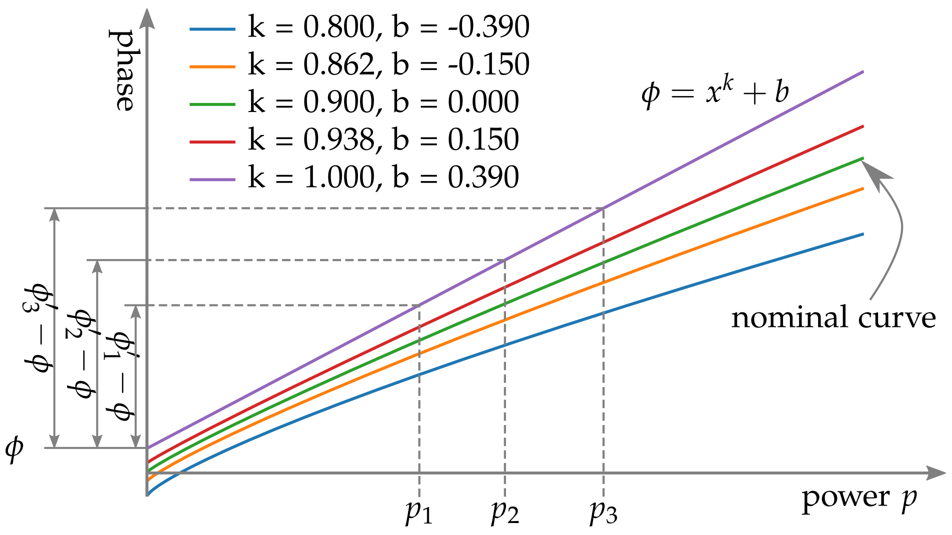


Figure 5.3: Characteristic curves of five MZIs under process variations, showing the relationship between the applied power and the corresponding phase change [ZZL+20] ©2020 IEEE.

matrix **I** as the input data, the matrix **M** is the same as the transformation matrix, so that $\mathbf{M} = \mathbf{T}_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1}$. Afterwards, we change the phases of all the MZIs in the column C_4 by applying the same amount of power. Due to process variations, the real phase changes of the MZIs in this column differ from each other. With the same input identity matrix **I**, the ONN produces another set of outputs $\mathbf{M}' = \mathbf{T}'_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1}$. Therefore, we can deduce

$$\begin{aligned}\mathbf{M}'\mathbf{M}^{-1} &= (\mathbf{T}'_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1})(\mathbf{T}_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1})^{-1}\\ &= \mathbf{T}'_{C_4}\mathbf{T}_{C_4}^{-1}\\ &= \mathbf{T}'_{C_4}\mathbf{T}_{C_4}^{*},\end{aligned} \tag{5.5}$$

where all matrices are unitary so that $\mathbf{T}_{C_4}^{-1} = \mathbf{T}_{C_4}^{*}$.

Similar to (2.31), all the transformation matrices of the columns are composed of submatrices in the form of (2.29). Therefore, the multiplication of the modified transformation matrix and the conjugate transpose of the original transformation matrix can always be processed by multiplying the individual transformation matrices of the MZIs. For example, the $\mathbf{T}'_{C_4}\mathbf{T}^*_{C_4}$ can be calculated as

$$\mathbf{T}'_{C_4}\mathbf{T}^*_{C_4} = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{T}'_6 & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{T}^*_6 & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{T}'_6\mathbf{T}^*_6 & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix} \tag{5.6}$$

where $\mathbf{T}'_6$ is the transformation matrix of the MZI corresponding to ϕ_6 in Fig. 5.4 after ϕ_6 is tuned to ϕ'_6 during calibration, and $\mathbf{T}^*_6$ is the conjugate transpose of $\mathbf{T}_6$.

Since both $\mathbf{T}'_6$ and $\mathbf{T}^*_6$ are in the form of (2.29), their multiplication can be expressed as

$$\begin{aligned} \mathbf{T}'_6\mathbf{T}^*_6 &= je^{\frac{j\phi'}{2}}(-je^{\frac{-j\phi}{2}}) \begin{bmatrix} \sin\frac{\phi'}{2} & \cos\frac{\phi'}{2} \\ \cos\frac{\phi'}{2} & -\sin\frac{\phi'}{2} \end{bmatrix} \begin{bmatrix} \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \\ \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \end{bmatrix} \\ &= e^{\frac{j(\phi'-\phi)}{2}} \begin{bmatrix} \cos\frac{\phi'-\phi}{2} & \sin\frac{\phi'-\phi}{2} \\ -\sin\frac{\phi'-\phi}{2} & \cos\frac{\phi'-\phi}{2} \end{bmatrix}. \end{aligned} \tag{5.7}$$

According to (5.5)–(5.7), $\mathbf{T}'_6\mathbf{T}^*_6$ is known since $\mathbf{M}'\mathbf{M}^{-1}$ can be obtained from test results. Therefore, $\tan\frac{\phi'-\phi}{2} = \sin\frac{\phi'-\phi}{2} / \cos\frac{\phi'-\phi}{2}$ becomes known by matching the corresponding items in the matrices. Consequently, $\Delta\phi = \phi' - \phi$ can be identified from this test procedure.

With the analysis above, we can observe that the phase change of an MZI with respect to a given power can be identified through test. However, this information is still insufficient to calibrate to which characteristic curve such as in Fig. 5.3 this MZI corresponds, since a curve in Fig. 5.3 is determined by two parameters k and b.

Therefore, for a column in an ONN such as in Fig. 5.4, we applied several different power values to tune the phases of MZIs in this column and obtain the corresponding phase changes. We then use these phase changes to match the characteristic curves as shown in Fig. 5.3, where three phase changes corresponding to the power values p_1, p_2 and p_3 are used.

