Energy-efficient DNN Training with Approximate Computing and Voltage Scaling

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2021

TaiYu Cheng

Publications

Journal Article (Refereed)

[J1] T. Cheng, Y. Masuda, J. Chen, J. Yu, and M. Hashimoto, "Logarithm-approximate floating point multiplier is applicable to power-efficient neural network training," *VLSI Integration*, vol. 74, pp. 19-31, 2020.

International Conference Papers (Refereed)

[I1] T. Cheng, J. Yu, and M. Hashimoto, "Minimizing power for neural network training with logarithm-approximate floating-point multiplier," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Rhodes, Greece, 2019, pp. 91-96.

[I2] T. Cheng, Y. Masuda, J, Nagayama, Y. Momiyama, J. Chen, and M. Hashimoto, "Mode-wise voltage-scalable design with activation-aware slack assignment for energy minimization," in *2021 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, accepted, 2021.

International Conference Papers (Unrefereed)

[C1] Y. Masuda, J. Nagayama, T. Cheng, T. Ishihara, Y. Momiyama, and M. Hashimoto, "Critical path isolation and bit-width scaling are highly compatible for voltage over-scalable design," in *2021 24th Design, Automation and Test in Europe Conference (DATE)*, accepted, 2021.

Summary

The proliferation of Artificial Intelligence (AI) has a wide range of impact on the worldwide daily life such as medicine, finance, and even industry. In many AI applications such as image, video, and speech processing, deep neural network (DNN) has been recognized as a vital solution in terms of its high quality and generality. DNN attains astonishing accuracy for tackling non-trivial and complicated Big Data analysis, though, at the cost of high computation. Even inference with a pre-trained DNN model requires considerable computation, and what is more, training stage demands further overwhelming amount of computation.

When pursuing a high-quality and general solution through DNN, we often rely on a large size of data for training in addition to sophisticated, i.e., very deep, DNN models, elevating the amount of parameters and computation. To make matter worse, to stabilize and converge the model parameters, several iterations (epochs) in training phase are indispensable, which elevates the computing cost. Both inference and training algorithms primarily perform matrix multiplication, which consists of a series of MAC (multiply-accumulate) computations. When the size of DNN models and input data increases, the amount of MAC computations also increases. Modern DNN training widely applies GPU in servers or cloud systems. The parallel-computing property of GPU can provide higher bandwidth for large-scale matrix multiplications and contribute to DNN training acceleration. On the other hand, while DNN training accelerates with the aid of GPUs, GPU is substantially a compute-intensive system and also highly demands power. Therefore, solutions for achieving an efficient DNN training scheme could undoubtedly save the resource and cost for many AI applications and facilitate the AI marketing revenue. Meanwhile, since the training quality is the most essential factor for a competitive DNN model, it is challenging yet worth studying to enhance a DNN training efficiency while sustaining its quality.

Besides GPUs, the demands for accommodating training capability in edge terminal servers or mobile ASICs also raise. These platforms have the advantage to provide more local services with privacy preservation. On the other hand, regarding the limited size and tiny-volume battery, developing an efficient training hardware becomes an essential requirement from AI service providers.

Approximate computing (AC) is a set of techniques providing cost-effective tuning knobs that make the applications enhance their efficiency such as speed, power, and area,

at the sacrifice of minor quality loss. DNN is one of the applications highly compatible with AC according to its natural characteristics of error tolerance and resilience. Several AC techniques are already explored to achieve efficiency improvement of DNN but mainly for inference purpose, assuming an accurate pre-trained model is ready. On the other hand, the training result is thought to be more sensitive and susceptible to computation inaccuracy once the DNN structure gets more complex, and then there is always a concern for final training quality once applying too much approximation in training stage. Therefore, the applicability of AC to DNN training is insufficiently explored. Only bit-width scaling (BWS) is majorly studied since it is intuitive and the approximation degree can be controlled gradually.

Due to gradient computation in the back-propagation algorithm, training is performed with a wide dynamic range of values, and therefore, many mainstream systems exploit floating-point (FP) expression for training. To attain sufficient quality, training of modern DNN models conventionally relies on 32-bit FP (FP32). On the other hand, some literatures claim that training with FP32 is more than necessary. While FP16 is recognized with its advantage of high computation and memory efficiencies, it is still controversial whether FP16 can attain the same training quality as FP32 without cooperating with other crafted strategies or techniques. In most of the cases, training with FP16 still requires FP32 to assist some computing that FP16 cannot well cover. Experimental results reproduced in this work reveal that the precision of FP16 training has not been well certified if tackling new real-world datasets or applications. The results also indicate that when there are only two choices of FP32 and FP16, it is difficult to well balance the quality and efficiency.

Voltage scaling (VS) is a general means to reduce the power dissipation of integrated circuits in a quadratic manner, whereas AC is effective for applications possessing inherent error tolerance. However, the delay of logic gates would increase and the circuits become risky to occur timing error as the voltage decreases. In order to achieve aggressive voltage scaling while preventing the timing errors, activation-aware slack assignment (ASA) is proposed to reallocate the timing margin for timing critical paths that might be activated during the operation. The previous works of ASA introduce mean-time-to-failure (MTTF) analysis regarding the timing error as stochastic events. MTTF-aware ASA can aggressively scale down the voltage for power reduction at the cost of allowing less but not zero timing error. Though ASA is thought effective, the previous works did not provide a methodology to maximize its benefit with guaranteeing no timing errors, which restricts its generality and consequently application domains. Then, an interesting question is raised; how much can ASA benefit VS efficiency with ensuring no timing errors?

This dissertation aims at providing a design methodology to achieve energy minimization for DNN training with useful implementation techniques. First, to reduce the primary MAC computation in training with FP, this work proposes to use logarithm-approximate multiplier (LAM) for training. By approximating float-point multiplica-

tion to cheap fixed-point addition, the NN training engine is expected to be implemented with smaller delay, fewer gates, and lower power consumption. With a dedicated hardware for NN training engine and a 2-D classification dataset, this dissertation demonstrates that LAM-based training can achieve 10% speed-up, 2.3X power and area improvement, respectively, compared with traditional training with FP32 exact computation. In addition, LAM also reveals its high compatibility with conventional BWS. If LAM is applied to NN where BWS is already implemented, LAM and BWS can enjoy synergy effect and up to 4.9X power reduction is attained with sacrificing up to 1% accuracy loss. Also, the processor level design can exploit the advantage of LAM. An experimental GPGPU embedded with LAM executing a NN training workload presents 28% power reduction improvement and the improvement reaches 41% with LAM + BWS. Finally, LAM is qualified for deeper-layer NN training. Up to four hidden layers, LAM- based training yields the same level of accuracy as training with accurate multipliers, even with aggressive BWS.

Next, this dissertation addresses the disadvantage of non-zero timing error risk involved in conventional ASA methods and provide a design applying to mode-wise voltage-scaling (MWVS) design with enhancing ASA benefit on VS efficiency yet guaranteeing zero timing errors. First, the MWVS design is formulated as an optimization problem that minimizes the overall power consumption considering each mode duration, as well as the achievable voltage reduction, and accompanied circuit overhead. The proposed method explores the solution space with the downhill simplex algorithm (DSA). To attain a solution, i.e., a design, the feature of multi-corner multi-mode (MCMM) in a commercial tool is exploited to perform mode-wise ASA, where the ASA is realized by assigning with sets of false paths specialized for individual modes to obtain the maximum voltage scaling for each mode. Experimental results based on a popular RISC-V hardware design show that the proposed MWVS design can save 20% more energy compared with the conventional VS approach and acquire 15% more gain compared with single-mode ASA. In addition, the cycle-by-cycle fine-grained false paths identification is also proposed and it successfully reduced 42% leakage power.

Finally, this dissertation provides an overall scheme for minimizing NN training energy as the main contribution of this work and gave it a name as adaptive bit-width and voltage scaling (ABVS) which can leverage the FP units with configurable bit-width. The key idea is that the NN training starts with smaller fraction bit-width (FB), and then FB is gradually increased depending on present training quality during the training phase. Since smaller FB possesses shorter latency, this training scheme can concurrently adapt the bit-width and voltage scaling and hence intensify energy reduction. Experimental results across a various scale of training datasets and DNN applications reveal that the proposed ABVS flow in training can achieve the comparable training quality as FP32, but up to 62% energy reduction, and at most 37% reduction even compared with the training under the least sufficient FB, with at most 0.5% quality loss. This work also gave an explanation of why ABVS works in modern DNN training. Furthermore,

the investigation of different rounding schemes is conducted. Results show that conventional round-to-the-nearest (RTNE) provides better trade-off between training quality and energy efficiency than round-to-zero (RT0).

Overall, this dissertation seeks for achieving energy minimization with two main countermeasures, "approximate computing" and "voltage scaling." LAM and BWS techniques are exploited in this work as the former, while MWVS belonging to the latter combines MCMM and ASA to achieve further voltage reduction. The ABVS flow can be recognized as an integrated solution for all the aforementioned strategies. In addition, all the methods apply to GPUs and the dedicated ASICs for achieving efficient DNN training.

Acknowledgments

First of all, I would like to express my deepest gratitude to Professor Masanori Hashimoto in Osaka University for providing me a precious opportunity and an excellent environment to study as a doctoral student in his laboratory. All of my productive researches are credited to none other than him. His advanced perspective and thoughtful advise led me to to the achievements.

I would like to appreciate Associate Professor Hiromitsu Awano in Kyoto University, Professor Katsuyoshi Miura and Associate Professor Ittetsu Taniguchi in Osaka University for detailed reviews and insightful suggested comments.

I would like to appreciate Associate Professor Jaehoon Yu in Tokyo Institute of Technology, Assistant Professor Yutaka Masuda in Nagoya University, Dr. Jun Chen for their precious suggestions and enormous help throughout my doctoral research.

My appreciation also goes to Mr. Yoichi Momiyama, Mr. Jun Nagayama in Socionext Inc. for technical discussions and suggestions.

I would like to thank other colleagues who belong or belonged to the Integrated System Design Laboratory in Osaka University for daily discussions and their support: Dr. Wang Liao in Kochi University of Technology, Dr. Koichi Mitsunari, Dr. Ryutaro Doi, Mr. Ryo Shirai, Mr. Pei-Hao Chen, Miss Yangchao Zhang, Mr. Dehua Liang. I express my heartful thanks to all members of Integrated System Design Laboratory in Osaka University for having an interesting and comfortable time in the laboratory. I would like to thank Mrs. Asako Murakami and Mrs. Naoko Isozaki for her various support. I would like to thank Japan-Taiwan Exchange Association to provide me a big financial support so I can enjoy my time for the daily life in Japan. I also would like to thank all of my friends for having good times.

Finally, I give my thank to family (Father, Mother, younger brother) for supporting my livelihood, and sincerely thank to my dear girl friend, Miss Yu-Ting Lin to cheer me for studying abroad and support me for pursuing this degree.

Contents

List of Figures

List of Tables

Abbreviations

ABVS	Adaptive Bit-width and Voltage Scaling
AC	Approximate Computing
AES	Advanced Encryption Standard
AI	Artificial Intelligence
ALU	Arithmetic logic unit
ASA	Activation-aware Slack Assignment
ASIC	Application-Specific Integrated Circuit
AVS	Adaptive Voltage Scaling
BWS	Bit-Width Scaling
CF-FPI	Cycle-by-cycle Fine-grained False Path Identification
CPI	Critical Path Isolation
CPU	Centor Processing Unit
DNN	Deep Neuron Network
DSA	Downhill Simplex Algorithm
ECO	Engineering Change Order
EDA	Electronic Design Automation
EFM	Exact floating-point Multiplier
FB	Fraction Bit
FF	Flip Flops

FP	Floating Point
FPGA	Field Programmable Gate Array
FPI	False Path Identification
FPU	Floating-Point Unit
GPGPU	General Purpose Graphic Processing Unit
GPU	Graphic Processing Unit
GTA	Gradual Training Approximation
IoT	Internet of Thing
ISA	Instruction Set Architecture
LAM	Logarithm-Approximate Multiplier
MAC	Multiply-ACcumulate
mAP	mean Average Precision
MCMM	Multi-Corner Multi-Mode
ML	Machine Learning
MLP	Multi-Layer Perceptron
MTTF	Mean-Time-To-Failure
MWVS	Mode-Wise Voltage Scaling
NCT	Non-Critical Transition
NN	Neural Network
PLAN	Piecewise Linear Approximation of a Nonlinear function
RT0	Round-To-zero
RTNE	Round-To-the-Nearest-Even
SOC	System-On-Chip
TPU	Tensor Processing Unit
VOS	Voltage Over-Scaling
VS	Voltage Scaling
WNS	Worst Negative Slack

Chapter 1

Introduction

This chapter provides the background and the objectives of this dissertation. This dissertation aims at providing a design methodology for energy-efficient deep neural network (DNN) training system. First of all, this dissertation briefly describes the basic of neuron networks and then describes the survey for approximate computing and voltage-scaling design paradigms. The major challenge of efficient training with approximate computing and voltage scaling are then discussed. Finally, this chapter provides the objective and the overall organization of the dissertation.

1.1 Background

Artificial intelligence (AI) has become the ubiquitous technology indispensable for personal daily life. It drives Big Data analysis and benefits many areas such as medical treatment, finance, and industry. Statistical data also reveals that the investment of AI incredibly grows up in recent years and could even burst in the near future. Fig. 1.1 [1] shows that AI market/revenue grows up to 22.6 billion USD by 2020, and the data forecast that the value would reach 126 billion USD by 2025.

Deep neural network (DNN) has thrived and formed the foundation for many advanced AI applications [2, 3] such as image classification, object detection [4, 5], and speech recognition [6]. DNN model performs excellent feature extraction for the given input data e.g., images, videos, audios, or speech, and it possesses even superior capability than human perception. Therefore, DNN contributes to unprecedented success in many AI tasks thanks to its outstanding accuracy and generality. While DNN already achieves state-of-the-art accuracy, it pays the cost of considerably intensive computation [2, 7].

Generally, a DNN computing system consists of two stages; training and inference. A DNN model needs to optimize its parameters of the network, e.g., weights and bias, during the training stage with a set of data, and then the trained model can perform the inference with the newly given input.

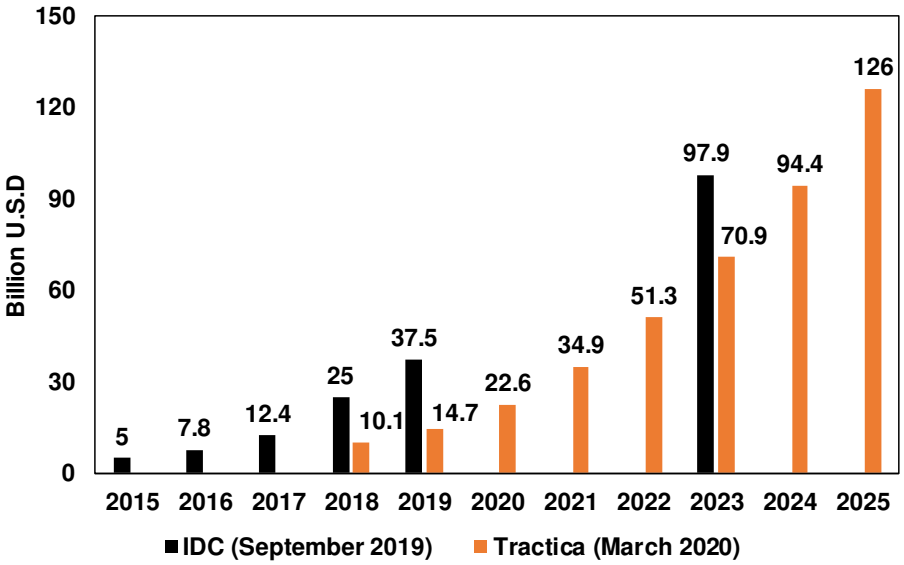


Figure 1.1: Market size/revenue comparison for artificial intelligence worldwide from 2015 to 2025.

Empirically, a DNN with more sophisticated structure, e.g., deeper layers, possesses higher capability for inference, Fig. 1.2 shows the top competitive DNN models at ILSVRC, a worldwide top contest for image classification and object detection, from 2012 to 2015 (collected by the reference [4, 7, 8]). The ImageNet database [9], which is used for this public challenge, is very popular and representative in academia and industry for evaluating the contemporary DNN models regarding their inference strength. The database includes 1.3 million images for training, 50,000 images for testing, and 1000 classes. Fig. 1.2 shows that more layers would offer higher accuracy. In 2015, ResNet, which is a well-acknowledged DNN model, achieves 4% top-5 error with utilizing up to 152 layers. It is conceivable that such a huge DNN model would demand considerable computations even merely for inference, and the amount of computations for training is further huge. Training needs to conduct more operations according to the algorithm than inference. Besides, compared with inference, training a model requires an even large amount of dataset. Moreover, several iteration cycles are required to optimize and converge the model during the training. Therefore, a more complicated algorithm, a large-size dataset, and many numbers of iterations contribute to the overwhelming amount of computations for training compared with inference.

Table 1.1 based on ImageNet [9] reveals that the CPU time difference between training and inference can lead to 1,300X according to large-scale of dataset (1.3 million images) and tremendous iterations (250,000 times with the batch size of 256). Since Energy = Power $\times$ Time, even if not considering the power difference, the overwhelming energy cost in training compared with inference is understandable. In order to save

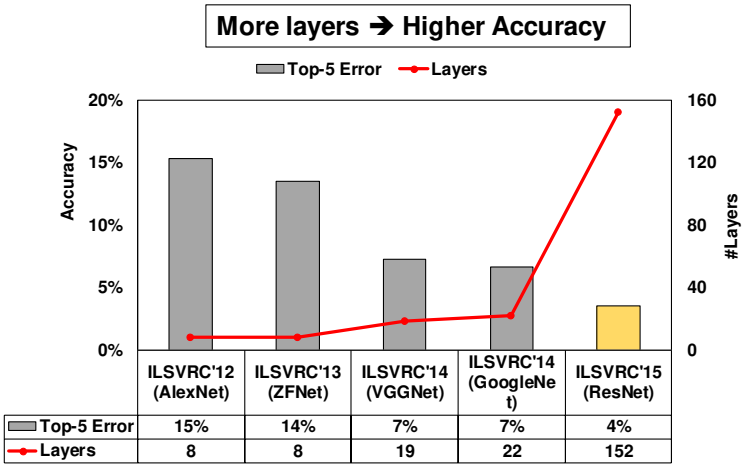


Figure 1.2: Accuracy vs. number of layers in DNN for ImageNet classification (#training dataset: 1.3 million). Empirically, models with more layers achieve higher accuracy.

Table 1.1: Processing time comparison between training and inference phases to train ImageNet for 50 epochs with CPU: Intel E5-v4 + GPU: Tesla V100 DGXS.

ImageNet dataset on Darknet reference model [12]		
Phase	Processing Time (sec)	Ratio
Training	52,131	1,303X
Inference	40	1X

the necessary run time, modern DNN training relies on GPUs (graphic processing units) to achieve higher throughput and shorter latency. GPU contains numerous arithmetic logic units (ALUs) and allows tens-of-thousands of parallel computing assuming it has 1000 cores. Such outstanding capability of parallel computing benefits DNN for realizing training acceleration. However, GPU is known for its huge power consumption [10], considering that the real-time data surges with an astonishing number (about 10 fold from 2014 to 2020) as shown in Fig.1.3 [11], the even bigger data keeps threatening the resource of inference and, not to mentioned, the training systems. Energy-efficient training would benefit GPU for the heat removal and power delivery, which improves the chip-level and system-level reliability and reduces the cost for heat removal and power delivery. To facilitate the marketing revenue in modern AI applications, the trend for the increasing amount of data drives the strong motivation for developing an efficient training mechanism.

Besides GPUs, efficient training based on more size-limited devices, such as FPGA [13, 14], or dedicated application-specific integrated circuit (ASIC) is also demanded [15]. Commercially available products developed based on different platforms to yield machine learning or DNN throughput improvement are exemplified in Table 1.2. Re-

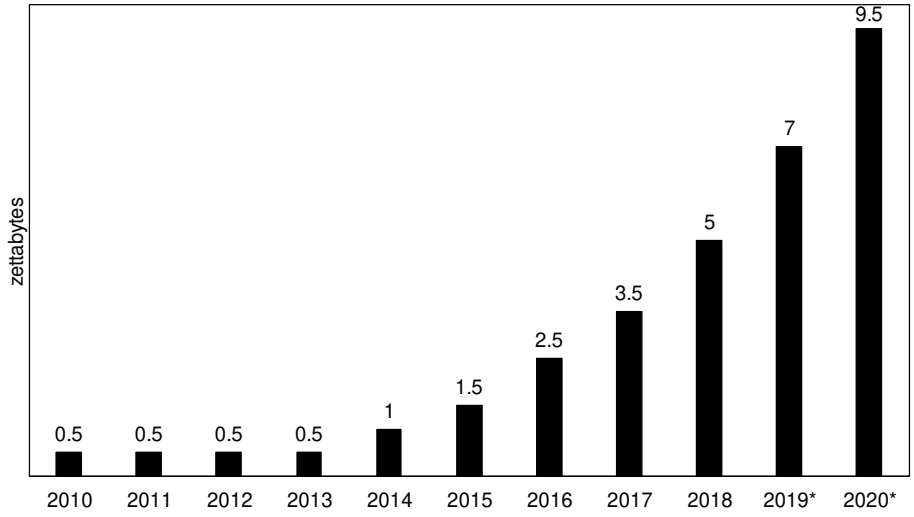


Figure 1.3: Annual size of real-time data in the global datasphere. The Global Datasphere quantifies and analyzes the amount of data created, captured, and replicated in any given year across the world.

Table 1.2: Commercially available products for AI/DNN acceleration.

GPU	NVIDIA A100, V100
FPGA	Xilinx Zynq DPU
ASIC	Google Coral Edge TPU, Nvidia Jetson Nano, Intel Movidius VPU

cently, the emerging markets of on-line or transfer learning move the infrastructures of training from the cloud to edge server-level computing [10, 16, 17] or mobile devices, e.g. Internet-of-Things (IoT). Training on server- or device- levels can provide more local and *in situ* services because they have the benefits of shorter latency and privacy protection. However, due to the size and power limitation, it is essential to provide a solution for efficient training systems [18, 19].

Different computing platforms have different training properties regarding their quality and efficiency. Fig. 1.4 shows the trade-off between the accuracy (denoted as prediction error) and power consumption based on four types of hardware platforms [20]. According to the figures, though GPU can achieve the highest accuracy, it also pays the largest power consumption. On the other hand, Fig. 1.4 shows that the dedicated design (ASIC) is very convenient to trade accuracy with efficiency and could be applicable for many different applications. In addition, ASIC can either operate standalone or be embedded into GPU, FPGA, or even CPU, which provides high design freedom and versatility.

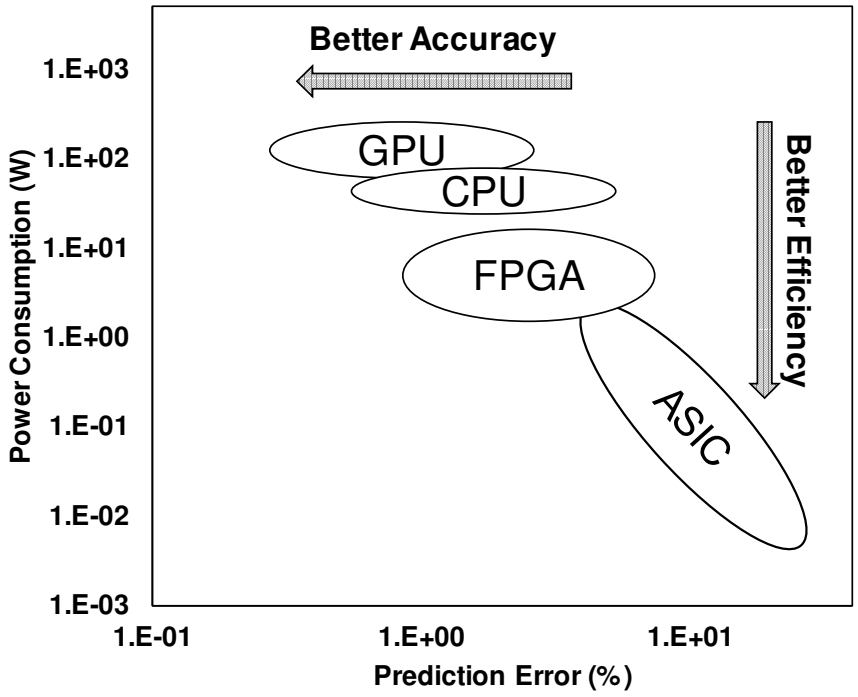


Figure 1.4: Prediction error and power consumption of hardware platform.

1.2 Basics of Neural Networks

Neural Network (NN) is one of the most widely-used techniques in machine learning, and DNN is one of its subset where they share similar basic architecture. A basic feedforward NN model is composed of a few to hundreds of layers, each of which includes a number of neurons. The neurons are connected layer by layer through synaptic weights. The synaptic weights are optimized to provide sufficiently high accuracy through the computationally expensive training phase. Hardware NN system is mainly categorized into two types. The first one is the inference engine that processes a network with given pre-trained weights, and the latter is the training engine that has the additional capability of weight optimization in the training phase. Regardless of the inference or training engine, multiply-accumulate (MAC) arithmetic computation is the primary operation. The rapidly increasing trend of NN size to deal with more intricate and sophisticated problems explodes the amount of MAC computation, resulting in a strong demand for efficient hardware engines.