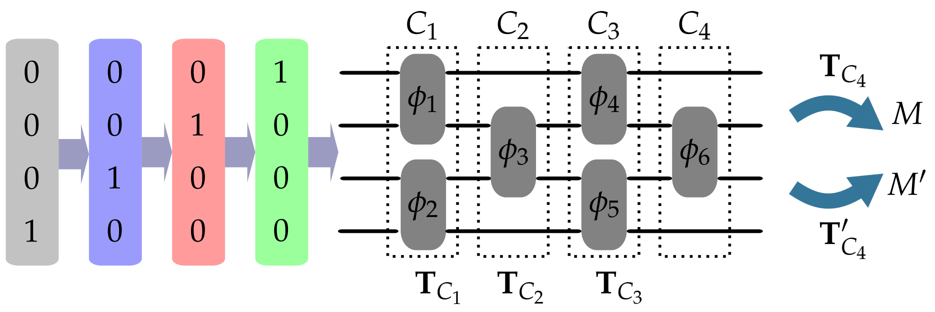


Figure 5.4: Differential testing to extract process variations of MZIs [ZZL+20] ©2020 IEEE.

Assume the three real phase changes of an MZI t corresponding to the power values p_1, p_2 and p_3 are obtained from test, as $\Delta\phi_1^t = \phi_1^t - \phi^t$, $\Delta\phi_2^t = \phi_2^t - \phi^t$, $\Delta\phi_3^t = \phi_3^t - \phi^t$, where ϕ^t is the original phase of the MZI t without any power applied. For the rth curve in Fig. 5.3, we can compute the phase changes $\Delta\phi_1^r$, $\Delta\phi_2^r$ and $\Delta\phi_3^r$ when the power values p_1, p_2 and p_3 are applied to this curve. These computed phase changes are then used to match the real phase changes $\Delta\phi_1^t$, $\Delta\phi_2^t$ and $\Delta\phi_3^t$ by comparing the difference defined as $\Delta_{t,r} = \sum_{k=1}^{M} |\Delta\phi_k^t - \Delta\phi_k^r|$, where M is the number of power values used during test and equal to 3 in this case. After all characteristic curves are enumerated, the curve i with the smallest $\Delta_{t,i}$ is selected as the characteristic curve of the MZI t.

With the characteristic curves of the MZIs in a column identified, the phases corresponding to any power values applied to these MZIs can be computed easily. Therefore, the transformation matrix of the column under test can be identified, such as $\mathbf{T}_{C_4}$ in (2.30). By multiplying the inverses of $\mathbf{T}_{C_4}$ to the left of $\mathbf{T}_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1}$, the result of $\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1}$ can be obtained. Therefore, the test procedure can be continued to identify the characteristic curves of the MZIs in the third column and thus $\mathbf{T}_{C_3}$. This procedure can be repeated on all the columns of MZIs to identify the characteristic curves of all MZIs in the ONN. The algorithm is summarized as Algorithm 1.

The characteristic curves calibrated above are specific to individual MZIs and are

Algorithm 1: Test the process variations.

```
1  foreach phase shifter t do
2      foreach power k do
3          calculate $\Delta\phi_k^t$ under power $k$ with (5.7);
4          foreach curve r do
5              get the reference $\Delta\phi_k^r$ under power $k$;
6              calculate $\Delta_{tr}^{[k]} = \Delta\phi_k^t - \Delta\phi_k^r$;
7          end
8      end
9      $\Delta_{t,r} = \sum_{k=1}^{M} \left|\Delta_{tr}^{[k]}\right|$;
10     Process variation of phase shifter $t$ fits the curve $\operatorname{argmin}_r \Delta_{t,r}$;
11 end
```

used to compute the required power p_k for the kth MZI according to the target phases determined by training, so that the effects of process variations are taken into account.

5.2.2 Power Adjustment to Counter Thermal-effected Inaccuracy

Even with thermal-aware training using (5.4) the target phases and thus the temperatures of some MZIs may still differ much instead of being balanced. Consequently, MZIs with different temperatures affect each other and cause their phases to deviate from the expected values. To counter this deviation, we adjust the power applied onto MZIs in advance with thermal compensation. Instead of the target power p_k to the kth MZI, the real power applied to this MZI is modified to p_k^r. Consequently, with the thermal effects of neighboring MZIs, the actual power the kth MZI receives matches p_k.

Assume that the kth MZI and the jth MZI are neighbors and the real power values applied to them are denoted as p_k^r and p_j^r. The thermal effect of the jth MZI to the kth MZI depends on their distance. Therefore, the target power p_k that is actually received by the *k*th MZI can be expressed as

$$p_k = \lambda_k p_k^r + \sum_{j \in N_k} \lambda_{j,k} p_j^r \tag{5.8}$$

where λ_k is the ratio of p_k^r received by the *k*th MZI directly. $\lambda_k < 1$ since a part of p_k^r is contributed to the neighboring MZIs, N_k contains the indexes of the MZIs neighboring the *k*th MZI, $\lambda_{j,k}$ is a constant determined by the distance between the *j*th MZI and the *k*th MZI and can be set as in [JSEF+19].

The target power p_k of the *k*th MZI is determined from training and variation calibration. With p_k determined, all the real power values of the MZIs can be calculated by solving the linear system of (5.8) for all MZIs. By applying the power p_k^r to the *k*th MZI, the power actually received by this MZI is p_k, so that the thermal effects between MZIs can be compensated.

5.2.3 Characterized Tuning of ONNs

After process variations are calibrated and thermal effects are compensated by adjusting power values in advance, statistical noise from unpredictable residual effects may still exist. Even though these effects are relatively small, they may still cause a remarkable degradation of inference accuracy as demonstrated in Fig. 2.20.

To improve the inference accuracy of ONNs, we tune the phases after MZIs are configured. Unlike in other online tuning methods such as [HLC+13], the result of a tuning iteration in an ONN is not observable, since the MZIs cannot be measured directly. To address this challenge, we characterize a given number of tuning directions by emulating the statistical noise during training and use them to tune ONNs online.