Fig. 1.5(a) illustrates a multilayer perceptron (MLP) structure [21], which is known as a basic feedforward NN. For the sake of clarity, the structure contains only one hidden layer, while the number of hidden layers can be extended to form deeper NNs. The state of each neuron in the network is computed from all the states of neurons in the previous

layer and then is propagated its state to the next layer. Taking the example in Fig. 1.5(b), since all the states are pre-determined in the input layer I, the state computation starts at each neuron in the hidden layer (neurons are denoted as H_1, H_2,..., H_h). Each neuron in the hidden layer computes the sum of all the states of neurons in the input layer (I_1, I_2,..., I_i) multiplied with corresponding synaptic weights (W_H), passes the sum with a bias term (B_H) through a non-linear activation function to determine its state, and then propagates the state to the output layer. This operation is repeated at the neurons in the output layer (O_1, O_2, ..., O_o) again, but here the sum of all the states of neurons in the hidden layer is computed with weights (W_O) and bias (B_O) to determine the states. The procedure finishes once all the states of neurons in the output layer are determined.

A basic unit for expressing the above operation is shown in Fig. 1.5(b). Suppose there is the a-th neuron in the i-th layer, its state Y_a^i can be computed by the following formula:

$$Y_a^i = \text{Act}\left(\sum_{k=1}^{N} W_{ka}^i Y_k^{i-1} + B_a^i\right), \tag{1.1}$$

where Y_k^{i-1} denotes the state of the k-th neuron in the $(i-1)$-th layer. W_{ka}^{i-1} represents the synaptic weight connecting from the k-th neuron in the $(i-1)$-th layer , and B_a^i denotes the bias term for the a-th neuron in the i-th layer. Act(.) means the activation function, which usually allows passing ≥ 0 values or limits the values between -1 and 1. This procedure, so-called forward propagation, keeps going until the states of all the neurons in the output layer are determined, which is the core and dominant operation that an inference engine with pre-trained weights executes.

On the other hand, training NN aims at finding a set of synaptic weights and bias values to minimize the loss function (Loss), which is usually defined as the error squared between the state from the forward propagation results in output layer O and target label T $(T_1, T_2, ..., T_o)$ [21]. For illustration purposes, the below describes the standard back-propagation. At the beginning of the training phase, all the weights are randomly initialized, the biases may be initially set to 0, and then forward propagation is launched. The next step is to reduce the loss function according to the contribution of each synaptic weight (W) and bias (B), which can be obtained through computing their gradient, i.e. $\partial\text{Loss}/\partial W$ and $\partial\text{Loss}/\partial B$. Based on the computed gradients, each synaptic weight and bias can be numerically updated during each iteration. Let us take Fig. 1.5(c) as an example. Suppose a synaptic weight W_{za}^i connects the z-th neuron in the $(i-1)$-th layer (state = Y_z^{i-1}) with the a-th neuron in the i-th layer (state = Y_a^i) and a bias B_a^i is for the a-th neuron in the i-th layer. Then, W_{za}^i and B_a^i are updated by:

$$W_{za}^i \mathrel{+}= -\eta \frac{\partial \text{Loss}}{\partial W_{za}^i} \text{ where } \frac{\partial \text{Loss}}{\partial W_{za}^i} = \delta_a^i Y_z^{i-1}, \tag{1.2}$$

$$B_a^i \mathrel{+}= -\eta \frac{\partial \text{Loss}}{\partial B_a^i} \text{ where } \frac{\partial \text{Loss}}{\partial B_a^i} = \delta_a^i. \tag{1.3}$$

The gradient terms $\partial \text{Loss}/\partial W^i_{za}$ and $\partial \text{Loss}/\partial B^i_a$ in Eqs. (1.2) and (1.3) share the same term δ^i_a while $\partial \text{Loss}/\partial W^i_{za}$ further includes the term Y^{i-1}_z. Basically, the gradient terms would decay during the weight and bias update, and thus these are also called gradient decent method. η is the learning rate, and δ^i_a is conditionally formulated as follows. If the i-th layer is the output layer, then δ^i_a is:

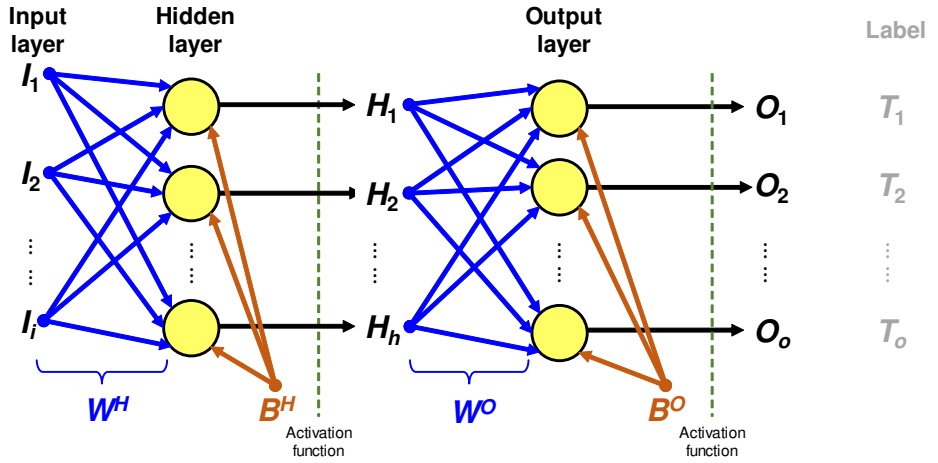
$$\delta^i_a = \text{Act}'(O_a)(O_a - T_a), \tag{1.4}$$

where O_a represents the state computed through forward propagation, T_a means its target state, and Act$'$ means the derivative of activation function. If the i-th layer is not the output layer, then referring to Fig. 1.5(c), δ^i_a is expressed as:

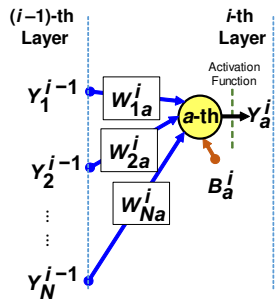
$$\delta^i_a = \text{Act}'(Y^i_a)\left(\sum_{n=1}^{b} W^{i+1}_{an}\delta^{i+1}_n\right), \tag{1.5}$$

where W^{i+1}_{an} denotes the synaptic weight to the n-th neuron in the $(i+1)$-th layer. δ^{i+1}_n can be recursively computed through Eqs. (1.4) and (1.5). Note that, with Eqs. (1.4) and (1.5), the output loss is propagating backward from the output layer, and thus this procedure is named as back propagation [21]. In addition, Eq. (1.5) indicates that the δ^i_a in the non-output layer needs to compute all the weighted sum of δ^{i+1}_n, meaning that MAC computation is also primary in back propagation. Therefore, the training phase executes huge amount of MAC computations during the iteration loops of forward and back propagation.

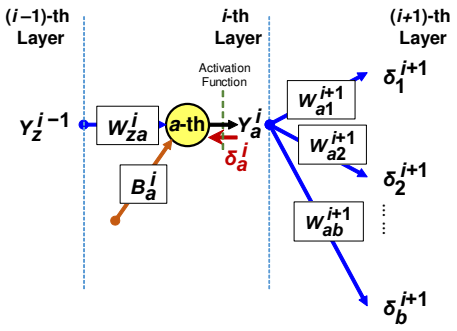
In addition, the gradient terms have a large dynamic range [22]. A simple example can help understand this property. Fig. 1.6 plots the distribution density of the gradient values found when training a NN for MNIST dataset [23]. The gradient values spread from 2^{-47} to 2^6. As shown in Fig. 1.7, if adopting fixed-point expression, more than 50 bits (6+47 = 53) are required to cover this range, while floating-point expression spends only a few bits for exponents (e.g. 7 bits if considering bias for negative value) and some extra bits α for fraction parts (only corresponded to precision rather than coverage) to cover this wide range. Thus, adopting floating-point units (FPUs) is beneficial for training engines to accommodate such gradient computation.



(a)



(b)



(c)

Figure 1.5: Feed-forward neural network. (a) is a schematic of a feed-forward neural network with 1 hidden layer. (b) and (c) are the schematics for illustrating forward- and back-propagation at the a-th neuron in i-th layer.

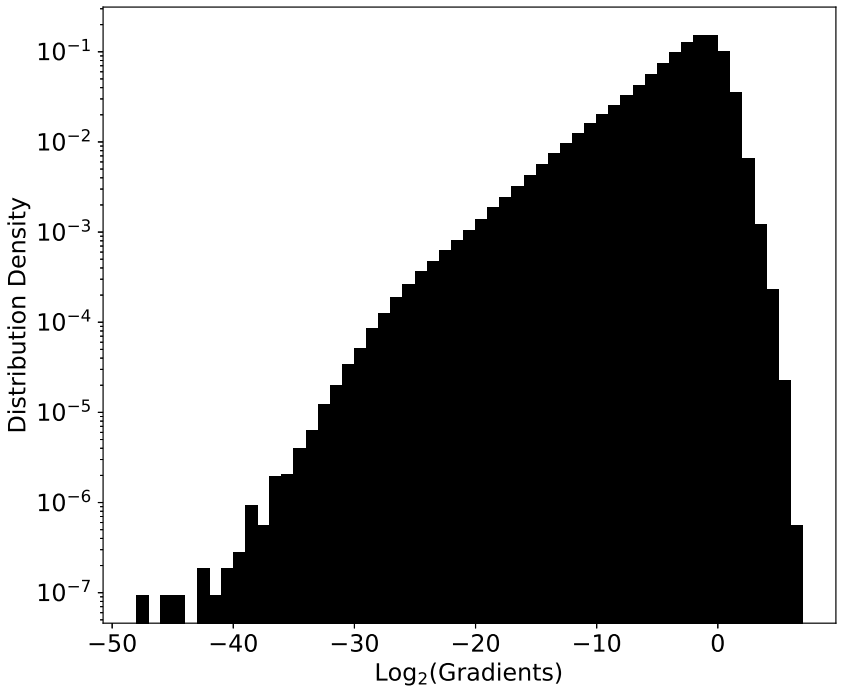


Figure 1.6: Distribution density of gradients observed when a NN is trained for MNIST dataset. x-axis represents the gradients in log scale with base 2, and y-axis is the normalized distribution density.

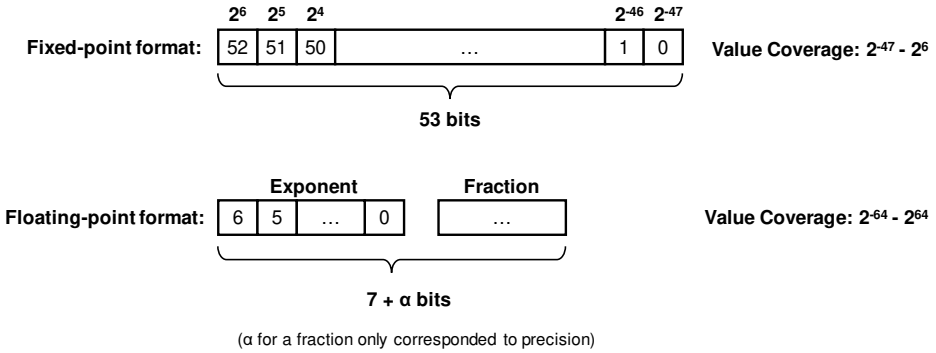


Figure 1.7: Different coverage of value for fixed-point and floating-point data format.

1.3 Energy-efficient DNN Training

The works related to efficient training spread over various area because of different motivations [10, 24, 25, 26, 27, 28, 29, 30, 31]. To train a NN with large size of data through high-end GPU systems, some works assume the computing resource is sufficient, e.g., computing in the cloud, and hence introduce "distributed training" to launch huge parallel computing with 256 or even more GPU cards to minimize the training processing time [24, 25, 26]. In other words, these works focus on "run-time" instead of "energy" efficiency.

Energy-efficient training roughly contain two categories (1) efficient DNN structures and (2) power/energy reduction, as listed in Fig. 1.8. The first group is about the efficient DNN structures. These works introduce compact DNN structures to effectively compress the model size and the number of parameters, which can enhance the efficiency both in training and inference with sustaining the accuracy. Representative works are like SqueezeNet [29], ShuffleNet [30], EfficientNet [31], etc. These works can also benefit distributed training by requiring less GPUs. Therefore, although the efficient DNN structures may benefit energy saving in use due to the lightweight models, still, the main intention for developing the efficient DNN structures is to improve the "run-time" efficiency when tackling large size of dataset through high-end GPU designs.

On the other hand, most of the motivations of the works in the second group are to accommodate the training into server- and device- level designs instead of high-end GPU design system. Since training in server- or device- level design is limited with size and memory storage, these works pursue higher energy-efficiency in training allowing acceptable quality sacrifice. Researchers in the this group focus on providing the solutions to achieve power/energy reduction through several levels of design methods such as algorithm, data representation, and low-power design methodology. Note that these methods are general means and can be applied to any DNN structures. Algorithm level such as weight/net pruning and sharing [32] or data sampling [24] are proved to be effective for improving efficiency. As for the data representation, training in shorter floating-point (less than 32-bit) [33, 34, 35], fixed-point [27, 28] or in log-domain [36, 37] are the choices for efficiency enhancement. And then for low-power design approach, voltage scaling or voltage over-scaling are the representative mechanisms [38]. Since many techniques applied in the second group require to trade the quality (accuracy) with the power efficiency, this area frequently leverages the knowledge from approximate computing (AC). Besides AC, voltage scaling (VS) is also an effective method in low-power design area, and VS can be considered as an orthogonal way to AC, i.e, AC and VS are combinable.

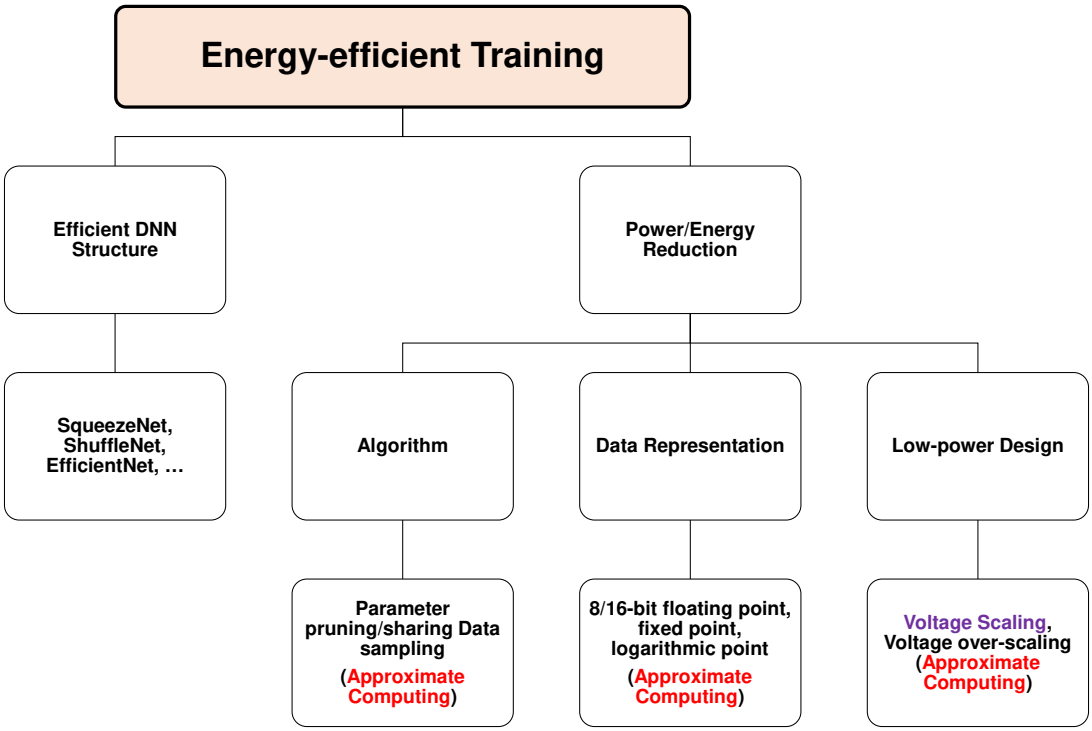


Figure 1.8: Categories of the area in energy-efficient training.

1.4 Approximate Computing to NN

Approximate computing (AC) is recognized as one promising technique to enhance the efficiency of computer systems since many modern applications require large-scale computation and memory storage demands which may burden current available resources [39, 40, 41]. In addition, many growing applications such as recognition, data mining, synthesis, and media processing (audios, images, or videos, graphics) [39, 41], have a common characteristic that it is usually nearly impossible or requires prohibitively high cost for those applications to find an optimal solution but allows to pursue less-than-optimal results or said approximation. Therefore, AC is introduced to be applied to computing systems for trading the efficiency (speed, power/energy, area) with the quality (approximation output results). The AC techniques rely on the statistical properties of computing systems which allows non-zero quality loss for improving efficiency. Although the manufacturing and environmental stochastic variations of the hardware due to process, voltage, temperature, or aging could also induce inaccurate computation results, they are not considered in the definition of AC. AC area assumes that the designd hardware has no stochastic variations [41].

Fig. 1.9 illustrates the overall framework of the AC flow [39]. The AC flow briefly can be categorized into two levels; design level and operation level. The former is

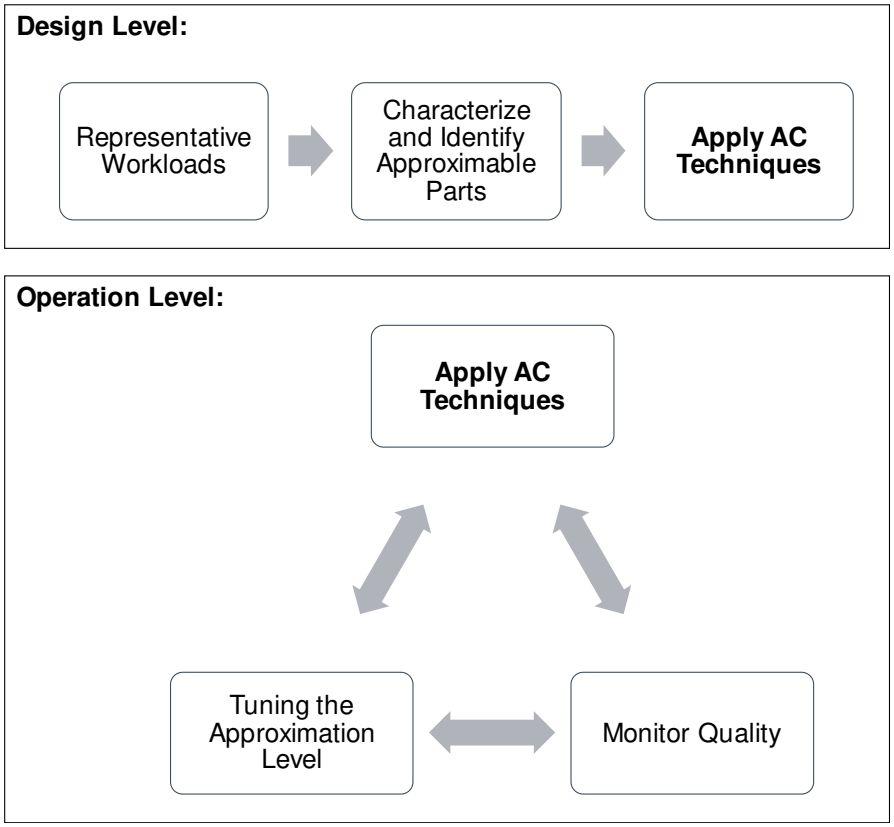


Figure 1.9: Framework of AC.

amenable for the applications (workloads) that are already explicitly specified. Therefore the users can characterize and identify the blocks by observing their impact on the final quality if applying with AC techniques. Finally, the determined approximable blocks can be more aggressively approximated while the other parts adopt light-weight AC techniques or accurate operations. On the other hand, the latter directly applies AC techniques on the system first but with a quality management system to detect the unacceptable error. Once the error is detected, it tunes the approximation level or applies error recovery schemes to improve accuracy. The computing systems have many layers (software, architecture, hardware), and hence the various AC techniques are also distributed in different layers. Note that the AC techniques are not independent in different layers, and occasionally multiple AC techniques and layers are necessary to be invoked simultaneously.

1.4.1 Software-level AC Techniques

AC techniques in software mainly relate to the algorithm where the invocations are related to stochastic computations such as average, accumulation, count, and percentile. For these stochastic computations, sampling is recognized as an efficient way to save computing and can achieve acceptable accuracy [42]. Therefore, some iterations of loops can be skipped or reduced (named as loop perforation) for the aforementioned computations in addition to randomized or Monte-Carlo algorithms [43, 44]. Another type of AC technique at the software level corresponds to the programming language that it is capable to annotate or detect which part of the blocks or segments in the codes are approximated, and then system follows the instruction and computes them in approximate storage (architecture-level) or circuits (hardware-level). These kinds of techniques usually need to enhance the present language or develop a new language. It, of course, necessitates AC techniques in compilers and non-software levels as well [45, 46].

1.4.2 Architecture-level AC Techniques

AC techniques for architecture-level mainly involve the enhancement of the interface between software and hardware such as instruction set architecture (ISA) to accommodate AC features [47]. Other kinds of AC techniques belong to this group are storage or say memory approximation [47, 48, 49]. The data can be stored in an approximate manner with lower-bit truncation. In SRAM, a lower voltage can be applied as long as the data can be stored with approximated value [48]. The refresh rate in DRAM can be separately assigned with regular values for critical data while the much lower value for non-critical or approximable one [49].

1.4.3 Hardware-level AC Techniques

This dissertation mainly applies AC techniques to hardware to improve NN training efficiency. AC techniques at the hardware level are intensively studied since this layer primarily affects efficiency. In addition, regardless of the inference or training phase, NN (DNN) algorithms involve high computation, where the AC techniques in hardware can directly benefit it. Below, existing hardware-oriented AC techniques are described.

Approximate Adder

Many applications, such as image processing, need to perform addition [41], and the primary MAC computation in NN algorithm also utilizes addition and accumulation. Therefore, approximate adder circuits are studied [50, 51, 52, 53]. Reference [50] introduced to partition an adder to many segments and the carry operations between the separated segments are truncated to simplify the circuit architecture. Reference [51] directly applies aggressive logic reduction at transistor-level. Conventionally, aggressive

approximation frequently targets least-significant-bit (LSB) to prevent severe accuracy loss. Since the carry operation could be even more complicated along with the increased bits, the partial carry skip manipulation can alleviate the design cost and achieve higher efficiency.

Approximate Multiplier

Multiplier is one of the most power-hungry and area-expensive operators, especially in FPUs [54]. Therefore, many researchers try hard to develop its approximate version to save computing resources. The approximate multiplier is especially adopted in NN for alleviating heavy MAC computation. A simple way to approximate multiplication can be implemented by directly rounding the input values to fewer bits before multiplication [55]. Also, since the multiplier circuits utilize many half or full adders to handle partial products, the techniques forming approximate adder mentioned in above can also be leveraged to form approximate multipliers. Besides, logarithm based multiplier is also proposed [36, 37, 56, 57, 58, 59] since logarithm converts multiplication to addition.

Bit-width Scaling

Bit-width scaling (BWS) is a classical AC technique that uses fewer bits in computation. Its precision degrades at a smooth pace, which results in a more moderate approximation than other AC techniques such as an approximate multiplier. Many researches prefer to adopt BWS in NN since fewer bits achieve shorter latency and save incredible computation and even memory consumption [60, 61]. BWS is popular not only for NN inference engine but also for training due to its tractability, even for a very deep DNN model [33, 34, 35]. Although mainstream computing systems such as CPU and GPU conventionally apply FP32 (floating-point expression with 32 bits in total) for training modern DNN model, it is now considered that FP32 is more than necessary. Then, the training with a shorter format is explored for improving the training efficiency, such as FP16 [33, 34] or even partially FP8 [35]. Recently, tensor process unit (TPU) is developed based on an 8/16-bit format rather than 32-bit in order to achieve a light platform and DNN acceleration [15].

Voltage Over Scaling (VOS)

At the operation level, reducing the voltage while fixing the operation clock period is the most direct way to achieve power/energy reduction [48, 62, 63]. However, regarding a fact that the circuit may experience timing error due to slower delay (latency) under low voltage, many computation errors or system catastrophic issues may happen. On the other hand, as long as those parts have the capability to tolerate the issues from timing error, the voltage can be aggressively scaled down to pursue significant power/energy efficiency. Conventionally, the scaled voltage in conventional VS would convey the

minimum voltage that induces zero timing error. On the other hand, VOS allows a certain timing error, and then it would reach even lower voltage than conventional VS.

Hardware AC in NN Inference and Training Phase

Several researchers study how to apply AC techniques to NN [32, 38, 55, 60, 62, 64, 65, 66, 67]. A notable thing is that researches related to AC on inference engine are in majority rather than training engine. That is because of the different primary concerns. Given a well pre-trained NN model, the inference engine can be deeply compressed with [32, 64, 65, 66, 67] or without [58, 68, 69, 70] a little additional training. Even distillation is notably studied to shrink the NN model size [71]. Since NN is a kind of highly approximable paradigm, a light-weight inference engine with sustaining acceptable accuracy is realizable, and thus they can be accommodated in mobile systems, such as IoT [72].

On the other hand, quality (accuracy) is the most competitive factor in training NN. A more sophisticated NN (DNN) model accompanied by a large scale of the training dataset is thought to be highly sensitive to the approximation [55, 68], and aggressive approximation may result in low-confidence quality in training. In addition, since training systems need to record more temporal parameters (gradients, updated weights) compared with inference systems, AC techniques applied to training schemes rely on more smooth or moderate approximation in addition to memory-friendly strategy, such as BWS [33, 34, 35]. Training in totally fixed-point [27, 28] or log-domain [37] could be alternative ways since fixed-point and logarithm domain can highly compress the necessary bits of data representation from floating-point values and can significantly save the memory resource.

1.5 Low-power Design Methodology: Voltage Scaling

Low-power operations are eagerly demanded in various computing systems, such as IoTs [72, 73], wearable equipment [74], and the sensor networks [75], in addition to mobile terminals. According to long standby time, tiny volume, and limited energy sources, those applications have strong demands for consuming ultra-low power. Also, power consumption becomes the most competitive factor in modern mobile SOC [76], or even high-performance chips [77]. Designers are dedicated to pursuing the maximization of power or energy efficiency.

Voltage scaling (VS) is one of the most common and powerful techniques for power reduction. An AC technique, VOS, can be viewed as its subset and is already investigated to improve DNN efficiency [62]. Voltage reduction remarkably contributes to the quadratic gain of power-saving with the fundamental equation:

$$P = \frac{1}{2}CV^2 \tag{1.6}$$

where P is the power, V is the voltage and C is the capacitance which should be invariant for a fabricated circuit. However, voltage decrease involves an increase in the circuit delay and raises the possibility of timing error. Therefore, the techniques to prevent timing errors are studied, and they can be categorized into two levels; operation level and design level. The former implements *in situ* monitoring devices in the circuits to adapt the supply voltage to maximize the power efficiency while sustaining circuit functionalities. These techniques are categorized as adaptive voltage scaling (AVS) techniques. The key idea is to insert the monitoring sensors behind the main circuits to detect or predict the logic error. If the main circuit keeps functional, then the voltage controlling unit keeps lowering down the supply voltage. On the other hand, once the sensors detect or predict the logic error, they would inform the controlling units to increase the voltage to maintain the functionality of the main circuit, as shown in Fig. 1.10. This area includes Razor [78], critical-path replicas [79], or canary flip-flops [80]. On the other hand, the latter intends to re-design the circuit such that the timing margin is manipulated to enhance voltage scaling efficiency. A state-of-the-art technique in this field is an activation-aware slack assignment (ASA), which associates the timing criticality of a path with its topological path delay and activity. A very recent research [81, 82] proposed an ASA method that allocates the timing margin with a stochastic mean-time-to-failure (MTTF) analysis. The timing errors are characterized by statistical static timing analysis and path activation analysis. This method accepts very few yet certain timing errors, and hence voltage can be aggressively scaled down.

Figs. 1.11(a) and 1.11(b) illustrates the concept of ASA, which originates from an earlier technique so-called critical path isolation (CPI) [82, 83, 84, 85]. ASA manipulates a synthesized circuit for the active paths by buffer insertion and cell swapping to allocate timing margin during an engineering change order (ECO) phase. After ASA, the active paths have more timing margin so that VS is applicable without timing error occurrence. Although the manipulation might increase the area due to inserted buffers and larger-size gates, the extra timing margin enables us to apply voltage scaling.

AVS and ASA are highly feasible to achieve low power/energy regardless of the type of design, and the general processor level design (CPU/GPU) or dedicated ASIC all can adopt these methods. Besides, the two techniques can even integrate together to perform a synergy effect. Reference [86] claims that thanks to ASA, the number of timing critical paths becomes less, and hence the paths need to be inserted with monitoring sensors can be effectively reduced, which can achieve significant area overhead reduction while sustaining the circuit functionality.

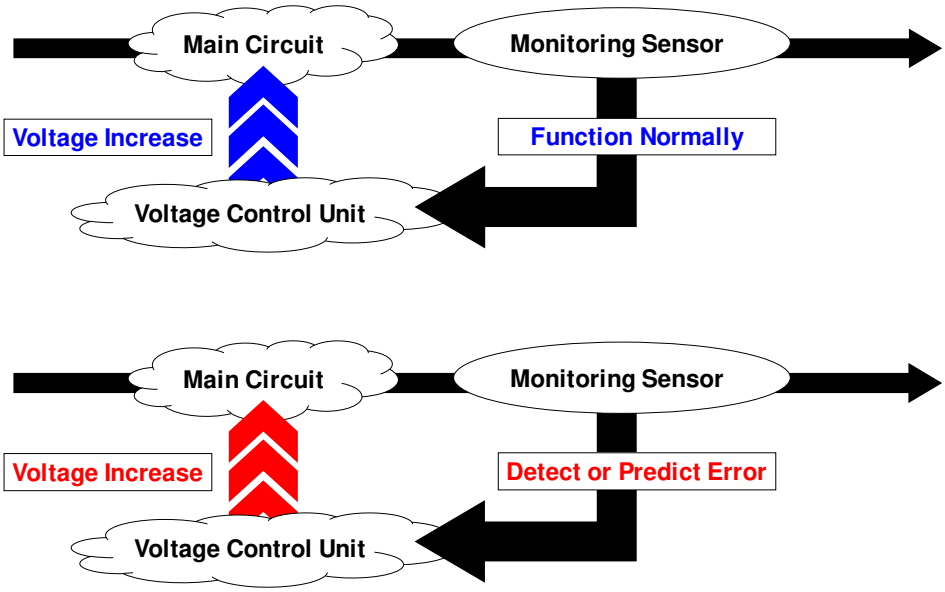
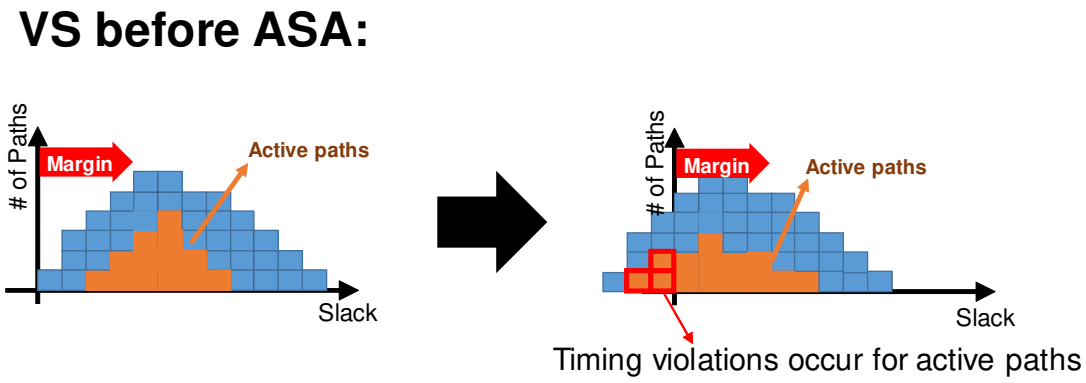
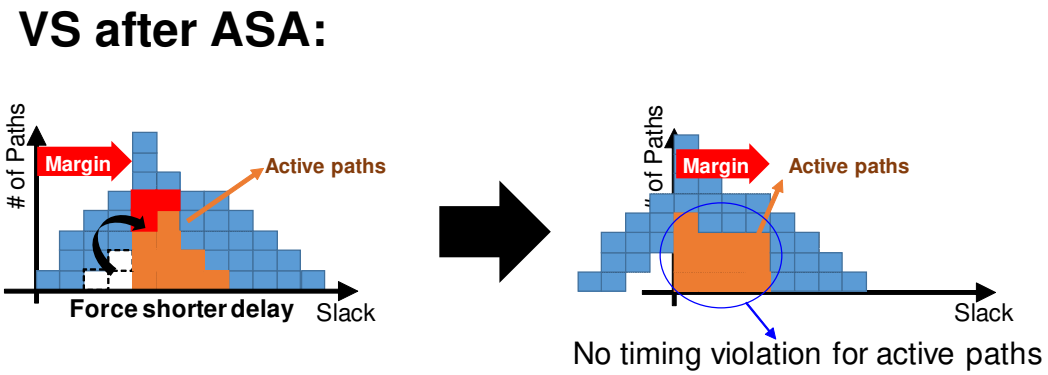


Figure 1.10: Schematic of the AVS techniques.



(a)



(b)

Figure 1.11: Concept schematic for applying ASA (a) VS before ASA and (b) VS after ASA.

1.6 Challenge for Realizing Efficient Training Engine

This section discusses the challenges for achieving power/energy efficiency for the training engine and the remaining issues based on AC and VS to study in this work.

1.6.1 AC to Achieve Computation Reduction

Efficient training is even more challenging than inference since accuracy is always the baseline that cannot be compromised. Training engines need to perform more arithmetic computation with a wider dynamic range since gradients, which is numerically computed, and used to guide the weight update, spreads in a broadband dynamic range. As mentioned in Section 1.4, though many AC techniques are widely adopted by NN or even DNN, they are mainly for inference engine [2, 7, 58, 62, 64, 65, 66, 67, 87, 88].

As for training, since deeper or more sophisticated NN model is highly sensitive to approximation [55, 68], the AC techniques for training attempt to be more smooth and moderate, such as BWS [33, 34, 35, 60]. Besides, BWS can also contribute to memory-saving with fewer bits of storage, which encourages researchers to explore it. Also, training in many tasks is basically a one-time effort, and then the applicability of AC has not been seriously investigated. Meanwhile, considering that floating-point representation can cover wide dynamic range and are widely adopted in mainstream systems like GPU, the appropriate technique targeting on floating-point is reasonable. As mentioned in Section 1.2, NN algorithm consumes heavy MAC computations, and even more during the training phase, and therefore, the mitigation of MAC computation should be quite effective for efficiency improvement. MAC computation involves multiplication (multiplier) and accumulation (adder) operations, where it is known that multiplier is a very power-hungry and area-expensive unit compared with adder. Fig. 1.12(a) and Fig. 1.12(b) exemplify the power and area benchmarking between a FP32 multiplier and an adder synthesized for the same clock frequency. The figures show that the floating-point multiplier consumes 3.0X power and 1.8X area. The MAC computation roughly consists of the same number of operations for multiplication and addition, indicating that the power consumed by the multipliers is the majority when performing MAC computation. As massive MAC computations are conducted in NN, it is reasonable to tackle the power for floating-point multipliers primarily to realize MAC computation mitigation.

1.6.2 DNN Training with FP16

Besides the accuracy, the conventional back-propagation algorithm in training temporarily generates many parameters for gradient and updated weights computation, which cost the resource for memory storage seriously. For effectively achieving computation and memory resource reduction while with smooth quality degradation, BWS is

(a) (b)

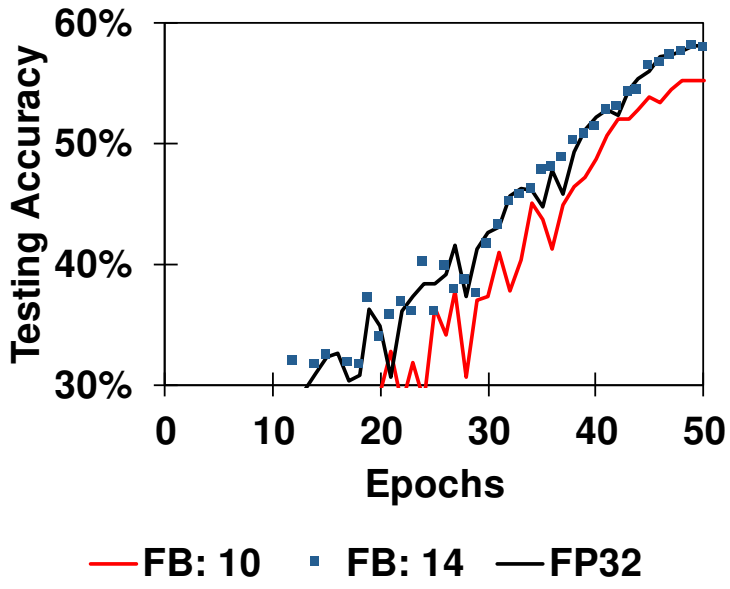
Figure 1.12: Hardware benchmarking for 32-bit floating-point multiplier and adder synthesized with Nangate 45nm cell library for the same clock frequency. Both (a) power and (b) area values are normalized by those of the adder, respectively.

widely used in modern DNN training. Recently FP16 shows its potential to accomplish training acceleration since it improves computation, throughput, mitigates necessary memory bandwidth and reduces power consumption [33, 34, 35]. However, there is a concern that the FP16 format may not have enough representation capability necessary for modern DNN training, especially in the case of large-scale datasets or complicated applications. Take two examples for examining FP16 training quality. Fig. 1.13(a) shows the training curves for the image classification of ImageNet and Fig. 1.13(b) is for the object detection of Pascal VOC. The implementation detail will be explained later in Chapter 4. In the figures, FB is an abbreviation of fraction bit-width used in floating-point format, where FB = 23 in FP32 format and 10 in FP16. Here, this evaluation aligns the bit-width of the exponent to 8 to sustain the dynamic range, and thus, the training quality entirely depends on FB. This example shows that 10-bit FB has a significant gap compared with FP32, and 14-bit FB and 12-bit FB are necessary for ImageNet and Pascal VOC, respectively, to reach the same quality (accuracy or mAP) with the FP32 case.

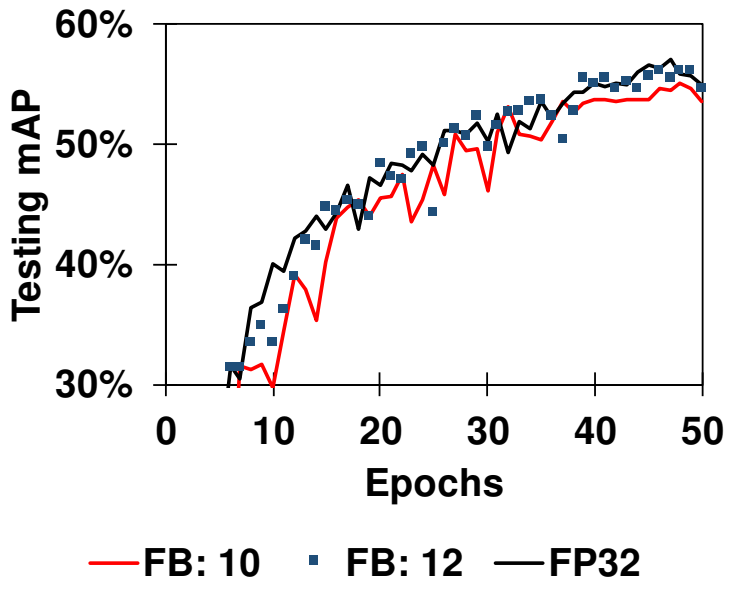
For certain public datasets, state-of-the-art researches are dedicated to achieving training with FP16 while sustaining the quality as FP32. Such sophisticated strategies or techniques like mixed-precision training [33, 34], chunk-based accumulation [35], or stochastic rounding [35] enable those datasets to be trained with FP16. However, it is not sure whether FP16 can always guarantee the training quality to be comparable with FP32 one when tackling a new real-world dataset.

1.6.3 ASA to Increase Timing Margin

Beyond the AC techniques for NN, current state-of-the-art ASA techniques also remain space for further discussion. Masuda *et al.*[81, 82, 86] first extract a set of active FFs for



(a)



(b)

Figure 1.13: FP16 precision (FB: 10) cannot guarantee its training quality to be comparable with FP32 one for (a) ImageNet (image classification) and (b) Pascal VOC (object detection).

the given pre-determined workloads. Then, the FF-based ASA is performed. First, they

insert dummy delay cells to the extracted active FFs, and then move out the dummy cells after ECO under the same clock period. As a result, the paths ending at the manipulated FFs increase their slack by the amount of dummy cell delays. For determining the necessary timing margin, timing failure probability is introduced and defined as a stochastic joint probability (P_{ERR}) by a flop activation (P_{ACT}) and its structural timing violation probabilities (P_{VIO}). Then, aggressive voltage scaling, which reaches 25% voltage reduction in their experiments, is applied as long as the P_{ERR} defined at the scaled voltage for each selected flop is within a given target value, as shown in Fig. 1.14. Here, nonzero positive value, and hence the circuit causes timing error at a certain probability while it is very small.

ASA steps forward to an industry-friendly style by Nagayama et al. [76]. This work aims to lower the supply voltage for a power-hungry operation mode (workload) to reduce the peak power. It expands the timing margins of all the active paths in the mode of interest and achieves visible VS of 50 mV in the mode in their design experiment with an industrial design [76], where this mode-dependent VS is called mode-wise voltage scaling (MWVS). Here, the term "mode" could be also thought as different workloads. For multi-function circuit designs. e.g,. GPUs, CPU, or many kinds of ASICs, different modes might activate different circuit paths in a time. There is a fact that the temporary usage duration could vary with individual modes, and also the activated circuit paths have different achievable timing margin. The existing temporal bias of the mode usage and achievable timing margin could be exploited to contribute to energy saving after ASA. When applying image processing or NN algorithm, most of the arithmetic operation relies on FPU. On the other hand, the encryption or decryption process does not need FPU but require fixed-point unit (or integer-point unit) only. There is a fact that FPUs conventionally demand longer latency then integer unit, and thus the potential of VS after ASA are smaller than fixed-point unit as well. Therefore, instead of applying a single-mode ASA for all the modes (involving image processing and encryption) to only allow the identical supplied voltage, if ASA is applied individually mode by mode, each mode can be operated at its own minimum voltage. According to their implementation, a timing error is supposed not to occur as long as all the active paths in the circuit fulfill the timing closure. However, this doctoral work reproduced their work and found that the timing error occurred at the scaled voltage of interest because of unexpected glitch events occurring in non-active paths. Thus, their ASA implementation is still risky for timing error issues.

The stochastic treatment of timing error limits application domains to those that tolerate timing error, such as image processing and machine learning [39, 89]. Though NN may accommodate to computation error, the MTTF-based ASA is very hard to be quantified for its impact on training or inference since there is no effective way to validate its functional correctness in a reasonable time especially when things come to an aging issue. Besides, not restricted to NN, the industrial designs may not tolerate any timing error even in those domains since it induces inconsistency with the current pro-

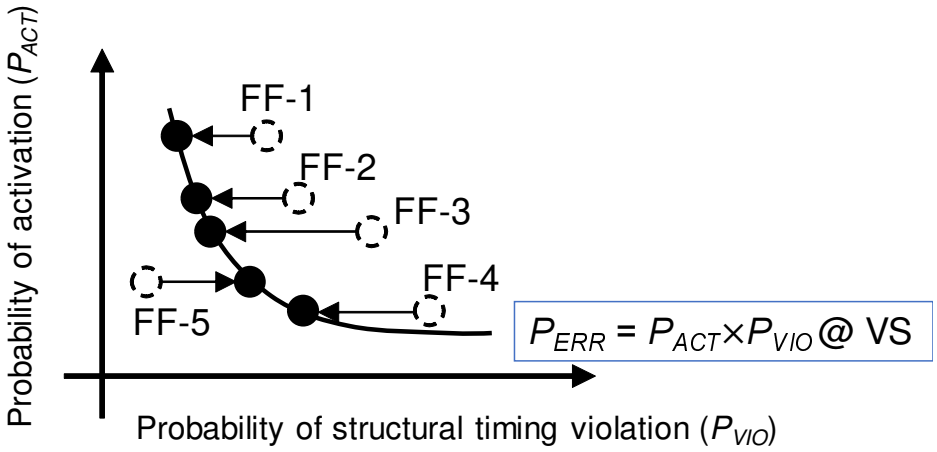


Figure 1.14: Timing margin for each FF is determined such that P_{ERR} is constraint while individual P_{ACT} values are different.

duction test policy. On the other hand, without MTTF treatment, pure ASA may attain a very limited margin for voltage scaling since the paths whose delays are squeezable by gates or structural manipulation through ECO may not be many. Therefore, how to enlarge the power/energy gain through ASA with guaranteeing no timing error becomes an interesting and attractive problem.

1.7 Objective of this Dissertation

The final goal of this dissertation is to provide a design methodology to achieve energy-efficient training that can facilitate both **high-end GPU designs** and **server- or device-level designs**. To achieve this goal, this dissertation chooses two countermeasures from the power/energy reduction approaches mentioned in Section 1.3, which are (1) approximate computing (AC) and (2) voltage scaling (VS) techniques. The former aims at mitigating the computation, especially the primary MAC computation to improve power/energy efficiency, while the latter intends to re-allocate the timing margin for critical paths, resulting in more voltage decrements and resultant power/energy reduction. This dissertation addresses the unexplored combinations and the disadvantage of the existing works mentioned in Section 1.6, such as insufficient exploration for AC in DNN training, controversial results of training with FP16, and timing error risk from state-of-the-art ASA works, and provide effective solutions to resolve or improve those issues. Besides, this dissertation provides a methodology to combine the two main strategies to come out with more benefits for DNN training efficiency.

Fig. 1.15(a) illustrates the organization of this dissertation. There are three parts, and each part performs different evaluations for (1) approximate computing, (2) voltage

scaling, and (3) approximate computing + voltage scaling. Note that either (1) or (2) can already perform significant power/energy consumption reduction, but the synergy of two countermeasures (3) can provide further more power/energy efficiency enhancement.

For (1), Chapter 2 applies the logarithm-approximate multiplier (LAM) to NN training. LAM is an approximate multiplier specific for floating-point data expression [90]. By approximating a floating-point multiplication as a fixed-point addition, LAM can contribute to smaller delay, fewer gates, and lower power consumption than an accurate multiplier. The dedicated NN training hardware can enjoy the benefit by LAM with little accuracy loss. LAM also has high compatibility with conventional BWS. A NN training engine with BWS can further extend its acceleration and power/area reduction thanks to LAM. LAM shows its advantage not only in a dedicated training engine, but also in GPU level design. In this chapter, an experimental GPU design embedded with LAM executing an NN-training workload, which is implemented in an FPGA, presents significant power improvement, and the improvement gains further with applying LAM + BWS. The applicability of LAM and LAM + BWS in deeper NNs are also presented and qualified.

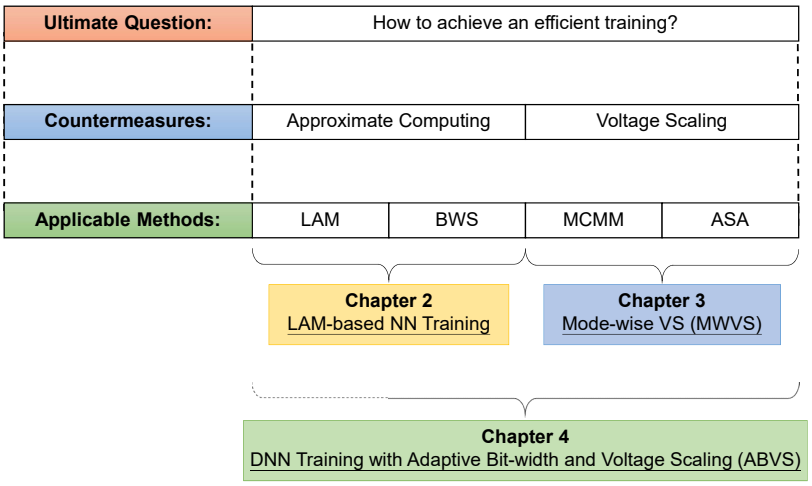
As for (2), Chapter 3 aims at addressing the concern of the current ASA approach utilized in [76, 81, 82, 86], and provide a design optimization methodology that achieves a design applying to mode-wise voltage scaling (MWVS) guaranteeing no timing error in each mode operation. This work formulates the MWVS design as an optimization problem that minimizes the overall power consumption considering each mode duration, achievable voltage lowering, and accompanied circuit overhead explicitly, and explores the solution space with the downhill simplex algorithm (DSA) that does not require numerical derivation and frequent objective function evaluations. For obtaining a solution, i.e., a design, in the optimization process, this work exploits the multi-corner multi-mode (MCMM) design flow in a commercial tool for performing mode-wise ASA with sets of false paths dedicated to individual modes. The test design applying the proposed design methodology proves higher energy efficiency thanks to MWVS compared with conventional voltage scaling and even the single-mode based approach utilized by preliminary works [76, 81, 82, 86]. MWVS is believed to be a promising way that maximizes the benefit of ASA, regardless of the type of included design constraint, e.g. even MTTF-aware constraint. This work also introduces a cycle-by-cycle fine-grained false path identification (CF-FPI) method to remarkably reduce the leakage power compared with the conventional way to determine FPI. Both the FPI and the proposed CF-FPI guarantee their timing correctness during voltage scaling.