After training, the phases of the MZIs are determined. We then generate a given number of samples according to noise distributions and inject them into the phases.

The gradients with the cost function in (5.4) are updated with respect to all these random samples and stored for online tuning later.

During online tuning, we first enumerate all the precharacterized gradient samples stored during training. In each gradient sample, we apply a small amount of power change to each individual MZI, and the direction of this tuning is determined by the gradient sample. After all the samples are enumerated, the gradient sample with the highest inference accuracy is selected and the corresponding tuning is applied. Online tuning repeats until a given number of iterations, n_{th} in Fig. 5.1, are executed, and the best tuning result in all the iterations is adopted.

5.3 Experimental Results

To evaluate the proposed framework for ONNs, three different neural networks, a 2-layer fully-connected neural network (FCNN), LeNet-5 with 2 convolutional layers and 3 fully-connected layers, and an augmented LeNet-5 with 4 convolutional layers and 3 fully-connected layers were implemented using ONNs. These networks were then used to process two image datasets, MNIST and Cifar10. To maintain a high inference accuracy, we duplicated each convolutional layer for LeNet-5 and the augmented LeNet-5 to construct the corresponding ONNs. The proposed method, however, is independent of the ONN architectures and can also be applied to the architecture based on SVD decomposition [YS+17] or the sparse tree structure [ZLL+19]. The neural networks and the proposed algorithm were implemented using TensorFlow [AAB+15] and tested with an Intel 3.4 GHz CPU and an Nvidia Quadro RTX 6000 graphics card.

To emulate process variations, we generated 100 sample chips for each ONN with process variations and noise modeled as Gaussian distributions. In each sample, the parameters of all MZIs were generated randomly according to the distributions of process variations. We used 1000 characteristic curves similar to Fig. 5.3 to

Table 5.1: Significant drop of inference accuracy without variation extraction and power adjustment [ZZL$^+$20] ©2020 IEEE.

		w/o var. ext.		w/o pow. eval.	
NN	Dataset	μ	σ	μ	σ
FCNN	MNIST	11.63%	1.47%	12.07%	1.79%
LeNet-5	Cifar10	11.22%	0.86%	11.05%	0.82%
Aug.LeNet-5	Cifar10	13.76%	0.82%	14.35%	0.93%

Table 5.2: Results of ONNs with the proposed framework [ZZL$^+$20] ©2020 IEEE.

		Acc. Software Training			Acc. ONN_MZI		Power
NN	Dataset	NN	ONN	ONN$_p$	μ	σ	Reduct.
FCNN	MNIST	97.40%	97.13%	94.41%	92.80%	0.6%	25.83%
LeNet-5	Cifar10	76.56%	75.91%	74.60%	74.11%	0.26%	20.24%
Aug.LeNet-5	Cifar10	82.81%	82.08%	81.23%	80.80%	0.22%	13.27%

capture the process variations. In these curves, the deviation of phases from the nominal curve increases as the applied power increases, which corresponds to large phases in the MZIs, and the largest deviation appears at the phase 2π. In the experiments, the 3σ of the phases at 2π was set to 20% to determine the parameters of k and b in the characteristic curves in Fig. 5.3. The noise is purely random and its 3σ was set to 1.9%.

To demonstrate the necessity and effectiveness of the proposed framework countering variations and thermal effects, we tested the baseline cases by disabling the extraction of process variations described in Section 5.2.1 and the countermeasure for thermal effects described in Section 5.2.2, respectively, and the results are shown in Table 5.1. The third and fourth columns in Table 5.1 show the mean value and the standard deviation of the inference accuracy without extracting process variations, respectively. The fifth and the sixth columns in Table 5.1 list the mean value and standard deviation of the inference accuracy without the countermeasure of thermal effects, respectively. According to this table, the inference accuracy of ONNs drops drastically down to an unusable level if either of the proposed techniques were not applied, confirming the indispensability of the proposed framework.

The results of applying the proposed framework of variation extraction and power adjustment are shown in Table 5.2. The third column in Table 5.2 shows the inference accuracy of the original neural networks without using ONNs as accelerator. These values represent the capability of these general networks in processing the corresponding datasets. The inference accuracy of ONN implementation using MZIs with the cost function (5.1) is shown in the fourth column. In these tests, process variations and thermal effects on MZIs have not been considered. Since these values are very close to those in the third column of Table 5.2, it has been demonstrated that ONNs with MZIs are capable of maintaining a similar inference accuracy in realizing neural networks while providing the advantage of high bandwidth.

To counter thermal imbalance, we penalized phases during software training using the cost function (5.4). The resulting accuracy is shown in the fifth column in Table 5.2. These accuracy values degrade further but are still comparable to the cases without penalization during software training. In countering thermal imbalance using (5.4), a side effect is that the phases of the MZIs were generally reduced, leading to a smaller power consumption. The results of this reduction using (5.4) over the case without this penalization are shown in the last column in Table 5.2.

After configuring the phases into MZIs, process variations, thermal effects and noise came into play, so that the accuracy values of the 100 sampled chips also become a distribution. The mean value μ and the standard deviation σ of the inference accuracy are shown in the sixth column and the seventh column in Table 5.2, respectively. With the introduced calibration of process variations and compensation of thermal effects, the inference accuracy only decreases slightly compared with software training and the distributions of the accuracy values are within small