Combining (1) and (2) to (3), Chapter 4 proposes an adaptive bit-width and voltage scaling (ABVS) scheme for DNN training assuming hardware with configurable fraction bit-width (FB) is available. The key idea is that starting from smaller FB, this work increases FB according to current training quality (e.g., accuracy or mAP). Considering that less FB can achieve shorter hardware latency, this training scheme can concurrently adapt bit-width and voltage scaling and intensify energy reduction. Experimental re-

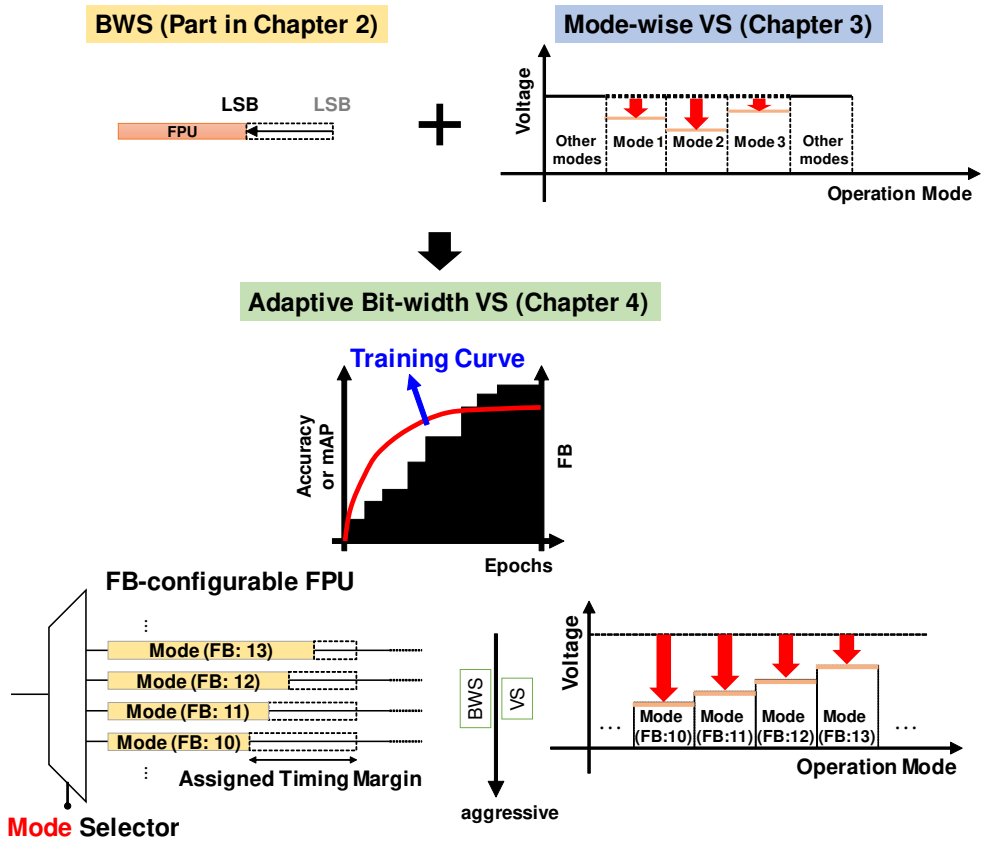
sults through the hardware evaluation show that training various popular datasets with the proposed ABVS flow can reach the same quality level as FP32, while significant energy reduction can be attained. Besides, a guideline for utilizing ABVS for a new dataset is also provided. This work further investigates different rounding methods and proves that the round-to-the-nearest-even (RTNE) method provides higher energy efficiency than round-to-zero (RT0) for attaining the same training quality. The proposed scheme is applicable to future GPU designs and dedicated training engines. Note that in Fig. 1.15(a), there is part of the big parentheses corresponded to Chapter 4 in the dashed line instead of a solid one. That means, theoretically, the proposed ABVS scheme can adopt LAM as well, but the validation platform exploits the existing GPU infrastructure due to the necessity of its high computing throughput for validation, where the GPU acceleration might be obstructed by LAM algorithm, and hence the implementation of LAM is omitted. As for a dedicated ASIC design, ABVS can simply work with LAM technique.

Fig.1.15(b) explains the connections between the chapters with the key ideas. The scaling of FB (bit-width scaling, BWS) is one of the key techniques discussed in Chapter 2, while the mode-wise VS scheme is the key contribution in Chapter 3. The key idea in Chapter 4 is to adapt the FB during the training. The implementation of this idea requires an efficient FB-configurable FPU design. Since the different FBs share the same concept of different modes, a configurable hardware satisfies the precondition of a circuit applying to mode-wise concept and therefore, we can enjoy the benefit from mode-wise VS as well. Training with ABVS is expected to serve as a promising solution for efficient training according to the fact that aggressive BWS and the resultant aggressive VS attain the superposition effect for the efficiency improvement.

The rest of this dissertation is organized as follows. Chapter 2 presents LAM's benefit in NN training efficiency and demonstrates the synergy effect with BWS on both dedicated ASIC design and GPU level processor. Chapter 3 provides the MWVS design methodology to maximize ASA advantage under MCMM co-design flow with achieving no timing error. Chapter 4 describes the ABVS scheme that can concurrently adjust the FB and VS together to achieve energy minimization for DNN training. Finally, Chapter 5 gives the overall concluding remarks of this dissertation.



(a)



(b)

Figure 1.15: (a) Overall organization of this dissertation. (b) Connections of key ideas between the chapters.

Chapter 2

Logarithm-Approximate-Multiplier-based (LAM-based) NN Training

This chapter proposes to adopt Logarithm-approximate Multiplier (LAM) in NN training to facilitate speed/power/area efficiency. With LAM, the primary MAC computation can be mitigated during the training, which contributes to significant performance efficiency improvement. In this work, a basic multilayer perceptron (MLP) architecture with 5 datasets is used as the testcases for experimental validation. The advantage of LAM to training engine are experimentally demonstrated through two platforms, where one is the dedicated training hardware while the other one is the GPU-level design.

2.1 Introduction

As mentioned in Chapter 1, DNN algorithm demands intensive computation, especially in training. Training involves forward and back propagation process. Through Eq. (1.1) used in forward propagation and Eqs. (1.2), (1.3), (1.4), (1.5) used in back-propagation, the primary computation used in training is MAC computation. In addition, the back propagation involves gradient computation (Eqs. (1.2) and (1.3)), which necessitates wide-dynamic range of the value. Thus, it is reasonable and quite common to apply floating-point during training, and hence GPU conventionally apply 32-bit floating point (FP32) for training. As mentioned in Chapter 1, the efficient training is desirable for both GPU design or dedicated ASIC design [16, 17, 18, 19] (edge terminal, mobile SOC) for AI service provider. Since one of the reasons for low efficiency in training is the huge computation resource, especially the primary MAC computations. To achieve efficient training, it is essential to mitigate the amount of primary MAC computation. MAC computations roughly demand equal usage of multiplication and addition. Referred to Fig. 1.12, it is known that floating-point multipliers demand 3X more power

than floating point adders, and therefore, a efficient scheme of floating-point multipliers would be really helpful to MAC computation mitigation.

This chapter demonstrates that the LAM, which approximates floating-point multiplication to fixed-point addition, benefits to NN training and improves the power efficiency of massive MAC computation involved in NN training under floating-point format. This work also shows that LAM is useful even when the BWS is already implemented in training, and hence power efficiency is further enhanced. These advantages are quantitatively evaluated through the experiments with dedicated training hardware (ASIC). This work then evaluates whether solo or hybrid usages of exact floating-point multiplier and LAM in training and testing phases affect the classification accuracy. Next, this work conducts additional experiments for evaluating the applicability of LAM and LAM + BWS to open-source GPU design, on which NN training programs are executed.

The contributions of this chapter are:

- **For dedicated training hardware:** Experimental results reveal that adopting LAM in training induces no significant accuracy degradation, and then there is no need to rely on accurate multipliers. Above 2.5X power reduction is achieved by LAM and 4.9X reduction by LAM + BWS while sustaining the accuracy, where 2.2X reduction originates from LAM. Moreover, up to 4-hidden layers, results show that the solo-LAM training achieves highly comparable results with solo-EFM training, where EFM means the accurate multiplier (exact floating-point multiplier). This trend sustains even when BWS is aggressively adopted as long as the acceptable training accuracy is obtained.

- **For GPU-level design:** Power reductions thanks to LAM and LAM + BWS are measured with an FPGA implementation of the GPU design, and they are 28% and 41% compared with the original design.

The rest section is organized as follows. Section 2.2 reviews the related work for LAM. Section 2.3 introduces LAM and discusses its approximate error. Experimental results of adopting LAM in dedicated training ASIC are presented in Section 2.4. Section 2.5 provides the environmental setup and measurement results of LAM-based NN training with FPGA implementation of an open-source GPU design. Section 2.6 applies LAM-based training to deeper NNs and provides evaluation results. Finally, Section2.7 concludes this chapter.

2.2 Previous work on Logarithm-based Multipliers

Floating-point training with BWS technique is classical [22, 33, 34, 35, 60, 61], but floating-point training with other AC techniques, such as logarithm-based multipliers, is relatively less studied. Table 2.1 briefly summarizes the existing works related to

Table 2.1: Existing works that adopt logarithm-based multipliers in NN.

Phase	Inference	Training
Fixed-point (linear-domain)	[36, 37]	[91]
Fixed-point (log-domain)	[59]	[36, 37]
Floating-point	[58, 88]	Proposed LAM-based training

the logarithm-based multipliers for NN. Logarithm based multiplier is a typical type of approximate multiplier since logarithm converts multiplication to addition. State-of-the-art logarithm multipliers, such as [56, 57]. adopt this property to approximate fixed-point multiplication in cooperation with a dedicated and efficient error-fixing solution to mitigate the calculation error compared with exact multiplications. Some researchers exploit the logarithm-based multiplier in NN. Reference [91] applies iterative logarithmic multipliers, which is a preliminary version of [56, 57], to error-tolerant training algorithm while [36, 37, 59] convert the multiplicand and multiplier into log domain to do multiplication as an addition. References [36, 37] perform even addition and activation function in log domain to execute the complete training in log domain. The above studies are all based on fixed-point representations of the original value and its log value. Recent works [58, 88] propose to adopt logarithm-based floating-point multipliers in NN, but [88] only adopts it in an inference engine. Although [58] uses the logarithm-based multiplier in training, their focus is to provide a run-time configurable solution that can switch exact and logarithm-based multipliers. Once an error that is larger than a pre-determined criterion is detected in the logarithm-based multipliers, the architecture automatically switches back to the exact multiplier. The same first author of [58] has an earlier publication that introduced two-stage training, in which the approximate training is allowed in the early stage while in the late stage, the accurate training is demanded [68]. Consequently, [58, 68] still rely on the exact multiplier in training and solo logarithm-based training is beyond their interest. Besides, the compatibility with BWS and the efficacy in deeper NNs are not addressed.

In summary, previous studies for floating-point NN training intensively focus on BWS, and it has left the space for evaluating the efficacy of the logarithm-based multiplier in training. Also, BWS is still the primary choice in the training engine design, and hence the compatibility between the logarithm-based multiplier and BWS must be investigated. Taking into account the tremendous number of MAC operations in training engines, exploring a useful approximate technique for pursuing higher power efficiency and examining its compatibility with BWS could give a useful implication for training-engine designers, which is the objective of this work. Besides, in addition to realizing a training engine through ASIC, GPU-level applicability for LAM is also one of this work's interests.

2.3 Introduction of Logarithm-approximate Multiplier (LAM)

Logarithm-approximate multiplier, LAM in short, is developed by [90]. With an approximation, a floating-point value in linear domain can be regarded as its value taken by the logarithm of base 2 in fixed-point format. Thanks to the log-domain property, floating-point multiplication can be approximated to fixed-point addition. This section introduces LAM and analyzes its approximation error.

2.3.1 Floating-point Multiplication

The floating-point format consists of three parts to represent a value in scientific notation: one bit for sign, several bits for exponent, and the remaining bits for fraction. When a value i is represented in the floating-point format containing N bits for exponent and M bits for fraction, then i is expressed as:

$$i = (-1)^{S_i} \cdot 2^{(E_i - \text{bias})} \cdot (1 + F_i/2^M). \tag{2.1}$$

S_i is 0 or 1, where 0/1 means i is a positive/negative value. F_i is the fraction part when i is converted to base-2 scientific notation with multiplying 2^M, and hence $0 \leq F_i/2^M \leq 1$. E_i is the exponent value that includes a bias $= 2^{(N-1)} - 1$.

Fig. 2.1 explains the multiplication of two floating-point values $C = A \times B$, where the sign parts (S_A, S_B and S_C), exponential parts (E_A, E_B and E_C) and fraction parts (F_A, F_B and F_C) in the floating-point representation are processed individually.

$$S_\oplus = S_A \oplus S_B, \tag{2.2}$$

$$E_+ = E_A + E_B, \tag{2.3}$$

$$F_\times/2^M = (1 + F_A/2^M) \times (1 + F_B/2^M), \tag{2.4}$$

where $(1 + F_A/2^M)$ is obtained by appending 1 to the binary representation of F_A, and supposing the binary point exists between 1 and F_A. Then, the multiplication result is expressed by:

$$S_C = S_\oplus, \tag{2.5}$$

$$E_C = \begin{cases} E_+ - \text{bias} & F_\times/2^M < 2, \\ E_+ - \text{bias} + 1 & \text{otherwise}, \end{cases} \tag{2.6}$$

$$F_C = \begin{cases} F_\times - 2^M & F_\times/2^M < 2, \\ F_\times/2 - 2^M & \text{otherwise}. \end{cases} \tag{2.7}$$

In the hardware point of view, the multiplier for $F_\times$ consumes considerable power and area, and it often limits the speed since it contains many full adders or similar logics in both width and depth.

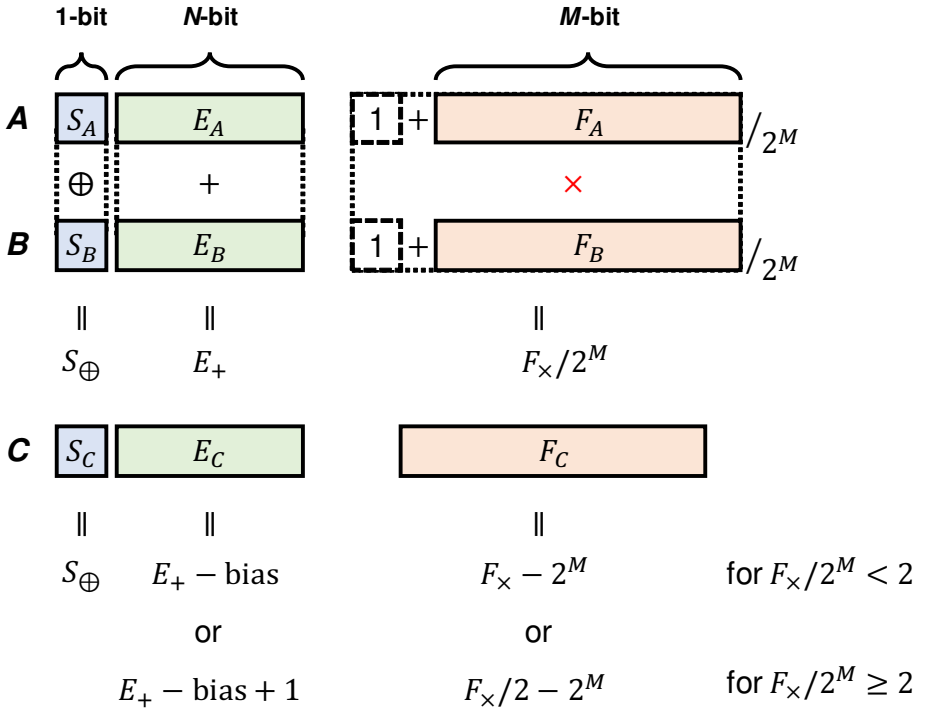


Figure 2.1: Operation of exact floating-point multiplier. ($S_\otimes$,E_+,$F_\times$) are individually calculated by XOR, addition, and multiplication based on the sign, exponent, and fraction part of multiplicand and multiplier, respectively. Then, E_C and F_C are the conditional formula according to $F_\times$.

2.3.2 Logarithm-approximate Multiplier

Focusing on a positive floating-point number i, Eq. (2.1) is simplified for the sake of clarity in the following discussion.

$$i = 2^e(1 + f), \tag{2.8}$$

where e replaces $E_i - \text{bias}$ and f replaces $F_i/2^M$. When converting i into log domain, Eq. (2.8) becomes

$$\log_2 i = e + \log_2(1 + f) \cong e + f, \tag{2.9}$$

where the approximated representation in the right term utilizes the approximation below.

$$\log_2(1 + x) \cong x, \text{ for } 0 \leq x \leq 1. \tag{2.10}$$

When approximating $\log_2(1+x)$ at $x = 0$, Eq. (2.10) becomes $1.44x$. On the other hand, when intending to approximate $\log_2(1+x)$ in the region of $0 \leq x \leq 1$, the approximation to $1.0x$ is also possible as shown in Fig. 2.2. With this approximation, Eqs. (2.8) and

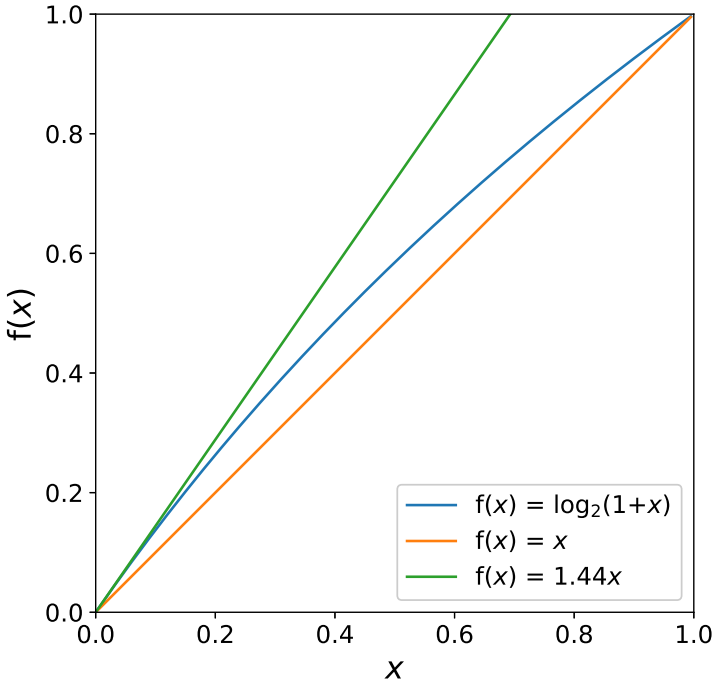


Figure 2.2: Curves of functions $\log_2(1+x)$, $1.0x$, and $1.44x$. Taking into the entire range of $0 \leq x \leq 1$, $1.0x$ is a possible approximation of $\log_2(1+x)$, while $1.44x$ is better at the point of $x = 0$.

(2.9) make any manipulation unnecessary to approximate a floating-point value i to a fixed-point value $\log_2 i$.

The logarithmic domain is beneficial in multiplication, as mentioned in Section 2.2, since it can convert multiplication to addition. Fig. 2.3 illustrates the approximate multiplier that is named as logarithm-approximate multiplier (LAM). To compute $C = A \times B$, according to LAM, the following primary computations are performed.

$$S_{\oplus} = S_A \oplus S_B, \tag{2.11}$$
$$E_{+} = E_A + E_B, \tag{2.12}$$
$$F_{+}/2^M = F_A/2^M + F_B/2^M, \tag{2.13}$$

where it can be found the calculation of fraction parts has changed from multiplication to addition thanks to the property of log-domain while Eqs. (2.11) and (2.12) are still identical to Eqs. (2.2) and (2.3). Making a mantissa from a fraction is also excluded, and the fraction part is directly used for computation. Then, the final multiplication result is

expressed by:

$$S_C = S_{\oplus}, \tag{2.14}$$

$$E_C = \begin{cases} E_+ - \text{bias} & F_+/2^M < 1, \\ E_+ - \text{bias} + 1 & \text{otherwise}, \end{cases} \tag{2.15}$$

$$F_C = \begin{cases} F_+ & F_+/2^M < 1, \\ F_+ - 2^M & \text{otherwise}. \end{cases} \tag{2.16}$$

Note that Eq. (2.14) to calculate sign part S_C is identical to Eq. (2.5) and totally isolated from computing E_C and F_C. Therefore, LAM can perform multiplication irrelevantly to positive and negative values using the equations from Eq. (2.11) to Eq. (2.16). In this work, BWS is achieved by varying the variable M denoted in Fig. 2.3, which can be easily integrated in both exact multiplication and LAM.

Although E_C and F_C in Eqs. (2.15) and (2.16) are conditional equations, they can be efficiently computed in hardware implementations. These terms $\{E_A, F_A\}$ and $\{E_B, F_B\}$ are concatenated, respectively, then added and subtracted the bias term followed by M-bits of 0s, as illustrated in Fig. 2.4. Then, the overflow coming from $F_A + F_B$ can directly add a carry to $E_A + E_B$. Finally, E_C and F_C are exactly the first N bits and the last M bits of the computed result.

The approximation error of LAM can be analyzed by directly comparing the computations of exact multiplication and LAM. Again, for the sake of clarity, let us just focus on two positive floating-point values A and B of $A = 2^{e_A}(1 + f_A)$ and $B = 2^{e_B}(1 + f_B)$, where the notations of e_A, e_B, f_A, and f_B refer to Eq. (2.8). The result of exact multiplication $C = A \times B$ is:

$$C_{exact} = 2^{e_A+e_B}(1 + f_A)(1 + f_B). \tag{2.17}$$

When computing $A \times B$ by LAM, the expression is:

$$C_{LAM} = \begin{cases} 2^{e_A+e_B}(1 + f_A + f_B) & f_A + f_B < 1, \\ 2^{e_A+e_B+1}(f_A + f_B) & \text{otherwise}. \end{cases} \tag{2.18}$$

By comparing the expressions under different conditions, relative error of approximation, $ErrLAM$, can be derived as a conditional function of f_A and f_B:

$$ErrLAM = \frac{C_{exact} - C_{LAM}}{C_{exact}} = \frac{Err}{(1 + f_A)(1 + f_B)} \tag{2.19}$$

where

$$Err = \begin{cases} f_A f_B & f_A + f_B < 1, \\ (1 - f_A)(1 - f_B), & \text{otherwise}. \end{cases} \tag{2.20}$$

Here, both f_A and f_B are between 0 and 1, and hence $ErrLAM$ is always above 0. Fig. 2.5 shows a contour map of $ErrLAM$ as a function of f_A and f_B. The maximum value of

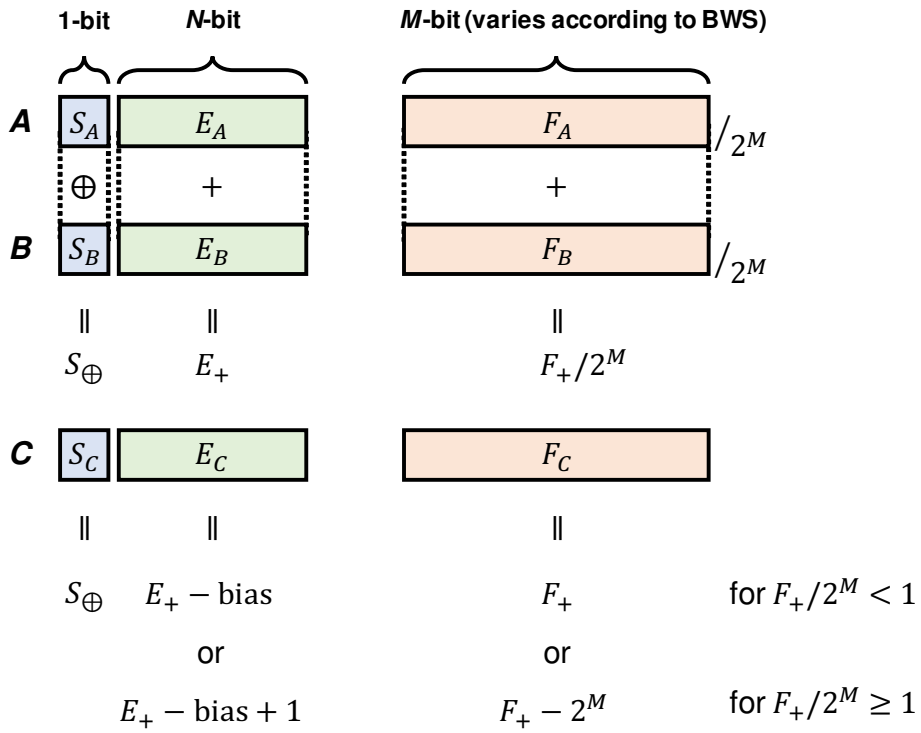


Figure 2.3: Operation of logarithm-approximate multiplier (LAM). $S_\otimes$ and E_+ are identical to those of the exact floating-point multiplier, but F_+ is directly computed by adding the fractions without making their mantissas. E_C and F_C are expressed by the conditional formula regarding F_+.

$ErrLAM$ is about 11.1% when both f_A and f_B equal to 0.5. Note that $ErrLAM$ is not affected by the exponents of A and B since e_A and e_B are all canceled out when dividing Eq. (2.18) by Eq. (2.17). Similarly, the sign values do not change the absolute value of $ErrLAM$ neither. Fig. 2.5 also indicates that, as long as either f_A or f_B is closed to 0 or 1, the approximate error is well suppressed. The advantage of LAM in terms of the speed, power, and area will be discussed in Section 2.4.1, and the impact of the approximation error on the NN training will be investigated in Sections 2.4.3 and 2.4.4.

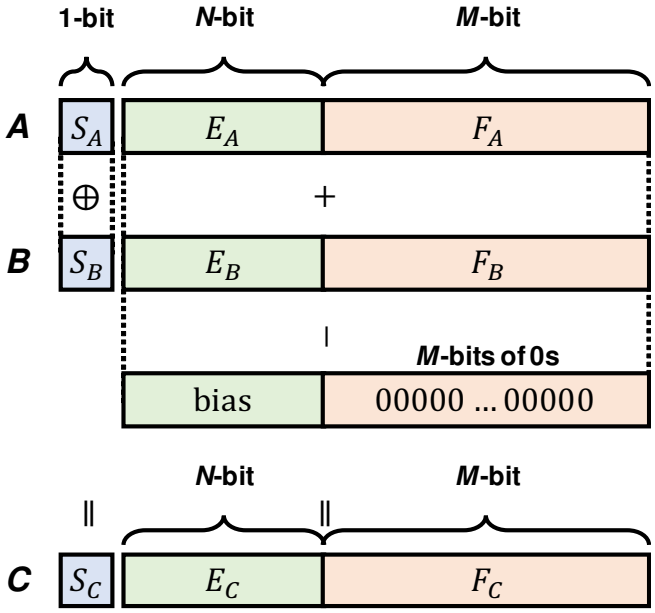


Figure 2.4: Algorithm for implementing LAM in hardware. E_A and F_A are concatenated, and E_B and F_B as well. Then, directly sum up them and subtract the bias term that is followed by M-bits of 0s to compute E_C and F_C.