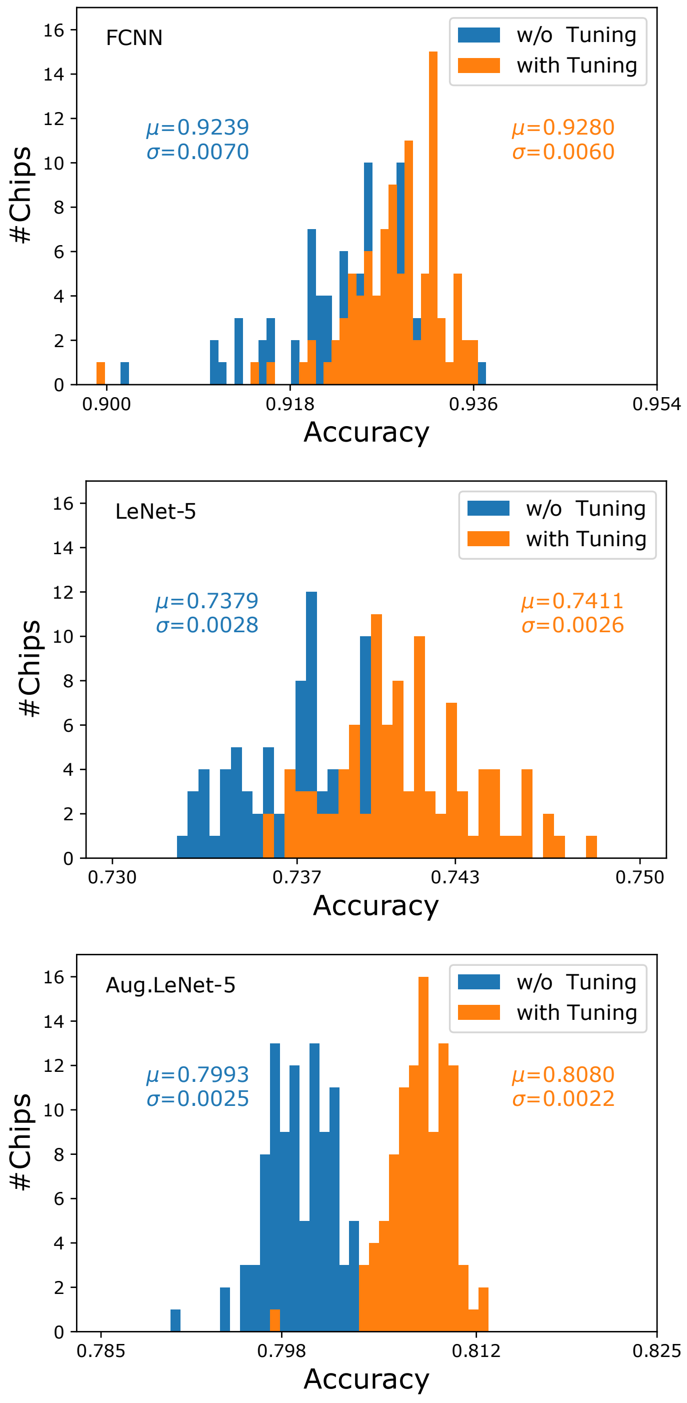


Figure 5.5: Accuracy distribution with/without characterized tuning for FCNN, LeNet-5 and Augmented LeNet-5 [ZZL+20] ©2020 IEEE.

ranges, leading to a good quality control.

To compensate residual noise, we tuned MZI phases online according to a precharacterized set of gradients. To verify the effectiveness of online tuning, this step was disabled in the proposed framework and the distributions of the inference accuracy of 100 chips are compared in Fig. 5.5. According to this comparison, it is clear that online tuning is able to improve the inference accuracy further.

5.4 Conclusions

In this chapter, we have proposed a framework for ONNs to counter their sensitivity to deviation of phases due to process variations and thermal effects. During software training, phases are penalized to reduce process variations and thermal imbalance. To configure phases of MZIs accurately on hardware implementation of ONNs, we extract process variations of MZIs and adjust power applied on them in advance. Residual noise is compensated by online tuning to improve inference accuracy further. Experimental results confirm that the proposed framework improves inference accuracy effectively.

Chapter 6

Conclusion

In the past decade, engineers have been exploring and developing emerging systems for biochemical labs and neuromorphic computing. Biochips and CIM systems have been proposed (see Chapter 2), as they can automatically carry out biochemical experiments and integrate storage and computing, respectively. To further improve the efficiency of the emerging systems, the control logic for biochips (see Chapter 3), memristor-based crossbars (Chapter 4) and MZI-arrays (Chapter 5) for neural network computing are optimized regarding power consumption and impacts from manufacturing faults, process variations, thermal issues, and so on.

To improve the efficiency and fault tolerance for biochips, the multi-channel multiplexing control logic accompanied by the backup path structure is proposed, with an ILP model established to design the control logic automatically. The novel control logic can provide an average 61.33% improvement in time cost and an average 45.27% saving of control valves compared to the direct connection structure. Fault tolerance is also improved in the proposed framework.

To reduce the impacts of process variations and noise in the memristor-based crossbars, a framework is proposed consisting of a statistical training method for neural networks by modeling process variations and noise as random variables and of a global variation compensation. The interference accuracy of neural networks implemented in hardware with process variations shows significant improvement. Future work is likely to include exploring statistical training on more complex multi-layer neural networks and convolutional neural networks. Considering the limited first-order statistical information propagation and the self-designed modified cost function in the existed framework, exploration directions may include how to propagate the high-order statistical information efficiently and how to design the cost function able to deal with output distributions with high-order statistical information. But most importantly, whether there exist variation-tolerant weights is needed to figure out, which may require more exploration in the mathematics of neural networks.

In the neuromorphic computing based on the MZI array, traditional neural networks with weights of general real-value matrices are modified to ONNs with unitary complex-value matrices. In ONN training, the cost function is modified to a cross entropy along with regulation terms of phase summation and cluster power difference from thermo-optical phase shifters to reduce power consumption, process variations, and thermal imbalance in the MZI array. A process variation testing algorithm and a power adjustment method are proposed to counter process variations and thermal effects, respectively, followed by a characterized tuning method to compensate for the residual noise. Simulation results show a potential 25.83% power saving compared to the ONN without power optimization, and the framework maintains an ONN inference accuracy well-matched to its electricity counterparts. Future work should include optimization and simplification of ONN architectures, efficiency improvements in process variation calibration algorithms and power adjustment methods, designing a tuning algorithm that works for larger residual noise, as well as applying online-training to reduce the sensitivity of ONNs to variations and thermal effects directly. The inspirations to design and optimize these algorithms could be from the classical algorithms or via machine learning.