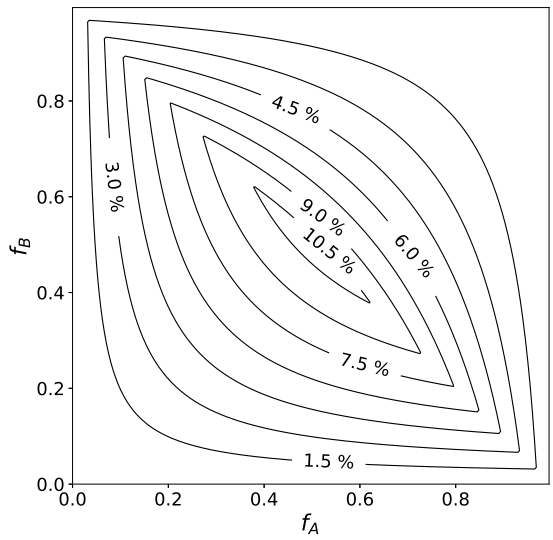


Figure 2.5: Contour plot of $ErrLAM$, which represents the relative approximation error between LAM and exact floating-point multiplier. The error depends on the fraction values (f_A, f_B) of the multiplicand A and multiplier B under base-2 scientific notation, where $0 \leq f_A, f_B < 1$.

Table 2.2: NN structures for testcases based on 1-hidden layer.

Dataset	#Neurons (I,H,O)	batch size
FOURCLASS	(2,8,2)	100
MNIST	(400,300,10)	100
HARS	(561,40,6)	40
ISOLET	(617,100,26)	60
CNAE-9	(856,100,9)	40

2.4 Evaluation for Dedicated Hardware Design

This section shows the advantage of LAM as an arithmetic unit and demonstrates the impact of LAM in NN training engine on the classification accuracy and hardware resource.

2.4.1 LAM Performance

The experiment first evaluates the performance of LAM as a multiplier by comparing two multipliers; one is LAM, and another is the baseline exact floating-point multiplier, denoted as EFM. LAM is manually implemented at RTL with Verilog while EFM directly adopts Synopsys DesignWare IP (DW_fp_mult) for functional credibility and sophisticated quality. The two designs are synthesized by Synopsys Design Compiler with an open-source 45nm Nangate cell library [92]. The experiment examines their speed, power, and area during benchmarking. The power and area evaluation are performed in two scenarios, one is at their individual highest speed and the other one is at a uniform speed at which the slowest multiplier can be synthesized.

The evaluation results are shown in Fig. 2.6. For the sake of clarity, all the results are normalized by that of 32-bit EFM. Fig. 2.6 shows LAM achieves 2.5X speed compared with EFM and consumes 5.9X less power and 8.3X less area at that speed while 12.5X less power and 7.7X less area at the uniform speed in case of 32-bit floating-point expression (where sign-exponent-fraction = 1-8-23). The results of 16-bit version (1-5-10) are also presented. An interesting observation is that 32-bit LAM operates faster and consumes less power and area than even 16-bit EFM. Thus, LAM can improve energy efficiency remarkably. The following sections evaluate the impact of LAM on NN training with BWS in terms of the NN classification accuracy and hardware performance.

2.4.2 Experimental Setup for NN Training

Table 2.2 lists the datasets [93, 23, 94, 95, 96] used for the experiments in this paper and the numbers of neurons in the NNs for each dataset. For the experiments, a 3-layer structure is prepared, which means 1 hidden layer (each layer is denoted as I,

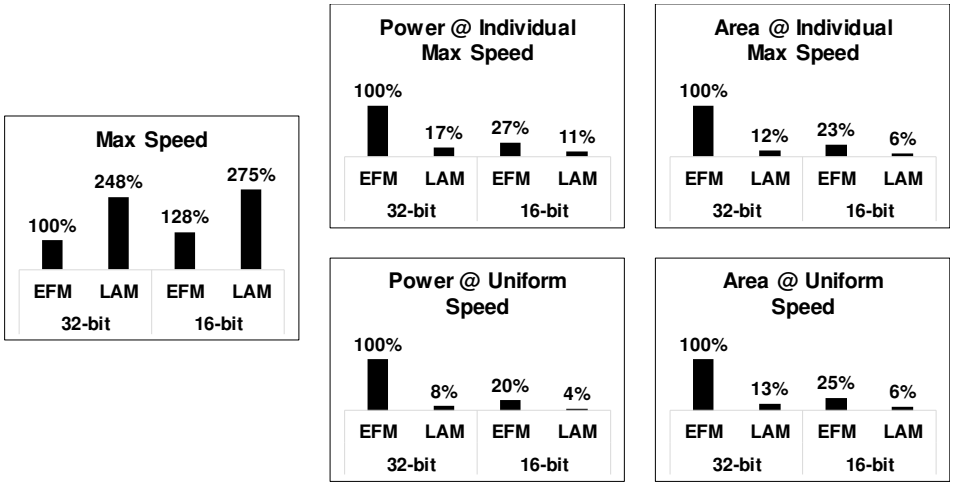


Figure 2.6: Speed, power, and area benchmarking for synthesized LAM and EFM. There is one speed comparison and two scenarios for power and area under the max speed circuit or the uniform speed circuit. All the values are normalized to EFM 32-bit case.

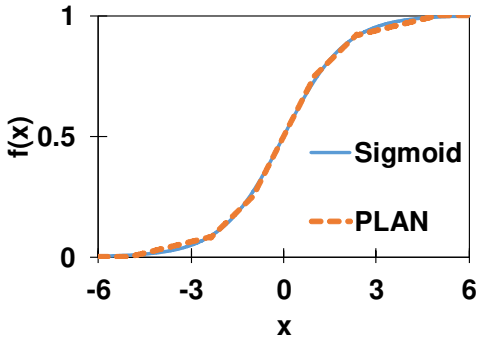
H, O), and adopt ReLU (Rectified Linear Unit, $y = \max(x, 0)$) and Sigmoid function ($y = 1/(1+\exp(-x))$) as the activation function of hidden and output layers, respectively. Cross entropy is chosen as the loss function since it can combine with the Sigmoid function to simplify Eq. (1.4) to a simple linear relation, $\delta_a^i = O_a - T_a$ [97]. This work also adopts stochastic gradient decent with mini batch, which updates weights and bias after accumulating $\partial\text{Loss}/\partial W$ and $\partial\text{Loss}/\partial B$ from a batch size of training data, and apply learning rate decay, which gradually declines learning rate along with iterations, as well for better convergent speed during the training.

Besides, the hardware implementation of non-linear Sigmoid function is costly, and hence the Piecewise Linear Approximation of a Nonlinear function (PLAN) model is selected to approximate Sigmoid which is proposed in [98]. PLAN function requires only shifters and adders, and hence it is friendly to hardware implementation. Fig. 2.7(a) lists the conditional equations used for Sigmoid approximation, and Fig. 2.7(b) shows the curves of the original and PLAN Sigmoid functions. It can be observed that PLAN is well correlated with the original Sigmoid function.

Fig. 2.8 plots the diagram of the dedicated NN training hardware engine containing arithmetic units such as $\otimes$ for multipliers and $\oplus$ for adders. Forward- block computes Eq. (1.1), Back- block calculates Eqs. (1.4) and (1.5), and then Updating block computes Eqs. (1.2) and (1.3). Here a, b, and c denote the numbers of neurons in Layer I, H, O, ReLU' means the derivative function of ReLU and Reg is register. The subtractor and PLAN function are annotated as adders since their functionality is similar while ReLU is drawn by $\bigcirc$. In Updating block, the accumulators is used to add up the gradients δ for the number of batch size and then update the associated weights and biases based on

PLAN $f(x), f(-x) = 1 - f(x)$	
Criterion	**Formula**
$x \geq 5$	1
$2.375 \leq x < 5$	$x/32 + 0.84375$
$1 \leq x < 2.375$	$x/8 + 0.625$
$0 \leq x < 1$	$x/4 + 0.5$

(a)



(b)

Figure 2.7: PLAN function, an approximate form of Sigmoid function. (a) Expression of PLAN and (b) Plot of original Sigmoid and PLAN functions.

the accumulated gradients with multiplying a learning rate η.

2.4.3 Evaluation for FOURCLASS Dataset

Now, the experiment evaluates the NN training engine with FOURCLASS dataset [93], which is a simple 2-D classification problem and its training results can be graphically illustrated with the boundary line separating the two classified groups. Fig. 2.9 shows the training results of 32-bit and 16-bit floating-point cases, where the samples in the same group share the same color. Note that the training and testing phase are both carried out with solo EFM or solo LAM. Although the boundary lines of EFM and LAM show some discrepancy, both of them successfully distinguish the groups of samples. In 32-bit case, both EFM and LAM achieve 100% classification accuracy. In 16-bit case, which corresponds to BWS adoption, LAM experiences 0.4% accuracy drop while it is negligibly small. This result suggests LAM is compatible with BWS.

Fig. 2.10 shows the hardware evaluation results, to which a normalization to 32-bit EFM engine is applied in the same way as Section 2.4.1. The training engine adopting LAM gains 10% speed-up over the EFM engine, where this speed-up is smaller than Fig. 2.6 since the floating-point adder is now a speed-limiting module. In addition to the speed improvement, the maximum speed circuit achieves 2.1X (= 100% / 47%) power and 2.2X (= 100% / 45%) area efficiency enhancements. On the other hand, in the uniform speed circuit, 2.3X (= 100% / 43%) power and 2.3X (= 100% / 44%) area efficiency enhancements are attained by LAM. These results confirm that the multiplier is so power and space demanding that reducing its computational cost improves the power and area efficiency considerably. Furthermore, considering 16-bit LAM as a scenario that LAM and BWS are applied together, 32% speed-up, 5.9X power, and 5.3X area efficiency enhancements are attained in the maximum speed circuit, where

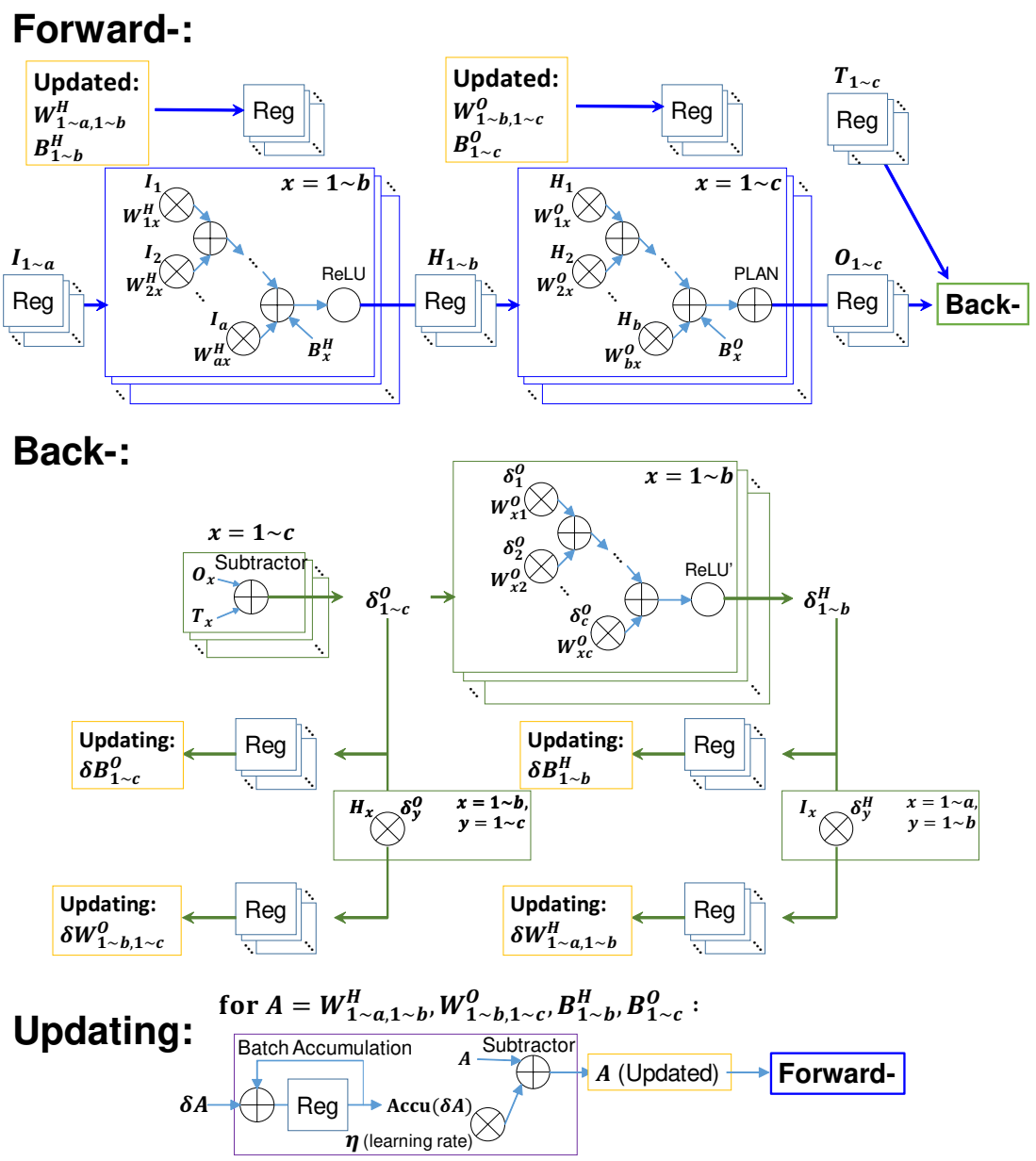


Figure 2.8: Diagram of the dedicated NN training hardware implemented in this work, including forward-, back-, and updating blocks. ⊗ and ⊕ denote multipliers and adders, respectively. All the ⊗ and ⊕ are spatially implemented at the same time.

5% (=132% − 127%) speed-up, 2.5X (=100%/17% − 100%/30%) power reduction, and also 2.1X (=100%/19% − 100%/32%) area savings originate from LAM. The benefits of training with LAM are quantitatively clarified.

Fig. 2.11 further plots the benchmarking results based on accuracy vs power con-

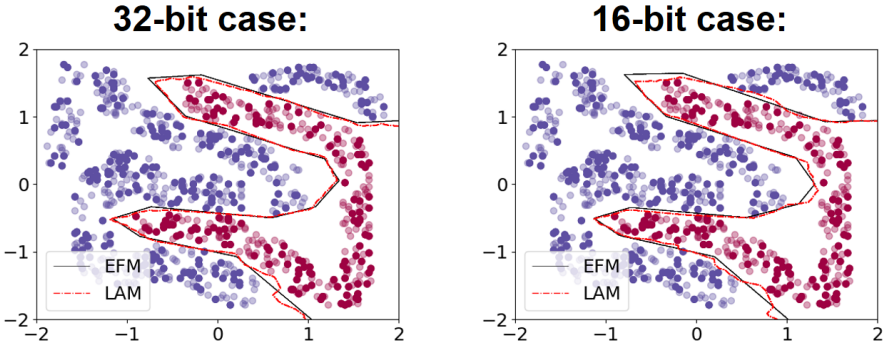


Figure 2.9: Boundaries trained for 2-D classification FOURCLASS.

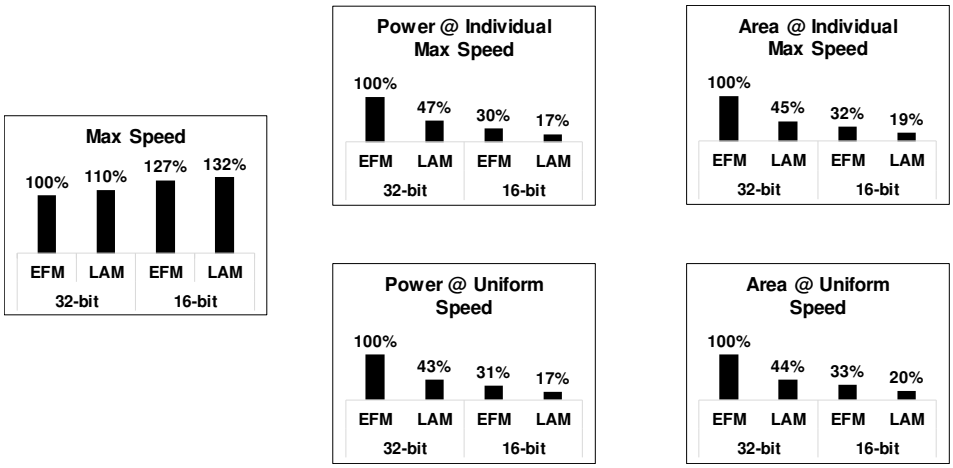


Figure 2.10: Speed, power, and area comparisons between synthesized LAM-based and EFM-based training engines. The synthesis setup is identical to that of Fig. 2.6, and all the values are normalized by those of EFM 32-bit case.

sumption for EFM-based and LAM-based training engines with different fraction bit-widths (FBs). Here, the exponent bits are fixed with 8 bits, aligning with the 32-bit format. From the results, as long as FB $\geq$ 10, the training results achieve 100% accuracy. In general, LAM-based training achieves at most 1.1% accuracy drop compared with EFM-based training at FB: 8 case, which is considered sufficiently accurate. The increasing FB strengthens the power saving for LAM-based training, and hence the case for the minimum power saving happens on FB: 6 case, which is 18%.

2.4.4 Evaluation for Higher Dimensional Datasets

The experiment further evaluates the effectiveness of LAM with four other higher-dimensional datasets. Also, to gain more insight into training methodology, this work

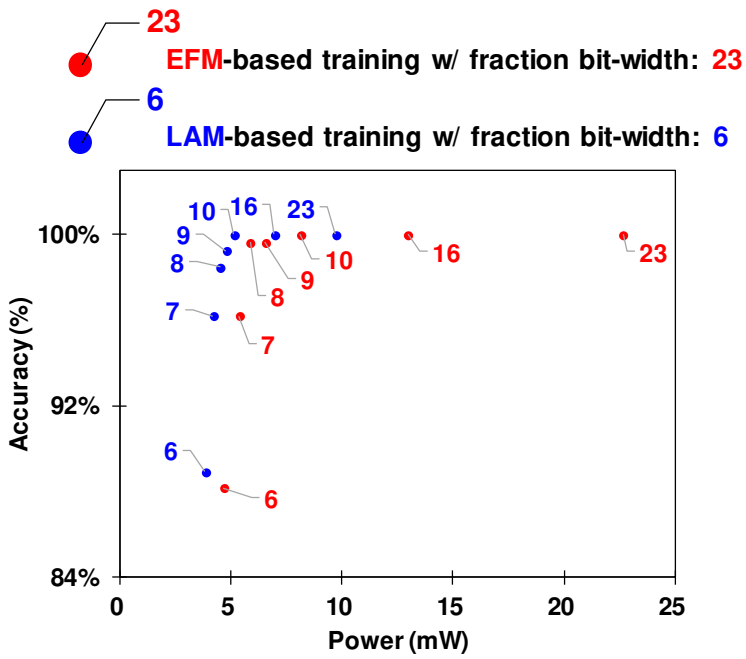
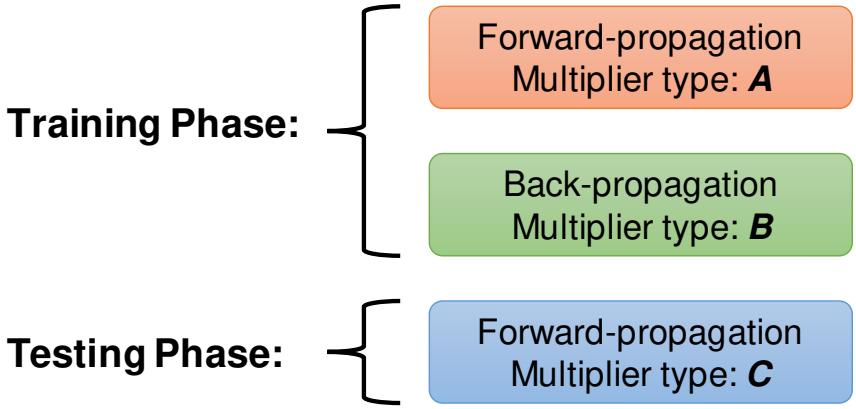


Figure 2.11: Accuracy vs power for synthesized LAM-based and EFM-based training engines, where the numbers marked on the points means the FBs. The synthesis setup is identical to that of Fig. 2.6.

compares approximation strategies that individually adopt accurate (EFM) and/or approximate (LAM) multipliers in training and testing phases, respectively. Fig. 2.12 illustrates the four strategies evaluated in this experiment.

Fig. 2.13 shows the training results for the four datasets. To test the compatibility of LAM to BWS, the experiment varied the number of fraction bits as 10, 16, and 23 while the bit width for exponent remains 8 bits to keep the dynamic range. Considering that different initial weights might have impact on final training results, this experiment is conducted with five sets of initial weights. The values summarized in the figure are the average accuracies for the five results, and then the ranges of the min-max difference are also marked on it. The results show that either case attains a similar accuracy both for training and testing sets. On the other hand, in different combinations of dataset and fraction bit-width, Cases #2 to #4 with LAM provide very close or even better classification accuracy compared with Case #1 only with EFM. Another observation is that the differences between Case #2 to Case #4 are also small. Overall, these results reveal that there is no reason to fully or partially use EFM in training, at least for the databases used in the experiment. LAM can provide trained NN models whose classification accuracy is comparable to that of those trained by EFM.

This experiment next evaluates the hardware cost. For estimating the hardware im-



Type	Case #1	Case #2	Case #3	Case #4
A	EFM	LAM	EFM	LAM
B				EFM
C			LAM	LAM

Figure 2.12: Different approximation strategies that use EFM and/or LAM in training and testing phases.

provement thanks to LAM, a simple projection is performed based on the FOURCLASS results. The power consumption of the training engine is estimated by the numbers of arithmetic units and registers. The power values of each arithmetic unit and register are obtained from the logic synthesis result. The remaining thing is to count the number of arithmetic units and registers referring to Fig. 2.8. Table 2.3 lists the statistics for the four datasets, indicating that the usage of three parts closed to 1:1:1.

As the number of multipliers, adders, and registers increase in higher-dimensional datasets, the power improvement rate by LAM for training a larger NN-size structure is roughly equal or slightly larger than that of FOURCLASS. Meanwhile, as a conservative estimate, the least improvement rate is accessed based on the FOURCLASS benefit. Table 2.4 shows the projected values. Here, the synthesized frequency are aligned, and in this particular estimation the power analysis is not annotated with actual switching activity (i.e. vectorless power). Terminology “EFM” is EFM-32bit, “LAM” represents LAM-32bit, “BWS” denotes EFM-19bit, and then “LAM + BWS” means LAM-19bit. Here, EFM-19bit and LAM-19bit adopt 8 bits for the exponent and 10 bits for the fraction, and they have the same fraction bit-width as conventional 16-bit format. From Table 2.4, LAM attains 2.5X and LAM + BWS achieves 4.9X power efficiency, where 2.2X originates from LAM.

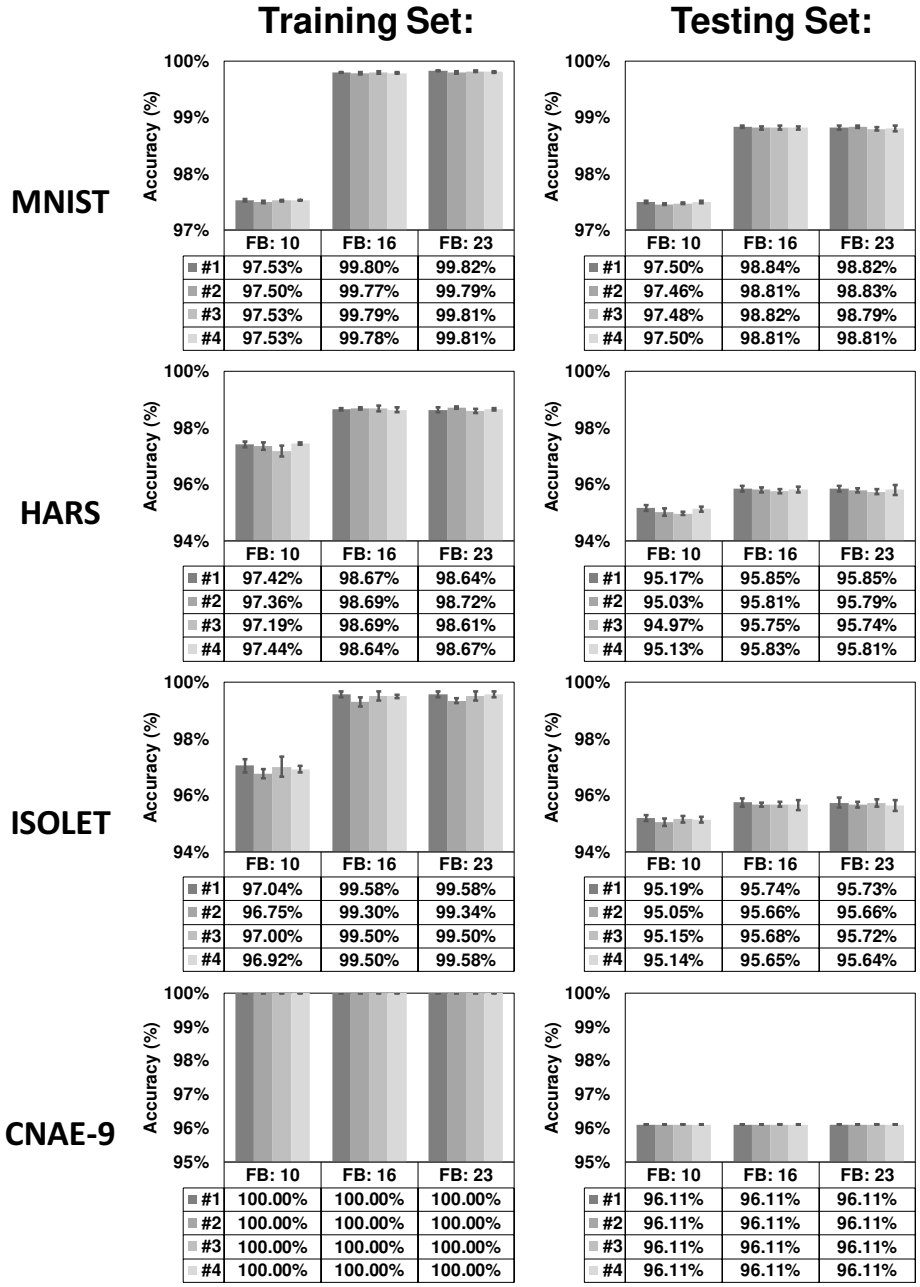


Figure 2.13: Training results for MNIST, HARS, ISOLET, CNAE-9 under different approximation strategies illustrated in Fig. 2.12 for the FB = 10, 16, and 23, with plotting the average accuracies and the ranges of min-max difference.

In this section, the power efficiency is evaluated based on estimation. Next section is going to demonstrate the power efficiency improvement through hardware measure-

Table 2.3: Statistics for usage of multipliers, adders, and registers in the dedicated NN hardware.

Dataset	Usage		
	#Multipliers	#Adders	#Registers
FOURCLASS	122	128	142
MNIST	372,310	372,340	371,050
HARS	68,326	68,344	69,352
ISOLET	195,626	195,704	194,664
CNAE-9	260,509	260,536	261,657

Table 2.4: Power estimation results for training larger-size of NN (projected from FOURCLASS benchmarking result).

Power (Normalized to EFM)			
EFM	LAM	BWS	LAM + BWS
100%	40%	37%	20%

ment.

2.5 Evaluation in GPU Design

Following the performance evaluation with dedicated training engines in the previous section, this section applies LAM and BWS to an open-source GPU design and clarifies the advantage in NN training.

2.5.1 LAM-based GPU Implementation on FPGA

To train large NNs, training engines demand programmability since the entire datapath cannot be implemented spatially in a chip at once and temporal sharing for, e.g., each layer and each kernel becomes indispensable. Then, this experiment selects Nyuzi, which is an open-source processor for GPGPU applications [99], as the baseline design and incorporate LAM and BWS with Nyuzi. Nyuzi is distributed as a synthesizable RTL Verilog with an instruction set emulator, and a C/C++ compiler.

Fig. 2.14 shows the power evaluation setup. To proceed with the experiment, the RTL code is modified to integrate LAM and BWS functionality. For performing power measurement on hardware, the original and modified RTL codes are synthesized by Intel Quartus with 50MHz clock frequency targeting Terasic DE2-115 evaluation board. Table 2.5 lists the logic element counts for each case after Quartus synthesis. Here, the bit-widths of the cases “EFM,” “LAM,” “BWS,” and “LAM+BWS” are aligned with those adopted in Table 2.4. From the table, the hardware design embedded with

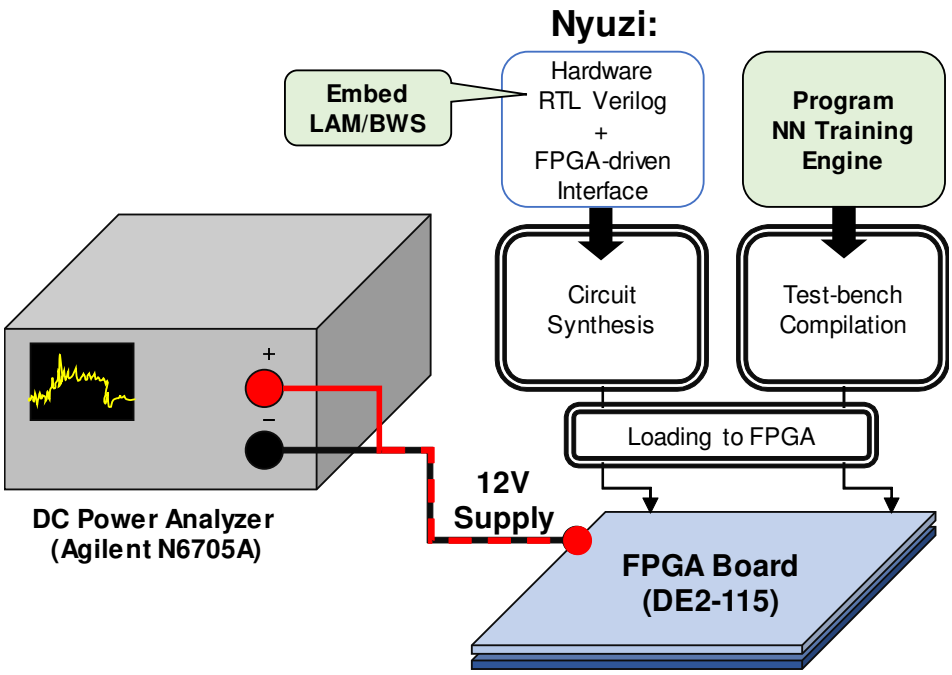


Figure 2.14: Flow for measuring power of LAM (and LAM+BWS) based Nyuzi on FPGA.

Table 2.5: Number of logic elements used to implement Nyuzi on FPGA.

	# of used logic elements			
	EFM	LAM	BWS	LAM+BWS
Overall	82,989	71,023	71,909	68,681
FPU	33,056	21,224	21,545	18,342

LAM and LAM+BWS saves about 12K and 14.3K logic elements compared with EFM, respectively, which are mainly contributed from FPU blocks.

A C-program code was prepared for NN training. The implemented code is compiled and the binary code is loaded in Nyuzi on FPGA. Then, this experiment launches the NN training program on Nyuzi and measures the power consumption of FPGA. Fig. 2.15 shows the hardware setup for measuring power dissipation. Agilent N6705A DC power analyzer is used to provide 12V power supply of DE2-115 board, and it also serves as a power meter. In this setup, the measured power includes not only the power of Nyuzi on FPGA but also that of other peripheral circuitry on the board.

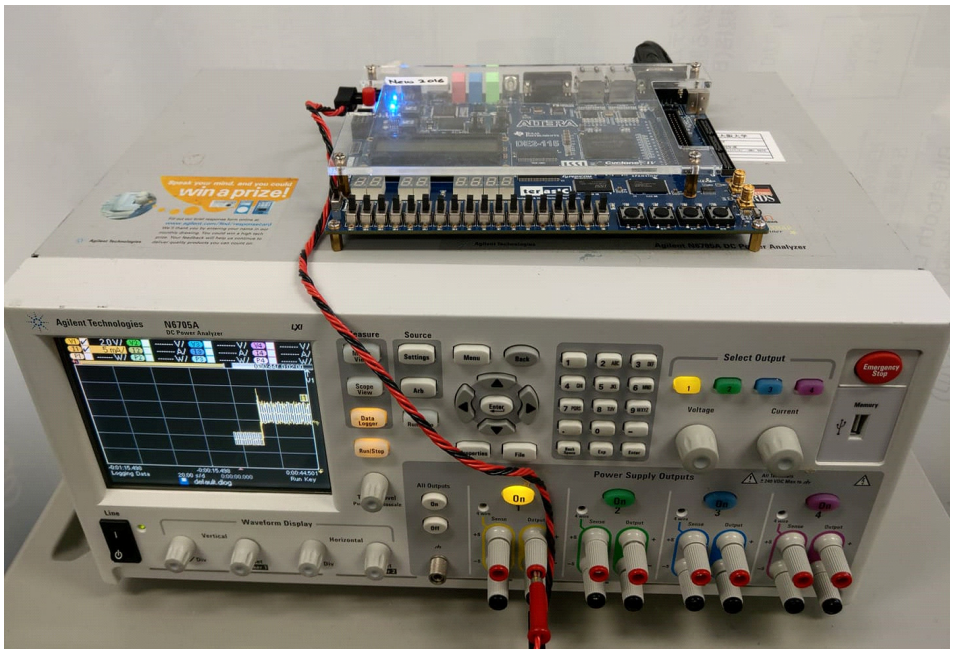


Figure 2.15: Photo for power measurement setup of Nyuzi processor. The FPGA board is placed on the DC power analyzer. The power analyzer is used for voltage supply and power measurement.

2.5.2 Measurement Results

Fig. 2.16 shows the measured transient power consumption during Nyuzi operation. The waveform is divided into three stages according to the Nyuzi operation status, and the red line in Fig. 2.16 represents the average value in each stage. Referring to the assembly code, the period (3) executes only the "NOP" operation. Thus, this work supposes that the difference in average power between period (2) and period (3) represents the power necessary for NN training.

Fig. 2.17 firstly shows the measured power of NN training for FOURCLASS dataset. Again, the definition of EFM, LAM, BWS, LAM+BWS are all consistent with Table 2.4. Here, the figure includes the results for different programming styles; single-thread and multi-thread. The power values for each style are normalized by that of EFM case with the same style. The results show that LAM and BWS are effective in power saving irrelevant to the programing styles.

Fig. 2.18 shows the measurement results for four higher-dimensional datasets. Again, a similar amount of power reduction is obtained for all the datasets. LAM-based floating-point training computation achieves 24%-28% power improvement, and the improvement increases to 35%-41% in the LAM+BWS case.

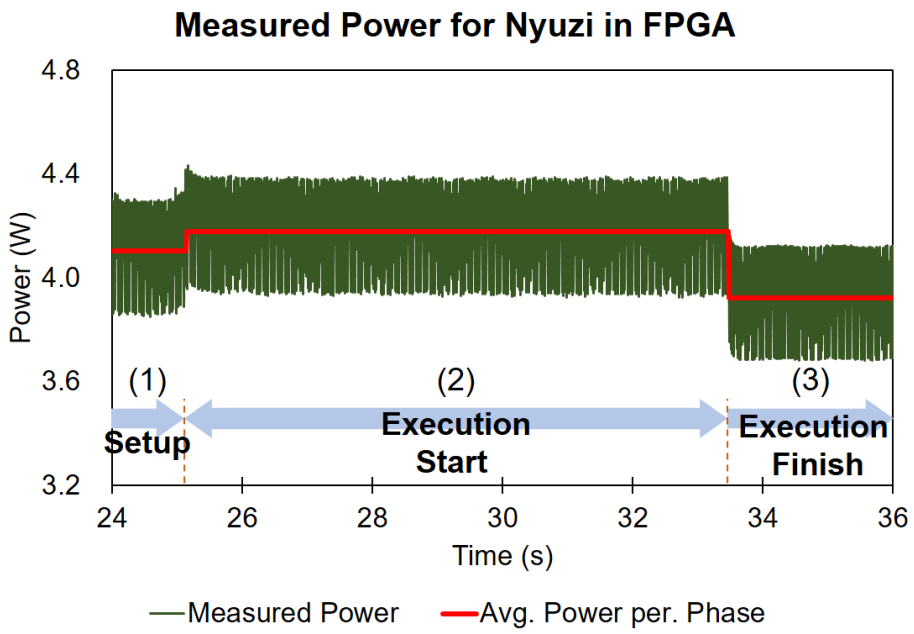


Figure 2.16: Transient power response measured during Nyuzi operation. Phase (2) is the phase for executing NN training program. In Phase (3), the training process already finished.

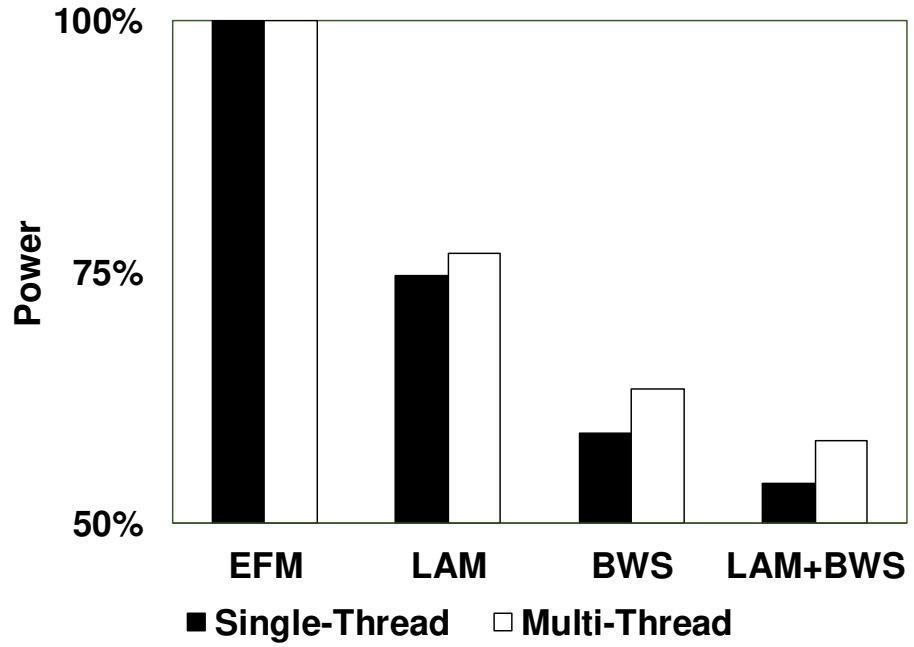


Figure 2.17: Power measurement results for single-thread and multi-thread (FOUR-CLASS dataset). All the values are normalized to EFM case.

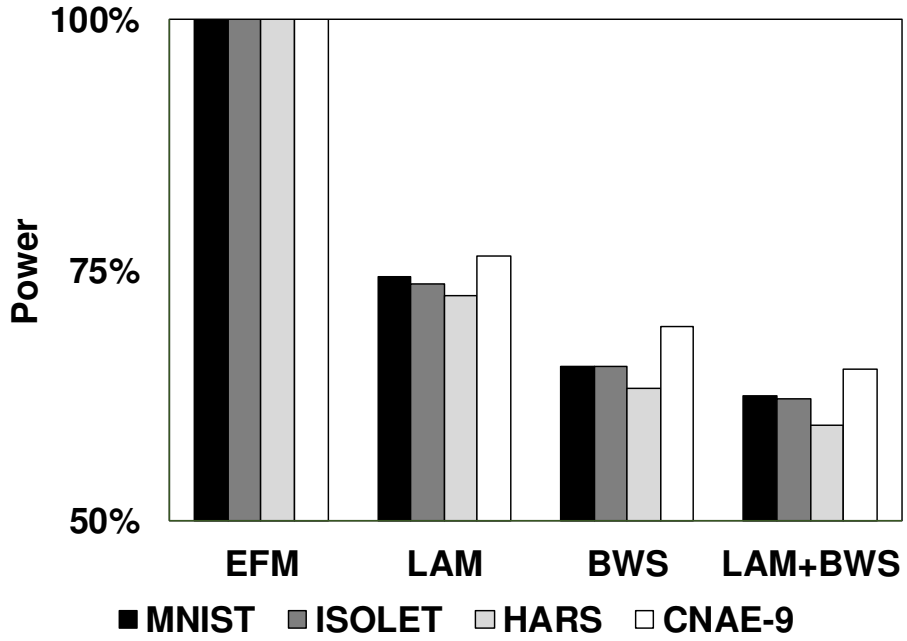


Figure 2.18: Power measurement results for MNIST, HARS, ISOLET, CNAE-9 datasets. All the values are normalized to EFM case.

2.6 Evaluations for Deeper NN Models

The advantage of applying LAM and BWS to deeper NN training is presented in this section. This work shows the training results of NNs with 2, 3, and 4 hidden layers, respectively, for MNIST dataset. In every NN structure, each hidden layer consists of 50 neurons. The NN training results for adopting solo-EFM and solo-LAM with different configurations of BWS are shown in Fig. 2.19.

Fig. 2.19 shows that, up to 4-hidden-layer NN, LAM-based training yields the accuracy comparable to that of EFM-based training as long as the result of EFM-based training with BWS is reasonably accurate. When 97% accuracy is considered acceptable (above 10 fraction bits), the accuracy drop contributed by LAM is less than 0.3%. This result indicates that the training accuracy is primarily determined by BWS instead of LAM, and LAM is applicable even when aggressive BWS is implemented. Note that the absolute accuracy could be improved with more sophisticated NNs, such as CNN [13, 14, 35]. Figs. 2.20(a), 2.20(b), and 2.20(c) show the results for other datasets of HARS, ISOLET, and CNAE-9 which present similar trend as MNIST. Overall, the average and maximum accuracy drops by LAM for the cases other than MNIST are mere 0.1% and 2.2% when 94% accuracy is considered acceptable, and LAM outperforms at 31% points in all the cases used in this work.

Fig. 2.21 shows the training curve for MNIST dataset. The scenario for the NN involves EFM, LAM, BWS, LAM + BWS, where the terms aligned with Section 2.4.1. As shown in the figure, at each epoch from 1 to 40, there is no significant difference in classification accuracy between LAM-based and EFM-based training. This result indicates that adopting LAM in NN training does not require additional processing time to reach the same accuracy, and thus, at least within 4-hidden-layer NNs, training completely relying on LAM is qualified and EFM is not necessary.

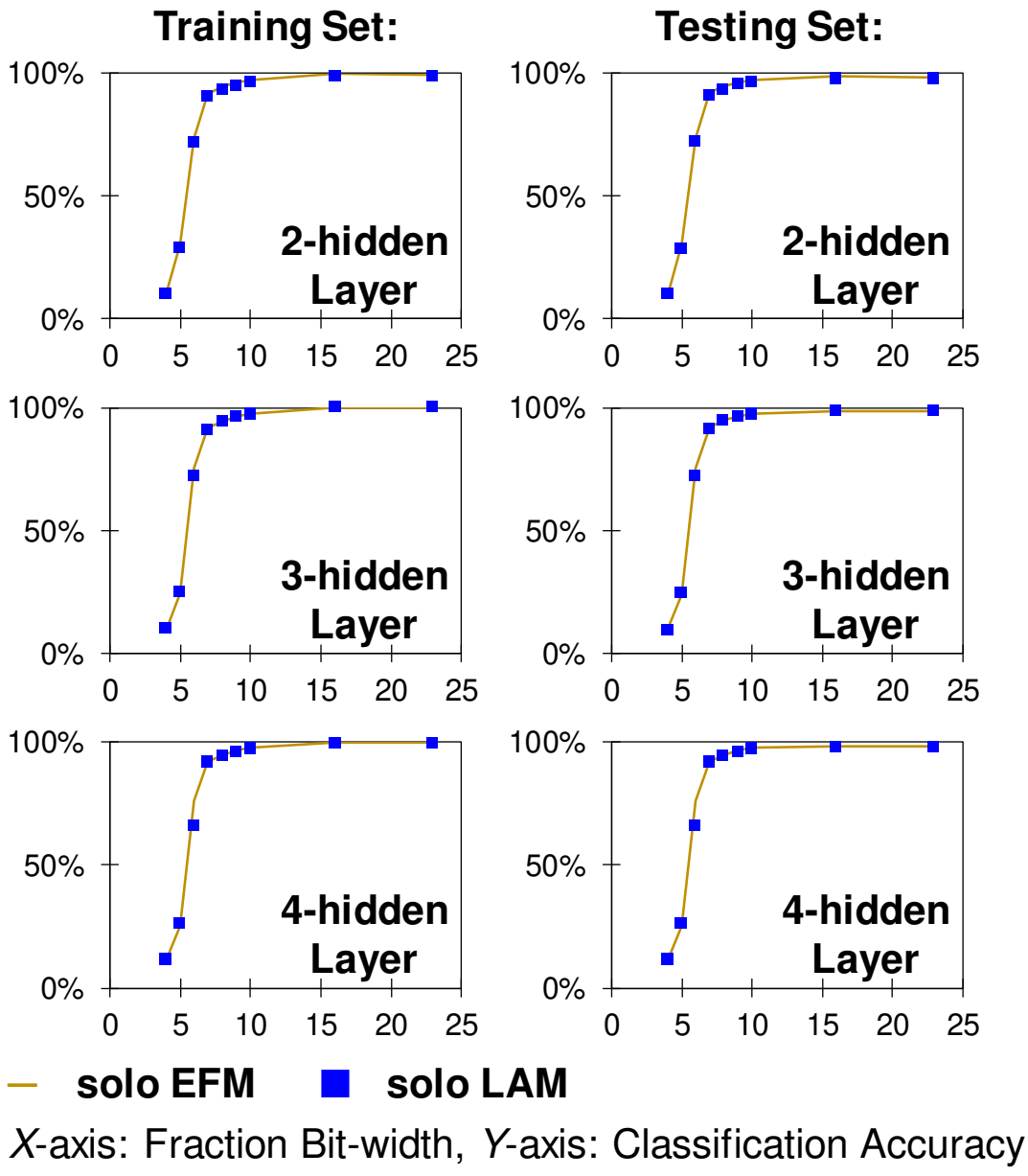
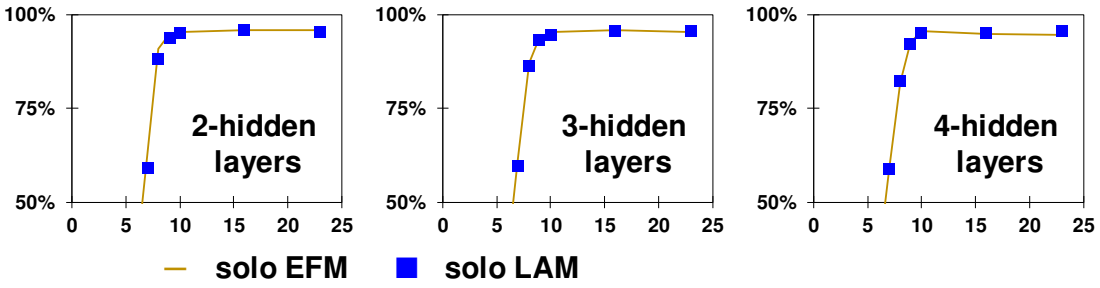
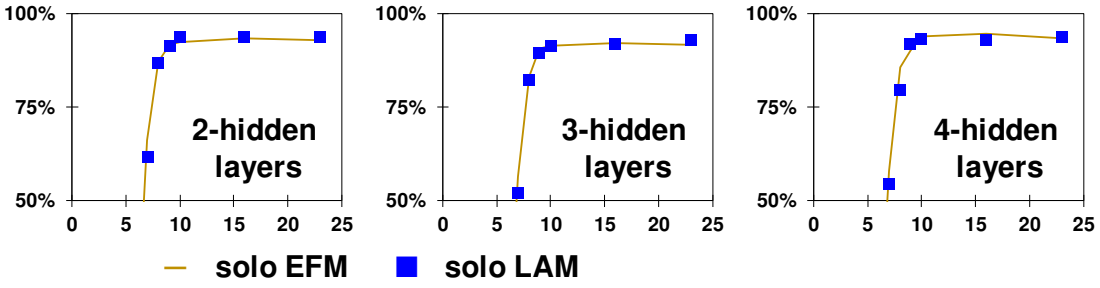


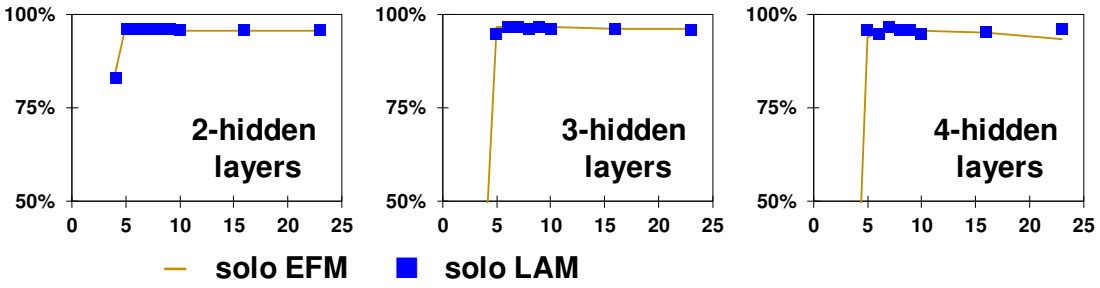
Figure 2.19: MNIST training results for NNs having 2, 3, and 4 hidden layers and various fraction bits. The attained accuracies are almost identical.



(a)



(b)



(c)

Figure 2.20: Deeper NNs training results varying with fraction bits. The attained accuracies are almost identical (a) HARS (b) ISOLET (c) CNAE-9.

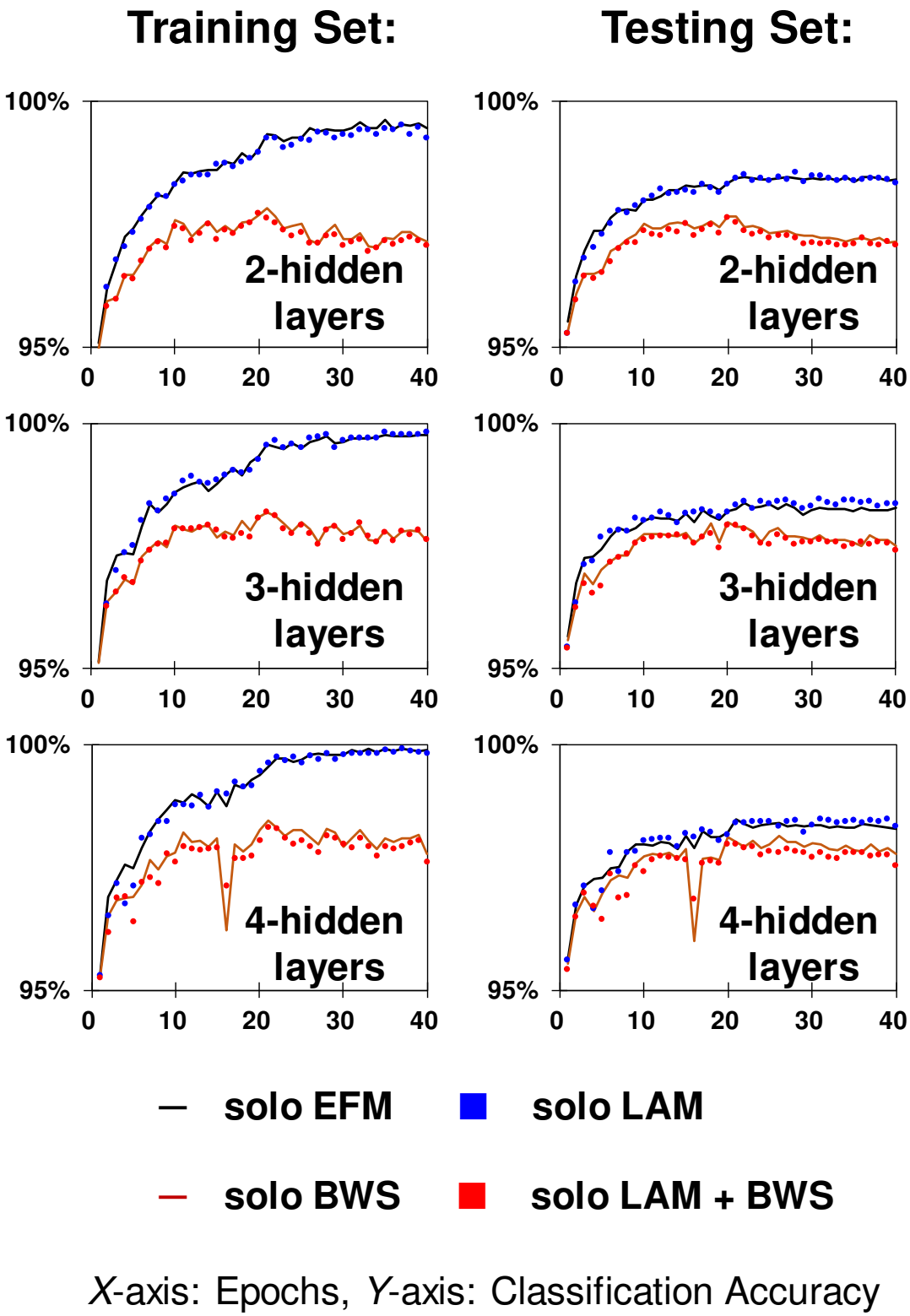


Figure 2.21: Training curves for MNIST. The convergence speeds of solo-EFM and solo-LAM training are almost identical.