Appendix A

Complex Expressions for Optical Wave

In the optical domain, if we use the trigonometric identities to operate the light transformation in mathematics, it will be quite difficult. Fortunately, another method exists. The oscillatory trigonometric function can be expressed as an exponent of the natural number e by Euler's formulas

$$e^{j\theta} = \cos\theta + j\sin\theta, \tag{A.1}$$

where j is the fundamental imaginary number

$$j = \sqrt{-1}. \tag{A.2}$$

The real and imaginary parts of $e^{j\theta}$ can be easily extracted by

$$\Re(e^{j\theta}) = \cos\theta \tag{A.3}$$

$$\Im(e^{j\theta}) = \sin\theta. \tag{A.4}$$

Therefore, the light signal can be written in its complex form

$$L^c = Ae^{j(\omega t + kz + \theta)} \tag{A.5}$$

with

$$\Re(L^c) = A\cos(\omega t + kz + \theta) \tag{A.6}$$

$$\Im(L^c) = A\sin(\omega t + kz + \theta). \tag{A.7}$$

In reality, $\Re(L^c)$ is the light signal. The amplitude A and phase P of the wave can be obtained by

$$A(L^c) = \sqrt{\Re^2(L^c) + \Im^2(L^c)}, \tag{A.8}$$

$$P(L^c) = \arctan\frac{\Im(L^c)}{\Re(L^c)} = \omega t + kz + \theta. \tag{A.9}$$

With the exponential expressions, it is easy to perform mathematical functions on the light waves including wave adding, wave multiplication, amplitude and phase obtaining [GGG17]. Take the operation $L_1 + L_2$ as the example. The generated wave can be easily obtained with the following steps:

1. Change the light signals from trigonometric to complex form:

$$L_1^c = A_1 e^{j(\omega t + kz + \theta_1)} \tag{A.10}$$

$$L_2^c = A_2 e^{j(\omega t + kz + \theta_2)}. \tag{A.11}$$

2. The two signals can be added directly in the complex forms to obtain the superposed wave, namely, adding real parts and imaginary parts, respectively as

$$L_{12}^c = \Re(L_{12}^c) + j\Im(L_{12}^c) = L_1^c + L_2^c = \Re(L_1^c) + \Re(L_2^c) + j(\Im(L_1^c) + \Im(L_2^c)) \tag{A.12}$$

3. According to (A.8) and (A.9), the amplitude and phase of the superposed wave can be obtained by

$$A(L_{12}^c) = \sqrt{(\Re(L_1^c) + \Re(L_2^c))^2 + (\Im(L_1^c) + \Im(L_2^c))^2} \tag{A.13}$$

$$P(L_{12}^c) = \arctan\frac{(\Im(L_1^c) + \Im(L_2^c))}{(\Re(L_1^c) + \Re(L_2^c))} \tag{A.14}$$

4. The real superposed wave can be extracted from the complex form by

$$\begin{aligned} L_{12} &= \Re(L_{12}^c) = A(L_{12}^c)\cos\left(P(L_{12}^c)\right) \\ &= \sqrt{(\Re(L_1^c) + \Re(L_2^c))^2 + (\Im(L_1^c) + \Im(L_2^c))^2}\cos(\arctan\frac{\Im(L_1^c) + \Im(L_2^c)}{\Re(L_1^c) + \Re(L_2^c)}). \end{aligned} \tag{A.15}$$

Other light signal operations can also be processed similarly in the complex domain.

Bibliography

[AAB$^+$15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AQ12] I. E. Araci and S. R. Quake. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab Chip*, 12:2803–2806, 2012.

[AR$^+$16] E. Amat, A. Rubio, et al. Memristive crossbar memory lifetime evaluation and reconfiguration strategies. *IEEE Trans. Emerg. Topics in Comput.*, 6(2):207–218, 2016.

[BBA07] R. Barak and Y. Ben-Aryeh. Quantum fast fourier transform and quantum computation by linear optics. *J. Opt. Soc. Am. B*, 24(2):231–240, 2007.

[BDHVV$^+$12] W. Bogaerts, P. De Heyn, T. Van Vaerenbergh, K. De Vos, S. Kumar Selvaraja, T. Claes, P. Dumon, P. Bienstman, D. Van Thourhout, and R. Baets. Silicon microring resonators. *Laser & Photonics Reviews*, 6(1):47–73, 2012.

[BDRDR03] R. E. Bryant, O. David Richard, and O. David Richard. *Computer systems: a programmer's perspective*, volume 2. Prentice Hall Upper Saddle River, 2003.

[BM$^+$13] M. Brunner, D.and Soriano, C. Mirasso, et al. Parallel photonic information processing at gigabyte per second data rates using transient states. *Nature Communication*, 4(1364):1–7, 2013.

[BMF$^+$18] J. Bueno, S. Maktoobi, L. Froehly, I. Fischer, M. Jacquot, L. Larger, and D. Brunner. Reinforcement learning in a large-scale photonic recurrent neural network. *Optica*, 5:756–760, 2018.

[Bro08] J. G. Brookshear. *Computer science: an overview*. Addison-Wesley Publishing Company, 2008.

[BV04] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CHG$^+$19] Z. Chen, X. Huang, W. Guo, B. Li, T.-Y. Ho, and U. Schlichtmann. Physical synthesis of flow-based microfluidic biochips considering distributed channel storage. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1525–1530, 2019.

[CHM$^+$16] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley. Optimal design for universal multiport interferometers. *Optica*, 3(12):1460–1465, 2016.

[Chu71] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[Chu11] L. Chua. Resistance switching memories are memristors. *Applied Physics A*, 102(4):765–783, 2011.

[CLC$^+$17] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, 2017.

[CMP13] A. Calimera, E. Macii, and M. Poncino. The human brain project and neuromorphic computing. *Functional neurology*, 28(3):191, 2013.

[COT00] M. Conti, S. Orcioni, and C. Turchetti. Training neural networks to be insensitive to weight random variations. *Neural networks*, 13(1):125–132, 2000.

[CYL14] S. Choi, Y. Yang, and W. Lu. Random telegraph noise and resistance switching analysis of oxide based resistive memory. *Nanoscale*, 6(1):400–404, 2014.

[DHMM96] C. Diorio, P. Hasler, A. Minch, and C. A. Mead. A single-transistor silicon synapse. *IEEE Transactions on Electron Devices*, 43(11):1972–1980, 1996.

[Dub17] K. Dubovikov. Pytorch vs tensorflow – spotting the difference. *Towards Data Science abs/1703.06907*, 2017.

[EiSWd13] K. S. Elvira, X. C. i Solvas, R. C. R. Wootton, and A. J. deMello. The past, present and potential for microfluidic reactor technology in chemical synthesis. *Nature Chemistry*, 5:905–915, 2013.

[Eta19] L. Etaati. Deep learning tools with cognitive toolkit (cntk). In *Machine Learning with Microsoft Technologies*, pages 287–302. Springer, 2019.

[Fai07] R. B. Fair. Digital microfluidics: is a true lab-on-a-chip possible? *Microfluidics and Nanofluidics*, 3(3):245–281, 2007.

[FM11] L. M. Fidalgo and S. J. Maerkl. A software-programmable microfluidic device for automated biology. *Lab Chip*, 11:1612–1619, 2011.

[FMW$^+$19] M. Y.-S. Fang, S. Manipatruni, C. Wierzynski, A. Khosrowshahi, and M. R. DeWeese. Design of optical neural networks with component imprecisions. *Optics express*, 27(10):14009–14029, 2019.

[FYW$^+$19] J. Feldmann, N. Youngblood, C. D. Wright, H. Bhaskaran, and W. Pernice. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, 569(7755):208–214, 2019.

[GGG17] G. D. Gillen, K. Gillen, and S. Guha. *Light propagation in linear optical media*. CRC Press, 2017.

[GP17] A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[GWY$^+$17] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille. Close-to-optimal placement and routing for continuous-flow microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 530–535, 2017.

[GZF$^+$20] J. Gu, Z. Zhao, C. Feng, M. Liu, R. T. Chen, and D. Z. Pan. Towards area-efficient optical neural networks: An FFT-based architecture. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 476–481, 2020.

[GZY$^+$19] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. [dl] a survey of fpga-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(1):1–26, 2019.

[Har14] Y. N. Harari. *Sapiens: A brief history of humankind*. Random House, 2014.

[HBM$^+$12] B. Hadwen, G. Broder, D. Morganti, A. Jacobs, C. Brown, J. Hector, Y. Kubota, and H. Morgan. Programmable large area digital microfluidic array with integrated droplet sensing for bioassays. *Lab on a Chip*, 12(18):3305–3313, 2012.

[HCH17] K. Hu, K. Chakrabarty, and T.-Y. Ho. *Computer-Aided Design of Microfluidic Very Large Scale Integration (mVLSI) Biochips*. Springer, 2017.

[HDHC17] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty. Control-layer routing and control-pin minimization for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 36(1):55–68, 2017.

[HGL$^+$18] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, et al. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018.

[HGR$^+$17] W.-L. Huang, A. Gupta, S. Roy, T.-Y. Ho, and P. Pop. Fast architecture-level synthesis of fault-tolerant flow-based microfluidic biochips. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1671–1676, 2017.

[HHCG19] X. Huang, T.-Y. Ho, K. Chakrabarty, and W. Guo. Timing-driven flow-channel network construction for continuous-flow microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2019. doi:10.1109/TCAD.2019.2912936.

[HHG$^+$19] X. Huang, T.-Y. Ho, W. Guo, B. Li, and U. Schlichtmann. Minicontrol: Synthesis of continuous-flow microfluidics with strictly constrained control ports. In *Proc. Design Autom. Conf. (DAC)*, pages 145:1–6, 2019.

[HLC$^+$13] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose. BSB training scheme implementation on memristor-based circuit. In *IEEE Symp. on Comput. Intelligence for Security and Defense Applications*, pages 80–87, 2013.

[HMM$^+$14] N. C. Harris, Y. Ma, J. Mower, et al. Efficient, compact and low loss thermo-optic phase shifter in silicon. *Opt. Express*, 22(9):10487–10493, 2014.

[HSL$^+$16] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[HYHC14] K. Hu, F. Yu, T.-Y. Ho, and K. Chakrabarty. Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 33(10):1463–1475, 2014.

[HZRS16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IL73] A. G. Ivakhnenko and V. G. Lapa. *Cybernetic predicting devices.* CCM Information Corporation, 1973.

[Jou16] N. Jouppi. Google supercharges machine learning tasks with tpu custom chip. *Google Blog, May*, 18:1, 2016.

[JRRR17] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan. Computing in memory with spin-transfer torque magnetic ram. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):470–483, 2017.

[JSD$^+$14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[JSEF$^+$19] M. Jacques, A. Samani, E. El-Fiky, D. Patel, Z. Xing, and D. V. Plant. Optimization of thermo-optic phase-shifter design and mitigation of thermal crosstalk on the SOI platform. *Opt. Express*, 27:10456–10471, 2019.

[KGPS18] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag. A multi-functional in-memory inference processor using a standard 6t sram array. *IEEE Journal of Solid-State Circuits*, 53(2):642–655, 2018.