2.7 Conclusion

This chapter evaluated whether approximate floating-point multiplier, which can cover a broad dynamic range, could be adopted in NN training achieving higher energy efficiency. Specifically, this work focused on logarithm-approximate multiplier (LAM) incorporating bit-width scaling (BWS) to reduce primary MAC computation complexity. The experimental results with dedicated hardware design show that training NNs with LAM can achieve 10% speed-up and 2.3X power reduction in addition to 2.3X area saving as well at the same speed when training a 2-D classification dataset. Even when training with LAM + BWS, there is no more than 1.0% accuracy discrepancy compared with the exact multiplier, where LAM + BWS outperforms, rather than degrades, the accuracy more frequently. As for the hardware performance, 4.9X energy efficiency is attained, where 2.2X originates from LAM. This work further quantified LAM effectiveness with an open-source GPU design. The power reduction was evaluated with the FPGA hardware measurement. About 28% power efficiency improvement in the LAM-based GPU design is confirmed compared with the EFM-based GPU design. Finally, LAM and LAM + BWS are experimentally qualified to be applicable to training up to 4 hidden layers, even with aggressive BWS.

Chapter 3

Mode-wise Voltage Scaling

Unlike AC techniques discussed in Chapter 2, this chapter seeks a design methodology to achieve power/energy efficiency for any types of designs. ASA introduced in [81, 82, 86] performs effective enhancement for voltage-scaling efficiency with stochastic treatment for less-but-not-zero timing error. On the contrary, this chapter leverages the "mode-wise" concept early proposed in [76] and further enhances it to involve multi-corner multi-mode (MCMM) feature in EDA tool that is highly compatible with mode-wise ASA implementation. With the aid of downhill simplex algorithm (DSA), the different operation modes in a design can activate different circuit blocks that are optimized for their minimum acceptable voltage (MAV). As mentioned in Section 1.6.3, Since the supplied voltage is scaled with mode-wise manner, the overall energy efficiency is improved compared with single-mode-based VS. Besides, this chapter also introduces a safe way to identify false paths during ASA implementation. Therefore, the design after mode-wise ASA manipulation guarantees there is no timing error at the scaled voltage of interest (MAV for each mode) after ASA circuit optimization.

3.1 Introduction

Voltage scaling (VS) is a classical yet powerful technique widely adopted by industry to achieve quadratic power reduction. Industrial designers conventionally would preserve extra timing margin for the devices to prevent circuit logic error or malfunction due to environmental variation in terms of the process, voltage, temperature, or aging. Therefore, VS plays a good role as a tuning knob for compensation. With judicious supply voltage control, the power/energy of the design in operation could be reduced. On the other hand, of course, there is a fact that reduced voltage would increase the gate delays and raise the probability of timing error. In case the induced timing error could result in logic error or even catastrophic malfunction, the promising solutions for circuit designer can be briefly categorized into two groups, where the one is adaptive voltage scaling (AVS) and the other one is activation-aware slack assignment (ASA), which are

already introduced in Section 1.5. In addition, as mentioned in Section 1.6.3, existing state-of-the-art MTTF-aware ASA implementation relies on allowing less-but-non-zero timing error [81, 82, 86], which limits the generality of applications and currently the scheme of functionality validation for aging-aware analysis remains controversial. On the other hand, in order to enhance VS efficiency without tolerating timing errors, it is necessary to explore for other sources of margin that can be exploited. As mentioned in Section 1.6.3, mode-wise idea is introduced by [76] to downscale the voltage on the most power-hungry mode for improving overall power consumption. However, immature ASA implementations by [76] may induce glitch events, which raises other risks regarding the timing error. Therefore, under the premise of no timing error, it is desirable to deliver a solution that can enhance ASA potential for better VS efficiency. Therefore, for circuit designers who want to achieve low-power design, a promising solution to enhance VS efficiency without any timing error concern should be quite attractive.

This chapter proposes a methodology to achieve a design applying to mode-wise VS (MWVS) under the scheme of ASA, with guaranteeing no timing error. First, a mode-wise voltage and corresponding design freedom is defined as the design variables, and then a problem for MWVS is formulated in consideration of the duration of each mode. Then, this work presents a fine-grained way to identify the false paths for each mode. The proposed methodology exploits multi-corner multi-mode (MCMM) design flow in a commercial tool that considers sets of false paths in individual modes for performing mode-wise ASA. The solution space is explored through DSA, which is a direct search method for solving nonlinear optimization problems without derivative computation. Note that the proposed method for MWVS is a general solution for application requiring low-power operation.

The contributions of this chapter are:

- Formulate an optimization problem and provide its solving procedure to achieve mode-wise voltage-scalable design. Experiment results based on RISC-V design show that 13% to 20% overall power saving when treating design with conventional VS as the baseline. In addition, 8% to 13% additional power gain originates from the mode-wise idea.

- Introduce a cycle-by-cycle fine-grained method to identify false paths which are used during ASA implementations. Compared with the conventional way of false-path identification, the introduced method can save 31% to 42% leakage power.

The rest of this chapter is organized as follows. Section 3.2 reviews the related work about ASA and MWVS, and also MCMM is briefly mentioned. Section 3.3 formulates the MWVS design optimization problem. Section 3.4 describes the proposed methodology for solving MWVS with DSA, which uses MCMM + ASA, and also presents fine-grained false-path identification. Section 3.5 applies the proposed methodology to RISC-V and shows experimental results. Section 3.6 concludes this chapter.

3.2 Previous Work

3.2.1 Mode-wise ASA

ASA is the technique to manipulate the timing margin of the active paths by engineering change order (ECO) phase. The activated paths after ASA have more timing margin with the cost of the area from additional buffer insertion or fast-but-large gates swapping, while the extra timing margin offers further down-scaling of voltage and benefits the power/energy reduction. The benefit from ASA has been promoted in the state-of-the-art works [76, 81, 82, 86].

However, even though researchers [76] have introduced the term "mode-wise ASA," their implementation approach of ASA, which is the same as the previous work of MTTF-aware ASA [81, 82, 86], considers only one specific operation mode and focuses on enhancing the VS efficiency only for that single mode. Therefore, if there are three distinct operation modes from 1 to 3, for example, those modes are united to form a single mode such as Mode $1 \cup 2 \cup 3$, as shown in Fig. 3.1(a). In this case, further VS opportunities existing in Mode 1 and Mode 2 are spoiled, as shown in Fig. 3.1(b). Conversely, if multiple modes, are individually considered in ASA-aware voltage-scalable circuit design, more benefits of overall energy saving can be expected because each mode consumes less power.

MWVS is a promising approach that can exploit the bias of active paths in different operation modes for voltage scaling. On the other hand, it is necessary to establish a design methodology that can ensure no timing error occurrence, cope with multiple operation modes, and efficiently explore the design space. Here, note that the scaled voltage is applied for each mode (workload) even under MWVS scheme, and the supplied voltage is spatially identical in the design of interest. Therefore, the concept of MWVS is practical and reasonable for industrial designs. On the other hand, it is noted that ASA for pursuing lower voltage operation involves circuit overhead due to timing margin expansion. Hence, this circuit overhead and achievable VS must be carefully taken into account in the design methodology.

3.2.2 MCMM

Multi-corner multi-mode (MCMM) design flow has been recently developed as one of the built-in features in many EDA vendors such as Synopsys and Cadence [100, 101]. The MCMM flow allows the timing analysis and the optimization of a hardware design according to multiple modes, where each mode can be associated with different clock periods, PVT corners (process, voltage, temperature), and design constraints for mode-based timing. The MCMM design flow works to generate a circuit design that satisfies all the required specifications in individual modes simultaneously. The MCMM flow can significantly save the design turn-around times for MCMM designs [101].

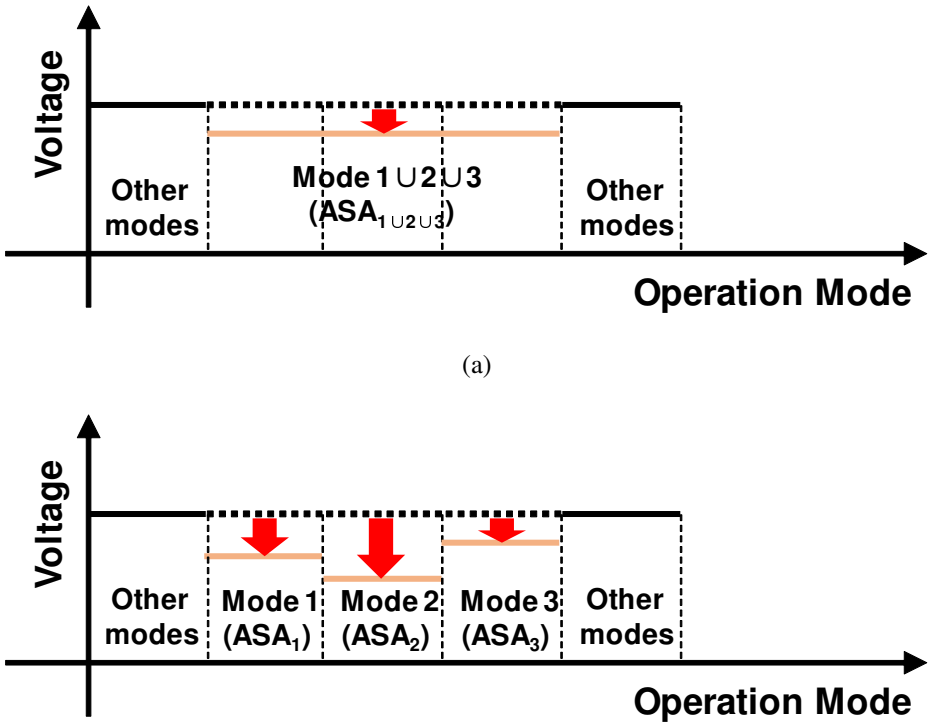


Figure 3.1: Expected voltage scaling operations for (a) single-mode VS design and (b) multi-mode (mode-wise) VS design.

With the MCMM flow, the ASA technique can be explicitly extended to MWVS design methodology. The mode-wise ASA aims to obtain a design where all the FFs at nominal voltage and the active FFs at scaled voltage meet the clock frequency constraint. Thus, it is expected that the MCMM flow is a promising choice to enhance the capability of MWVS design flow.

3.3 Problem Formulation for MWVS

This section formulates the problem for optimizing a design under MWVS as follows.

- *Input*:
  1. a gate-level pre-ASA circuit design
  2. Mode M_i and associated duration DR_i, where $i = 1, 2, ..., N$
  3. nominal voltage V_{NOM}
  4. cycle time T

- *Output*:
  1. a gate-level circuit after MWVS-aware ASA.
  2. voltage V_i for mode M_i, where $i = 1, 2, ..., N$
- *Objectfunction*
  1. Minimize: $\sum_{i=1}^{N} \text{Power}(M_i, V_i) \times DR_i$
- *Constraints*
  1. $\text{Delay}(V_i, \text{Act\_Paths}_i) \leq T$, where $i = 1, 2, ..., N$
  2. $\text{Delay}(V_{NOM}, \text{All\_Paths}) \leq T$
  3. $\text{Area} \leq \text{Area}_{\text{max}}$
- *Variables*
  1. voltage V_i
  2. size and Vt type of individual cells

The input and output are the gate-level circuits before and after MWVS-aware ASA. M_i is the i-th mode in N modes, and DR_i is the portion of the i-th mode duration, and hence each $\text{DR}_i \leq 1$ and $\sum_{i=1}^{N} DR_i = 1$. The predetermined nominal voltage and clock cycle are V_{NOM} and T, respectively. The objective of this problem is to minimize the summation of the power multiplied by the duration from mode M_1 to M_N. The first two constraints are given for the timing closure. That is, for each mode M_i, all the delays of active paths in mode M_i, which are denoted as Act_Paths_i, should meet the setup time constraint for the given clock period of T at the operating voltage of V_i. In addition, at the nominal voltage V_{NOM}, the delays of all the paths, All_Paths, in the design should keep setup timing clean within the same criterion of T. Also, the area is constrained with Area_{max} since faster logic cells tend to be large. Hold timing constraints are considered, but they are omitted in the above since they are not the primary concern in this work.

Here, it should be mentioned that the power in each mode M_i is determined by its operation voltage V_i in two ways. The first way comes from the fact that the power is expressed as the product of voltage and current, i.e., Power=$V \times I$. The second way is that the cell sizing and Vt assignment (swapping the same type of cells to different threshold voltage) result, which affects power dissipation, depends on V_i since the cell swapping is performed such that the path delay constraints are satisfied at V_i. Therefore, solving this problem needs to consider these two dependencies of power dissipation on the selection of V_i.

Fig. 3.2 shows an illustration regarding the timing constraints. There are four groups of paths in the figure; the paths activated in Mode 1 only (blue), Mode 2 only (red), both Mode 1 and 2 (purple), and non-activated paths (yellow). Three voltages having

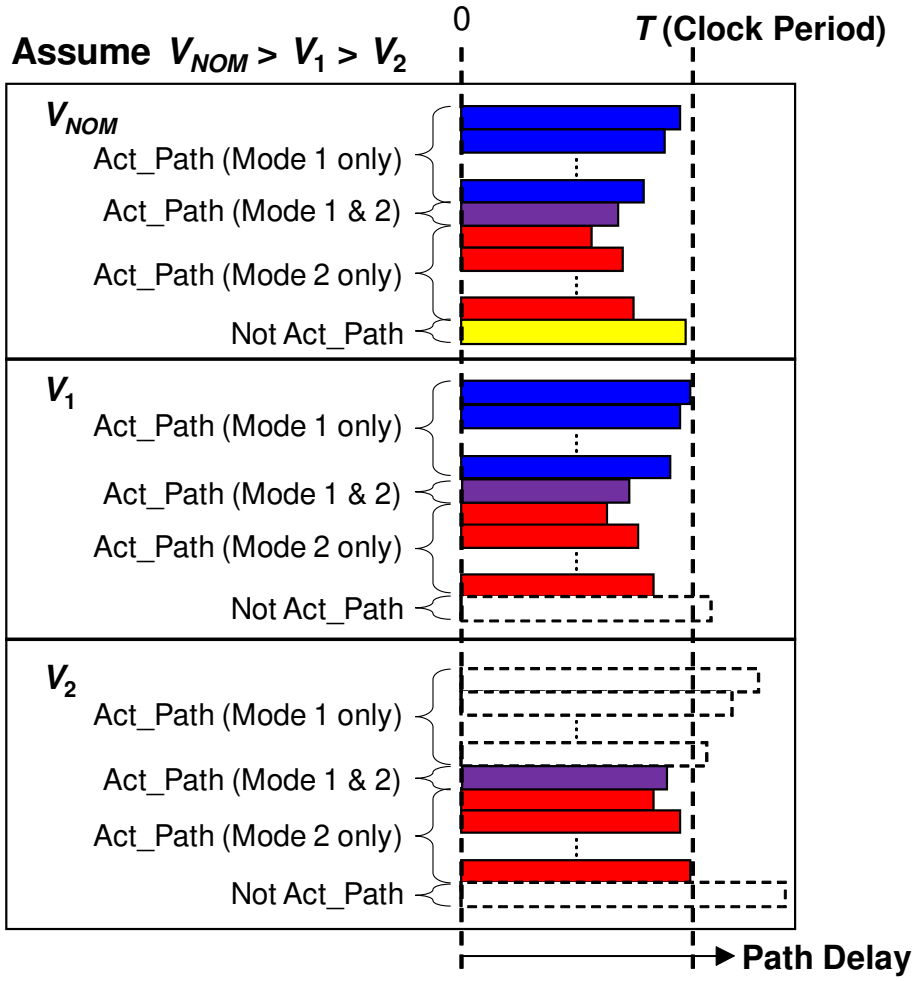


Figure 3.2: Mode-wise timing constraints (two-mode case).

$V_{NOM} > V_1 > V_2$ relation are assumed. At V_{NOM}, all the path delays should pass the setup-time criterion with a given clock period of T. V_1 is the scaled voltage for Mode 1, and thus all the paths activated in Mode 1 (blue and purple) should meet the criterion of T under V_1. V_2 for Mode 2 (purple and red) follows accordingly. Note that the paths in blue are excluded from the criterion for Mode 2, and their delays can violate the setup-time constraint. The paths in purple activated in both Mode 1 and Mode 2 are required to meet the criterion for both the modes, while the paths in yellow not activated in Mode 1 or Mode 2 can violate the criterion in both the modes.

In summary, this optimization problem aims at the minimization of the total sum of Power × Duration. Since the duration represents the time period and Energy = Power × Time, the objective is equivalent to energy minimization.

3.4 Proposed Design Methodology for MWVS

This section introduces the proposed MWVS design methodology that solves the optimization problem formulated in Section 3.3. Ultimately, the solution want to determine V_i, cell sizes, and cell types simultaneously. However, for each V_i value, the solution need to perform time-consuming circuit optimization, i.e., cell sizing and swapping for timing closure. Furthermore, this timing closure optimization needs to satisfy all the timing constraints separately specified for each mode, This work, therefore, takes a two-step approach that decouples V_i optimization and circuit optimization. Fig. 3.3 shows the overall flow. The iteration loop aims to obtain the set of V_i that can minimize the objective function. In this work, the downhill simplex algorithm (DSA) is used for this iterative optimization. DSA is a classical algorithm that can solve optimization problems having multidimensional variables without derivatives, which helps save the computation cost [102, 103], κ is the parameter to limit the number of iterations. This loop includes the circuit optimization with MCMM flow, where the timing closure optimization is executed for given sets of V_i. The detail of DSA in MWVS flow is described in Section 3.4.1. This circuit optimization with MCMM flow is explained in Section 3.4.2. Before this iterative optimization, sets of false paths should be prepared separately for each mode, which is described in Section 3.4.3.

3.4.1 Downhill Simplex Algorithm (DSA)

Downhill simplex algorithm (DSA), which is also known as Nelder-Mead method, is a simple numerical method for finding the optimal solution in a multidimensional space [102, 103]. The advantage of DSA is that it does not rely on the gradients to search the next decision, and therefore, DSA can deal with the optimization problem in which computing the gradient of the objective function is impossible or too time-consuming. Thanks to this property, DSA saves computing effort for the problem in which the computation of the objective function is time-consuming and numerical calculation of the gradient is prohibitively expensive. Hence, the proposed methodology adopts DSA.

To solve an M-dimension problem, DSA prepares an initial geometrical simplex composed of $M + 1$ vertices in the solution space. Alternating the vertex iteratively through the comparison of their corresponding objective function values makes the simplex approach to the optimal solution. Fig. 3.4 illustrates a three-dimension simplex, or as known as tetrahedron, as an example. Let us assume there is an objective function, F(.), which needs to be minimized, and there are $M + 1$ vertices X_1, X_2, ..., X_M, X_{M+1}, which are initially selected and sorted as $\mathrm{F}(X_1) \leq \mathrm{F}(X_2)$ ... $\leq \mathrm{F}(X_M) \leq \mathrm{F}(X_{M+1})$. Points R, E, O, I represent the next candidates of the vertices of the simplex with the name as reflection, expansion, outer contraction and inner contraction, respectively. The potions of R, E, O, I in the solution space are calculated as the linear combination of X_1, X_2, ..., X_M, X_{M+1}. DSA chooses the next set of vertices for the simplex in each iteration by

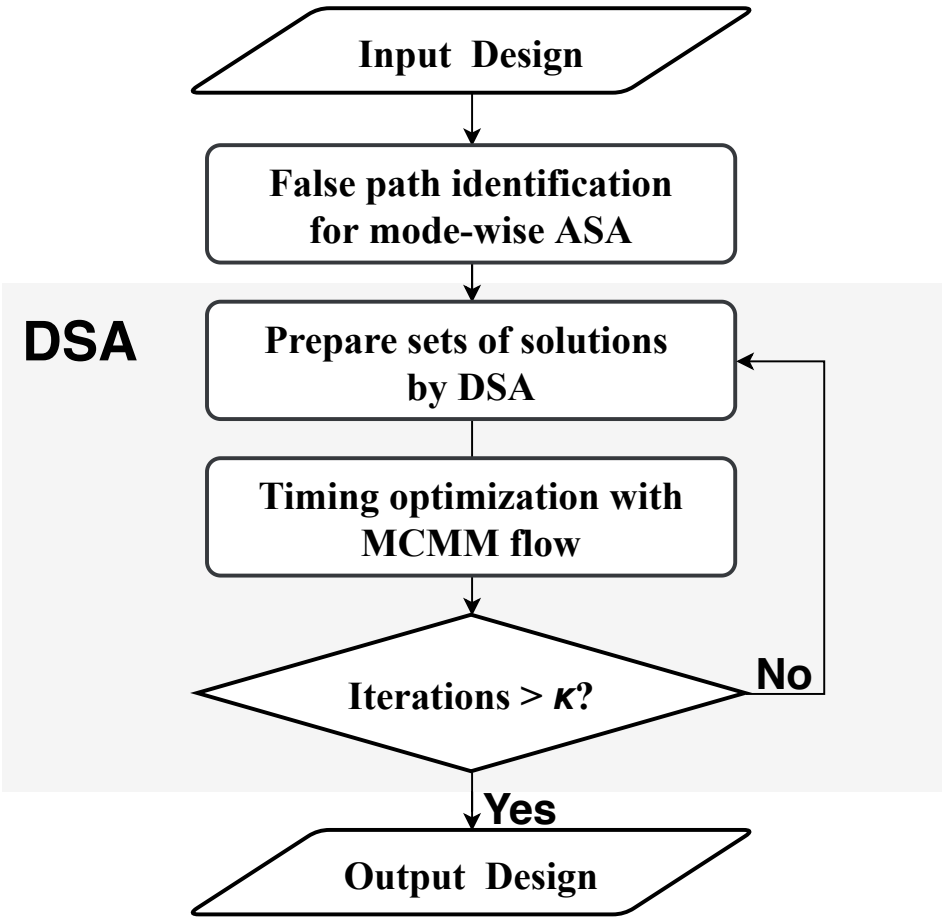


Figure 3.3: Overall flow for solving MWVS problem.

conditionally comparing F(R), F(E), F(O), F(I) with F(X_1), F(X_M), and F(X_{M+1}). For the details, please see textbook, such as [102]. Note that the number of points newly evaluated in each iteration is at most four independent of M, which is an advantage of DSA.

This paragraph explains how to solve the MWVS problem with DSA. As mentioned at the beginning of Section 3.4, the solution aims to search an optimal set of V_i using DSA, where a set of V_i corresponds to a vertex in DSA, and the total number of vertices are $N+1$. For each vertex, this flow performs circuit optimization for timing closure with MCMM flow and evaluate the objective function. However, when the set of V_i includes too low voltage, the given timing constraints cannot be satisfied and non-zero negative slack is observed. In this case, the set of V_i cannot be considered as a solution candidate. On the other hand, DSA cannot consider such constraints directly. Therefore, this flow adds a penalty function to the objective function to guide DSA taking into consideration timing violation. Here, there are many candidates of the penalty functions, but this

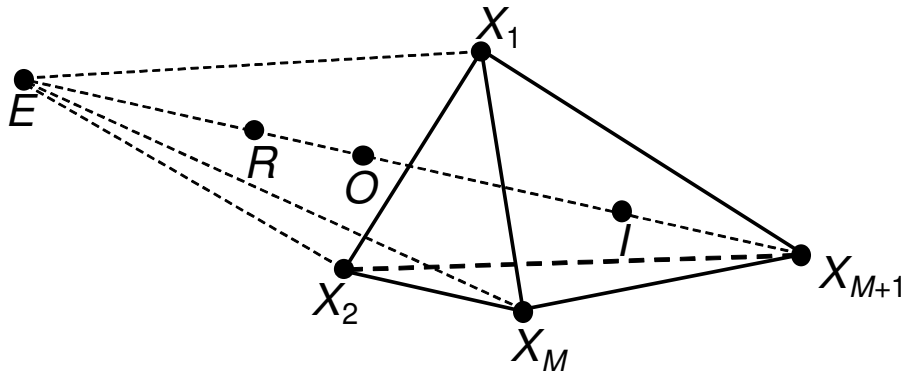


Figure 3.4: Illustration of three-dimensional simplex in DSA.

work uses the simple penalty function below since the experiment results are empirically found the penalty shape does not affect the final solution much.

$$\mathrm{F}(.) = \begin{cases} \sum_{i=1}^{N} Power(M_i, V_i) \times DR_i & \mathrm{WNS} = 0, \\ EXT & \mathrm{WNS} > 0, \end{cases} \tag{3.1}$$

where EXT is an extreme large number, and WNS stands for worst negative slack that is reported by the EDA tools.