[KNH14] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSJ$^+$00] E. R. Kandel, J. H. Schwartz, T. M. Jessell, D. of Biochemistry, M. B. T. Jessell, S. Siegelbaum, and A. Hudspeth. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[KSYC10] H. Kim, M. P. Sah, C. Yang, and L. O. Chua. Memristor-based multilevel memory. In *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, pages 1–6. IEEE, 2010.

[KTR$^+$08] I. Kuon, R. Tessier, J. Rose, et al. Fpga architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[LCB10] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. 2010.

[LCH13] Y. Luo, K. Chakrabarty, and T.-Y. Ho. Design of cyberphysical digital microfluidic biochips under completion-time uncertainties in fluidic operations. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–7. IEEE, 2013.

[LLB$^+$17] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T.-Y. Ho, and U. Schlichtmann. Testing microfluidic fully programmable valve arrays (FPVAs). In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 91–96, 2017.

[LLC$^+$14] C.-X. Lin, C.-H. Liu, I.-C. Chen, D. T. Lee, and T.-Y. Ho. An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips. In *Proc. Design Autom. Conf. (DAC)*, pages 141:1–141:6, 2014.

[LLC$^+$15] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang. Vortex: Variation-aware training for memristor x-bar. In *Proc. Design Autom. Conf. (DAC)*, 2015.

[LLY$^+$17a] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann. Transport or store?: Synthesizing flow-based microfluidic biochips using distributed channel storage. In *Proc. Design Autom. Conf. (DAC)*, pages 49:1–49:6, 2017.

[LLY$^+$17b] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann. Transport or store? synthesizing flow-based microfluidic biochips using distributed channel storage. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[LSE$^+$05] C. Lee, G. Sui, A. M. Elizarov, C. J. Shu, Y. S. Shin, A. N. Dooley, J. Huang, A. Daridon, P. G. Wyatt, D. B. Stout, et al. Multistep synthesis of a radiolabeled imaging probe using integrated microfluidics. *Science*, 310(5755):1793–1796, 2005.

[LTA16] G. Lacey, G. W. Taylor, and S. Areibi. Deep learning on fpgas: Past, present, and future. *arXiv preprint arXiv:1602.04283*, 2016.

[LTL$^+$16] M. Li, T. Tseng, B. Li, T. Ho, and U. Schlichtmann. Sieve-valve-aware synthesis of flow-based microfluidic biochips considering specific biological execution limitations. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 624–629, 2016.

[LV$^+$12] D. M.-M. Laurent Vivien, Andreas Polzer et al. Zero-bias 40gbit/s germanium waveguide photodetector on silicon. *Opt. Express*, 20:1096–1101, 2012.

[LWW$^+$14] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang. Training itself: Mixed-signal training acceleration for memristor-based neural network. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, 2014.

[LYY$^+$15] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Q. Wu, and H. Jiang. A spiking neuromorphic design with resistive crossbar. In *Proc. Design Autom. Conf. (DAC)*, 2015.

[MAQ06] J. S. Marcus, W. F. Anderson, and S. R. Quake. Microfluidic single-cell mRNA isolation and analysis. *Analytical Chemistry*, 78(9):3084–3089, 2006.

[MBGK$^+$17] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov. High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays. *IEEE transactions on neural networks and learning systems*, 29(10):4782–4790, 2017.

[McK04] S. A. McKee. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*, page 162, 2004.

[Mit19] S. Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *Machine learning and knowledge extraction*, 1(1):75–114, 2019.

[MKB$^+$10] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[MQ07] J. Melin and S. Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annu. Rev. Biophys. Biomol. Struct.*, 36:213–231, 2007.

[NCXX10] D. Niu, Y. Chen, C. Xu, and Y. Xie. Impact of process variations on emerging memristor. In *Proceedings of the 47th Design Automation Conference*, pages 877–882, 2010.

[PGM$^+$19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[RB17] A. Ribeiro and W. Bogaerts. Digitally controlled multiplexed silicon photonics phase shifter using heaters with integrated diodes. *Optics express*, 25(24):29778–29787, 2017.

[RKF$^+$09] D. Rosenbluth, K. Kravtsov, M. P. Fok, , and P. R. Prucnal. A high performance photonic pulse processing device. *Opt. Express*, 17:22767–22772, 2009.

[RZBB94a] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani. Experimental realization of any discrete unitary operator. *Physical review letters*, 73(1):58, 1994.

[RZBB94b] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani. Experimental realization of any discrete unitary operator. *Phys. Rev. Lett.*, 73(1):58–61, 1994.

[Sch15] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[SDCG$^+$15] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky. Memristor-based multilayer neural networks with online gradient descent training. *IEEE Trans. Neural Netw. Learn. Syst.*, 26(10):2408–2421, 2015.

[Seb09] G. A. Seber. *Multivariate observations*, volume 252. John Wiley & Sons, 2009.

[SKK10] S. Shin, K. Kim, and S.-M. Kang. Memristor applications for programmable analog ics. *IEEE Transactions on Nanotechnology*, 10(2):266–274, 2010.

[SL12] A. W. Snyder and J. Love. *Optical waveguide theory*. Springer Science & Business Media, 2012.

[SLGB$^+$18] A. Sebastian, M. Le Gallo, G. W. Burr, S. Kim, M. BrightSky, and E. Eleftheriou. Tutorial: Brain-inspired computing using phase-change memory devices. *Journal of Applied Physics*, 124(11):111101, 2018. Available at https://doi.org/10.1063/1.5042413.

[SSEM18] A. Shawahna, S. M. Sait, and A. El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2018.

[SSSW08] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[Ste99] J. Stewart. Calculus, brooks. *Cole Publishing Company*, 6:378, 1999.

[SW13] R. Stanley Williams. How we found the missing memristor. In *Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM, composed by Eleonora Bilotta*, pages 483–489. World Scientific, 2013.

[TLF+18a] T. Tseng, M. Li, D. N. Freitas, T. McAuley, B. Li, T. Ho, I. E. Araci, and U. Schlichtmann. Columba 2.0: A co-layout synthesis tool for continuous-flow microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 37(8):1588–1601, 2018.

[TLF+18b] T.-M. Tseng, M. Li, D. N. Freitas, A. Mongersun, I. E. Araci, T.-Y. Ho, and U. Schlichtmann. Columba s: A scalable co-layout design automation tool for microfluidic large-scale integration. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[TLHS15] T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann. Reliability-aware synthesis for flow-based microfluidic biochips by dynamic-device mapping. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[TLL+16a] T.-M. Tseng, B. Li, M. Li, T.-Y. Ho, and U. Schlichtmann. Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 35(12):1981–1994, 2016.

[TLL+16b] T.-M. Tseng, M. Li, B. Li, T.-Y. Ho, and U. Schlichtmann. Columba: Co-layout synthesis for continuous-flow microfluidic biochips. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[TLSH15a] T.-M. Tseng, B. Li, U. Schlichtmann, and T.-Y. Ho. Storage and caching: Synthesis of flow-based microfluidic biochips. *IEEE Design & Test*, 32(6):69–75, 2015.

[TLSH15b] T.-M. Tseng, B. Li, U. Schlichtmann, and T.-Y. Ho. Storage and caching: Synthesis of flow-based microfluidic biochips. *IEEE Design & Test*, 32(6):69–75, 2015.

[TLZ$^+$19] T.-M. Tseng, M. Li, Y. Zhang, T.-Y. Ho, and U. Schlichtmann. Cloud columba: Accessible design automation platform for production and inspiration. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2019.

[TMQ02] T. Thorsen, S. J. Maerkl, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.

[TNSP14] A. N. Tait, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal. Broadcast and weight: An integrated network for scalable photonic spike processing. *Journal of Lightwave Technology*, 32(21):4029–4041, 2014.

[VRK$^+$06] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett. First-order incremental block-based statistical timing analysis. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 25(10):2170–2180, 2006.

[WHD$^+$13] N. N. Watkins, U. Hassan, G. Damhorst, H. Ni, A. Vaid, W. Rodriguez, and R. Bashir. Microfluidic cd4+ and cd8+ t lymphocyte counters for point-of-care hiv diagnostics using whole blood. *Science translational medicine*, 5(214):214ra170–214ra170, 2013.

[WXX$^+$19] T. Wang, J. Xiong, X. Xu, M. Jiang, Y. Shi, H. Yuan, M. Huang, and J. Zhuang. MSU-Net: Multiscale statistical U-net for real-time 3D cardiac MRI video segmentation. In *Proc. Medical Image Computing and Computer Assisted Interventions*, 2019.

[WXXS19] T. Wang, J. Xiong, X. Xu, and Y. Shi. SCNN: A general distribution based statistical convolutional neural network with application to video object detection. In *Proc. AAAI Conf. on Artificial Intelligence*, 2019.

[WZY+17] Q. Wang, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai. Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 524–529, 2017.

[WZY+18] Q. Wang, H. Zou, H. Yao, T.-Y. Ho, R. Wille, and Y. Cai. Physical co-design of flow and control layers for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 37(6):1157–1170, 2018.

[XCP10] T. Xu, K. Chakrabarty, and V. K. Pamula. Defect-tolerant design and optimization of a digital microfluidic biochip for protein crystallization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(4):552–565, 2010.

[XY19] Q. Xia and J. J. Yang. Memristive crossbar arrays for brain-inspired computing. *Nature materials*, 18(4):309–323, 2019.

[XZH07] J. Xiong, V. Zolotov, and L. He. Robust extraction of spatial correlation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 26(4):619–631, 2007.

[YHC15] H. Yao, T.-Y. Ho, and Y. Cai. PACOR: practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips. In *Proc. Design Autom. Conf. (DAC)*, pages 142:1–142:6, 2015.

[YS+17] S. S. Yichen Shen, Nicholas C. Harris et al. Deep learning with coherent nanophotonic circuits. *naturephotonics*, 11:441–446, 2017.

[YWR$^+$15] H. Yao, Q. Wang, Y. Ru, Y. Cai, and T.-Y. Ho. Integrated flow-control codesign methodology for flow-based microfluidic biochips. *IEEE Design & Test*, 32(6):60–68, 2015.

[ZHL$^+$19] Y. Zhu, X. Huang, B. Li, T. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann. Multicontrol: Advanced control logic synthesis for flow-based microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2019.

[ZLH$^+$18] Y. Zhu, B. Li, T.-Y. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann. Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 123:1–8, 2018.

[ZLL$^+$19] Z. Zhao, D. Liu, M. Li, Z. Ying, L. Zhang, B. Xu, B. Yu, R. T. Chen, and D. Z. Pan. Hardware-software co-design of slimmed optical neural networks. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 705–710, 2019.

[ZSL18] M. A. Zidan, J. P. Strachan, and W. D. Lu. The future of electronics based on memristive systems. *Nature Electronics*, 1(1):22–29, 2018.

[ZWV17] J. Zhang, Z. Wang, and N. Verma. In-memory computation of a machine-learning classifier in a standard 6t sram array. *IEEE Journal of Solid-State Circuits*, 52(4):915–924, 2017.

[ZZL$^+$19] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann. Aging-aware lifetime enhancement for memristor-based neuromorphic computing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1751–1756. IEEE, 2019.

[ZZL$^+$20] Y. Zhu, G. L. Zhang, B. Li, X. Yin, C. Zhuo, H. Gu, T.-Y. Ho, and U. Schlichtmann. Countering variations and thermal effects for accurate optical neural networks. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7, 2020.

[ZZW$^+$20] Y. Zhu, G. L. Zhang, T. Wang, B. Li, Y. Shi, T. Ho, and U. Schlichtmann. Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1590–1593, 2020.