3.4.2 Integrating Mode-wise ASA into MCMM Flow

This work exploits MCMM flow in EDA tools [100, 101] to carry out mode-wise ASA. The EDA tool asks users to prepare scenarios, where each scenario is associated with the timing library under a particular PVT corner and corresponded design constraints files. Therefore, the false-path specification commands for each mode should be included in the design constraints files of the associated scenarios.

Fig. 3.5 illustrates the preparation of the scenarios for mode-wise ASA. Several timing libraries characterized at different voltages are prepared beforehand. Then, for each scenario corresponding to Mode 1 to Mode N, this flow assigns the library at the specified voltage and associate the corresponding timing constraints files M_1.sdc to M_N.sdc as well. In addition to the clock period T, the false paths identified for individual modes are described in the design constraint. An additional scenario (for Mode NOM) is for the library of the nominal voltage of V_{NOM}, where no information on false paths is given. In other words, this scenario specifies all the paths in design that need to pass the timing at V_{NOM}. Note that if there are design-specific false paths, they should still be specified in M_{NOM}.sdc as well. Hence, as long as operating in nominal voltage, the design after MWVS flow can guarantee their functionality even for any modes that are not considered in MWVS design flow. The optimization in the MCMM flow aims to

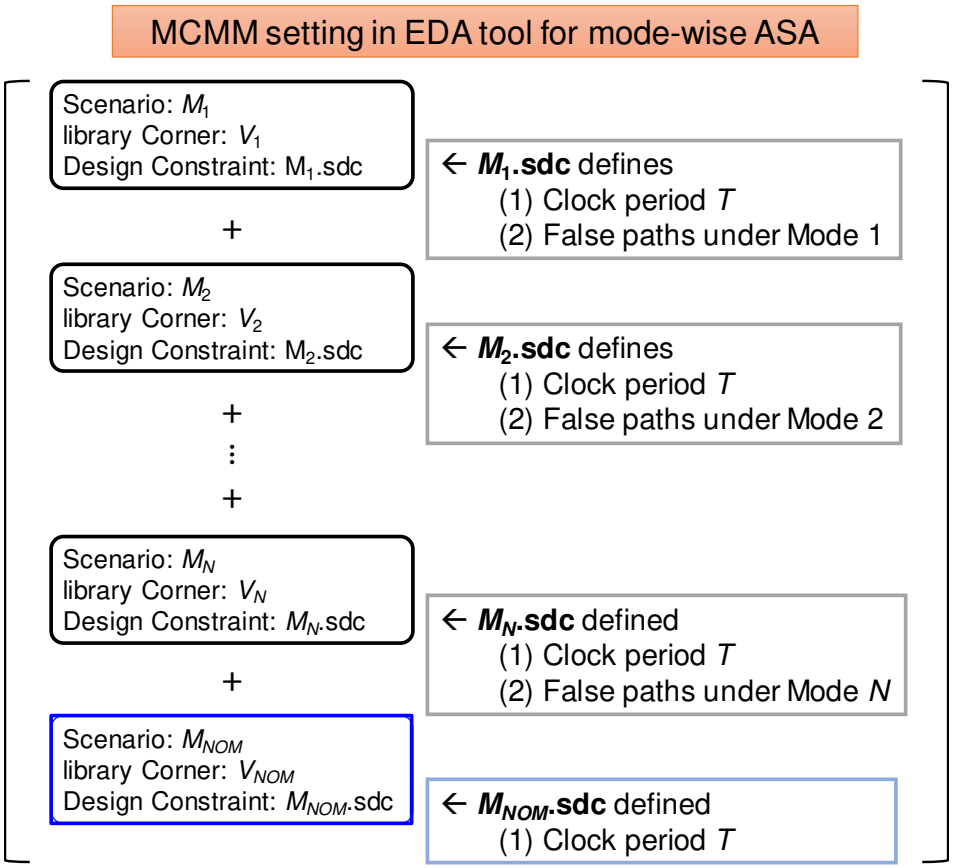


Figure 3.5: MCMM setting in EDA tool for MWVS design.

meet all constraints simultaneously. Therefore, as long as the timing is clean, i.e., WNS = 0, the design is guaranteed to operate correctly in every mode at the voltage of interest, and, for example, the logic simulation passes.

3.4.3 False Path Identification

Previous works on ASA [76, 81, 82, 86] pay attention to active paths (or flops) obtained from logic simulation results and allocate additional slacks to those paths. However, due to glitch events, it cannot guarantee that there is no timing error at the scaled voltage of interest after ASA circuit optimization. Let us explain the reason. Even when there is no transition on a path, a glitch may arise and propagates through the path after ASA timing optimization since glitch occurrence strongly depends on transition timings. Such new glitches are hard to be predicted and then raise the risk for timing violation. Therefore, instead of specifying active paths, this work decides to identify safe false paths that are non-active during the mode operation of interest. Thus there is no need to worry about accidental glitch transitions even after timing optimization.

Fig. 3.6 exemplifies false paths. The illustrated circuit is composed of flops (FF-0∼FF-7), AND2 gates (A1∼A5), and OR2 gates (O1∼O6). After performing a one-time logic simulation for a particular workload (mode), the states (0 or 1) for each cycle can be extracted at the output pins of all the flops and define the flops whose output states are identical with the previous cycle as non-active flops. Then, for each end-point flop (e.g., FF-0), with assigning non-active flops (e.g., FF-1, FF-7) in static timing analysis, the false paths can be extracted that include false sub-paths (e.g., from FF-1/Q to O1/A and FF-7/Q to O6/B). Even though the input patterns vary every cycle, some flops are non-active from the beginning to the end in a particular mode. Similar always non-active flops in a mode might be extracted by propagating mode-dependent constant values. The false paths are primarily extracted based on those flops and regard this method as conventional false-path identification (FPI).

On the other hand, it is found that applying cycle-by-cycle analysis could potentially increase the number of false paths. Let us take the circuit in Fig. 3.6 again as an example. Topologically there are 13 paths in the circuit, which are listed on the right side with ID numbers. Conventionally, in this schematic, these 13 paths are considered during timing closure. On the other hand, supposing FF-1 and FF-7 are non-active and 0 in a particular mode, four false paths (paths 1,2,12,13 in gray) are attained and the number of paths to be considered is reduced to 9. This reduction is identical to FPI mentioned above. Next, consider the case that FF-2, FF-3, FF-5 are not always non-active but either of FF-2 and FF-3 transitions only when the other stays 0, and the same relation holds between FF-5 and FF-3. This case reduces the number of paths to be considered in timing closure to 3 (paths 6,9,11 in black). Cycle-by-cycle analysis can automatically extract such FF-to-FF relations from the logic simulation results, which will be explained in the next paragraph. After the false path identification, the false paths are assigned in EDA tools in an ordinary way. EDA tool can simply assign the false paths through the false stages. Taking an example from Fig. 3.6, all the paths are assigned through A1,A2,A4,A5 as false paths. Gate A3 is also the false stage, but the paths through A3 can be specified by A4, and then A3 is skipped.

This paragraph explains how to automatically extract false paths mentioned above from the logic simulation results. The idea is very simple. The false paths are extracted for each clock edge. In this part, the executed analysis is the same as FPI. The set of false paths varies for each edge, but some false paths are included for all the clock edges. Finally, such false paths are given to the timing closure tool. In this way, the additional false paths discussed in Fig. 3.6 can be obtained. This way of false path identification is called as cycle-by-cycle fine-grained false path identification (CF-FPI). Fig. 3.7 illustrates CF-FPI from the example circuit in Fig. 3.6. The false paths are extracted from FF-2/Q to O2/A, FF-3/Q to O2/A, and FF-3/Q to A3/B for cycle_1 to cycle_2, and the false paths and obtain the final list, i.e., path from FF-2/Q to O2/A and FF-3/Q to O2/A. This example includes only two clock edges, but the same analysis can be applied to long simulation results.

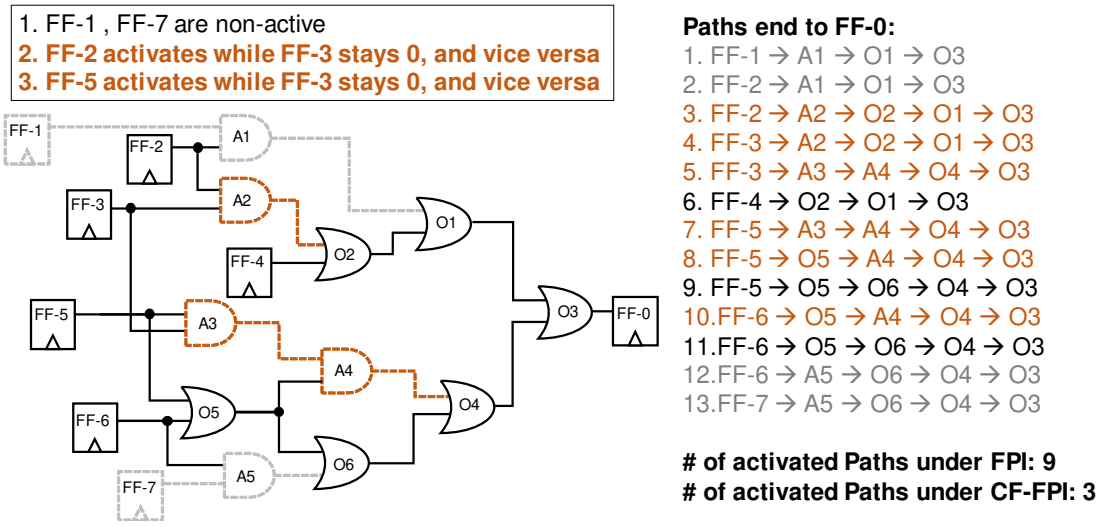


Figure 3.6: Examples of false path identification including cycle-by-cycle analysis.

Finally, this work further squeezes false paths. Till now, the false paths are extracted primarily by using the information on non-active flops. On the other hand, additional paths from active flops can be identified as “false". This possibility arises in multi-input combinational gates, e.g., AND2 in Fig. 3.8. Suppose that U1/A has only one rise toggle, and U1/B has only one fall toggle in the same cycle. In this case, due to the AND2 boolean logic, the output transition U1/Y is dominated by the falling input i.e., U1/B. The other pin U1/A has a non-critical transition (NCT), and thus the path through U1/A can be used as a false path. The important property of NCT is that this dominance relation is independent of timing, and it holds even after any timing optimization. Therefore, this false path can be exploited safely in cell-swapping optimization. It should be noted that this false path identification based on the NCT is possible with the cycle-by-cycle analysis. Besides, the multi-input gates possessing similar characteristics include (N)AND , (N)OR, AOI, OAI series of gates. MUXs have NCT possibilities since in the case of S=0 or S=1, either A or B pin becomes don’t care term, meaning that the transition at that pin becomes NCT. X(N)OR series, HA, and FA do not have this characteristic since every input transition affects the output state.

In order to give a more concrete picture for explaining how to extract FPI and CF-FPI, Algorithms 1 and 2 show the pseudo codes for the processes utilized in this flow for extracting the false-path list in a certain mode. The inputs are all based on one-time simulation results based on that mode, and the output is the final false-path list (denoted as FP_list) that can be loaded by EDA tool accompanied with MCMM feature for implementing mode-wise ASA. In Algorithm 1, the false paths are extracted from the non-active flops, where these flops have 0 toggle rate through the whole clock cycles in the m-th mode and the total number of these flops is denoted as F. **Paths-starting-from-$FF_{(j)}^{(m)}$** means that the set of paths which start from the j-th non-acitve FF in the m-

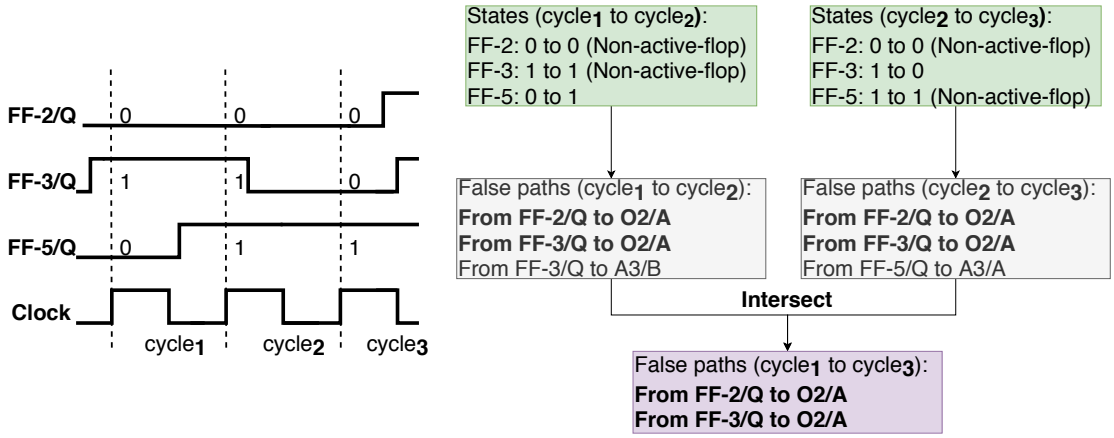


Figure 3.7: Cycle-based false path analysis.

th mode. With the help of EDA tool such as Synopsys PrimeTime or Design Compiler, it is available to trace the paths starting from the $FF_{(j)}^{(m)}$. CF-FPI shown in Algorithm 2 has a major difference compared with FPI in Algorithm 1 that it needs to extract the false paths cycle-by-cycle, and then performs an aggregation to find the intersect for all the cycle-based false-path lists. F^k in Algorithm 2 means the total number of the flops with 0 toggle-rate in the k-th cycle, and M^k means the total number of the nets with non-critical transition in the k-th cycle. **Paths-from-$\boldsymbol{FF_{(j)}^{(m,k)}}$** means the set of paths starting from the flop $FF_{(j)}^{(m,k)}$, and **Paths-through-$\boldsymbol{N_{(x)}^{(m,k)}}$** means the set of paths through the net $N_{(x)}^{(m,k)}$, where the set of paths can be extracted through the EDA tools.

Algorithm 1: FPI for m-th mode

1: Run logic simulation results with m-th mode.
2: Based on the results, collect the flops with 0 toggle-rate through whole clock cycles $FF_{(j)}^{(m)}$, where $j = 1, 2, ..., F$.
3: FP_list = [ ]
4: **for** ($j = 1; j \leq F; j++$)
5: FP_list $\leftarrow$ FP_list $\cup$ Paths-from-$FF_{(j)}^{(m)}$
6: **Output:** Design constraint file includes FP_list.

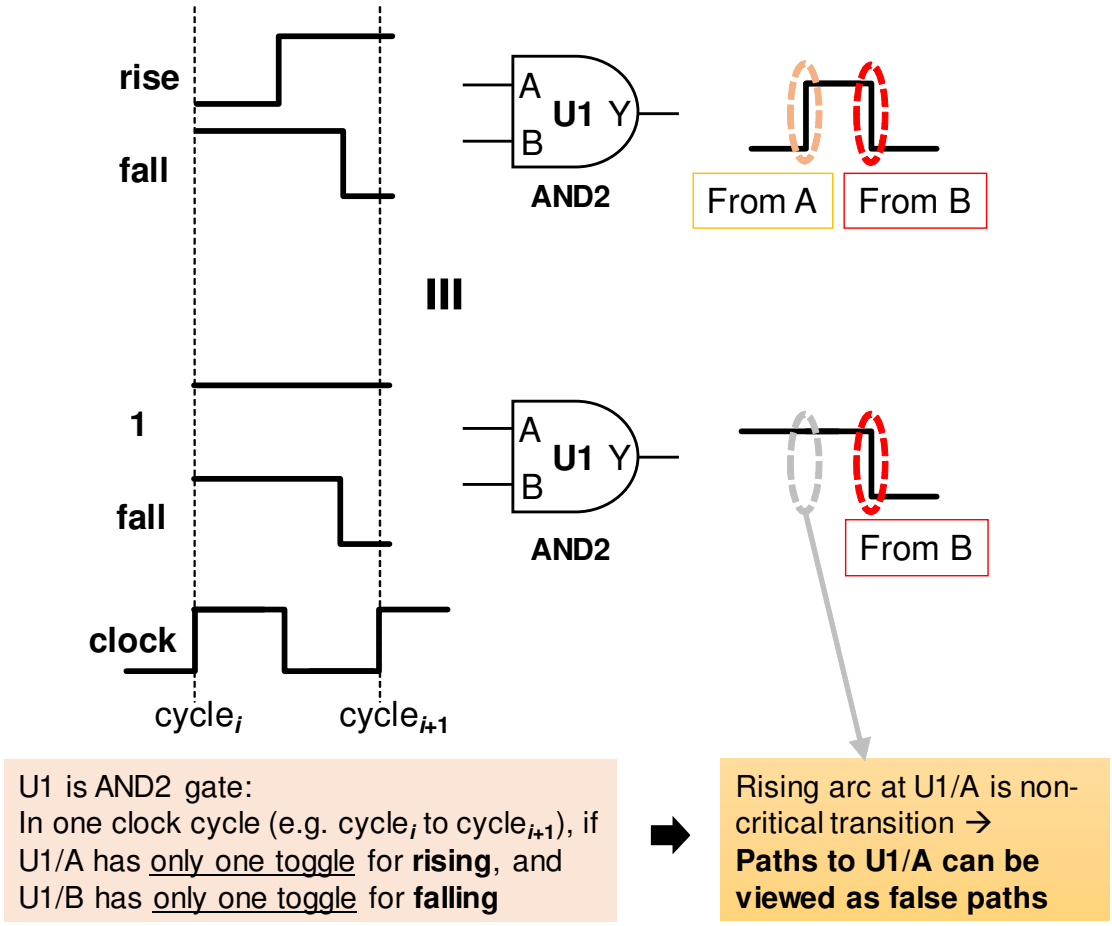


Figure 3.8: Cycle-based non-critical transition and false path analysis in AND2 gate.

Algorithm 2: CF-FPI for m-th mode

Run logic simulation results with m-th mode.
Based on the results, for k-th cycle ($k = 1, 2, ..., C$), collect
(1) the flops with 0 toggle-rate $FF_{(j)}^{(m,k)}$, where $j = 1, 2, ..., F^k$, and
(2) the nets with non-critical transition $N_{(x)}^{(m,k)}$, where $x = 1, 2, ..., M^k$.
FP_list = [ ]
for ($k = 1; k \leq C; k++$)
  FP_list$^{(k)}$ = [ ]
  for ($j = 1; j \leq F^k; j++$)
    FP_list$^{(k)}$ $\leftarrow$ FP_list$^{(k)}$ $\cup$ Paths-from-$FF_{(j)}^{(m,k)}$
  for ($x = 1; x \leq M^k; x++$)
    FP_list$^{(k)}$ $\leftarrow$ FP_list$^{(k)}$ $\cup$ Paths-through-$N_{(x)}^{(m,k)}$
  if $k == 1$
    FP_list $\leftarrow$ FP_list$^{(k)}$
  else
    FP_list $\leftarrow$ FP_list $\cap$ FP_list$^{(k)}$
Output: Design constraint file includes FP_list.

3.5 Experimental Results and Analysis

3.5.1 Setup

This section performs an experimental evaluation of the proposed methodology with RISC-V processor, a popular open-source CPU. The processor is synthesized with Nangate 45nm library [92] for VTG (low-Vt) and VTH (high-Vt) cells at 0.5 GHz by Synopsys Design Compiler (DC) which enables MCMM and set V_{NOM} to 1.0V. The power for each mode is estimated by DC. After obtaining the value for mode-wise power, the operation duration is considered to calculate the overall power. Since 0.5 GHz is not the highest frequency for synthesizing RISC-V, many logic cells in the circuit are swapped to higher-Vt cells to save leakage power in the initial synthesis.

Three workloads are selected in the experiments; (1) dijkstra and (2) sha, which are included in MiBenchmark [104], and (3) mt-matmul-fp, which is a floating-point (FP) matrix-multiplication. The workloads of dijkstra and sha use similar parts of processor components since they use integer arithmetic and logic operations. On the other hand, mt-matmul-fp utilizes the FP units, especially for multiply-accumulate computation. Therefore, the modes dedicated for dijkstra and sha are quite different from mt-matmul-fp, which meets this work's assumption that the operating voltage and the power consumption might have a discrepancy between different modes.

For comparison, this experiment prepares three methods for evaluation; (A) conventional VS, (B) single-mode ASA + VS, (C) mode-wise ASA + VS (proposed). Method (A) directly re-synthesizes the design at a lower voltage without any specification of false paths, which is a standard design flow and then the baseline in this work. Method (B) applies ASA that is based on the false paths that are not activated in Mode 1 to Mode 3, namely Mode 1 to Mode 3 are merged into a single mode. This method has the similarity as the previous work [76, 81, 82, 86] in terms of the number of modes while the stochastic treatment of timing error used in previous work [81, 82, 86] is disabled. Method (C) is the proposed approach described in Section 3.4. In (C), the duration for each mode can be explicitly considered to minimize the overall power dissipation, while the duration information cannot be considered in (A) and (B).

3.5.2 Results

Table 3.1 lists the experimental setup and results. The first column represents the three methods (A), (B), (C). The second column lists how many modes are used and shows the combination of the used modes. Method (A) and (B) only apply one mode, but (B) considers the merged mode. The third column is the assumed duration that only (C) can explicitly consider when computing overall power. The fourth column for (A) and (B) is the minimum voltage at which the WNS sustains 0, while for (C) it is the optimization results after 30 DSA iterations. The final column for (A) and (B) represents the power dissipation, and for (C) it means the overall power considering the duration. The

Table 3.1: Optimization results for (A) conventional VS (baseline), (B) single-mode ASA + VS, and (C) mode-wise ASA + VS (proposed).

Method	#Modes (used modes)	Duration	Voltage (V)	Power (W) (reduction)
A			0.82	0.358 (baseline)
B	1 (1∪3)		0.80	0.340 (-5.0%)
B	1 (1∪2∪3)		0.80	0.340 (-5.0%)
C	2 (1, 3)	50:50	(0.73, 0.80)	0.312 (**-12.8%**)
C	2 (1, 3)	70:30	(0.73, 0.80)	0.301 (**-15.9%**)
C	2 (1, 3)	95:5	(0.73, 0.80)	0.286 (**-20.1%**)
C	3 (1, 2, 3)	33:33:33	(0.76, 0.73, 0.80)	0.307 (**-14.3%**)

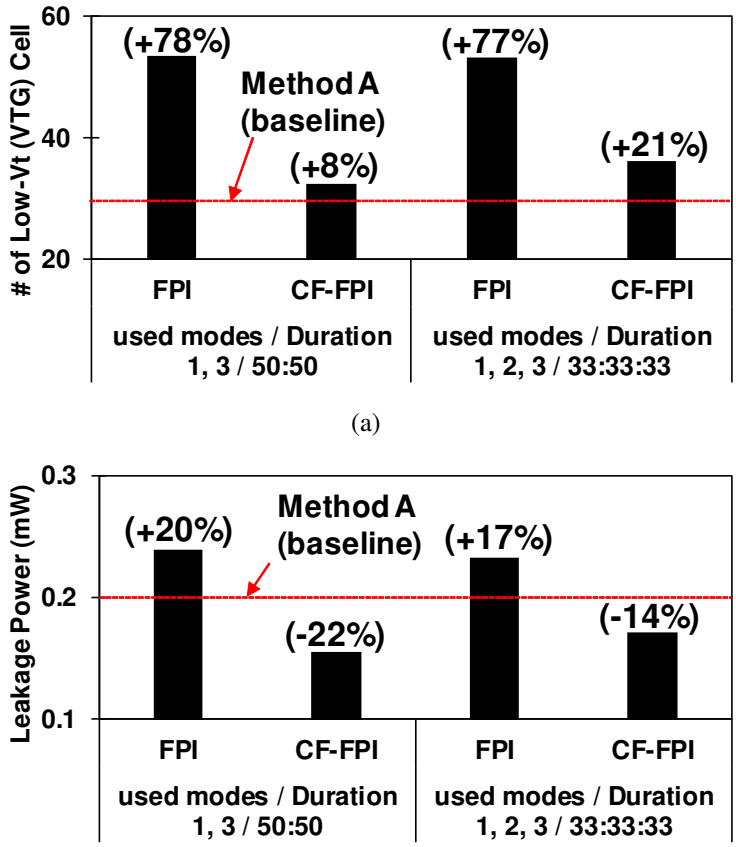
experiment applies FPI here for (B) and (C). FPI and CF-FPI for (C) will be compared later.

From Method (A) to (B), the power is reduced by 5.0%. (B) unites different modes to one, and hence the VS efficiency of (B) is limited by the mode having the highest voltage. In this case, Method (B) is determined by Mode 3 (mt-matmul-fp) due to its smaller room for VS than the other two modes. However, the proposed methodology of Method (C) optimizes the design considering all the modes separately, and thus it enhances the VS efficiency. Even in the case of two modes (1, 3) having 50:50 duration, an additional 8% gain is obtained from Method (B) to (C). Additionally, the proposed methodology allows different sets of duration. When the duration ratio of Mode 1 to Mode 3 is changed from 50:50 to 70:30 and even 95:5, the proposed method can gain the power reductions of 16% and 20%, respectively. In this work, the area increase is limited to be smaller than 0.5%, and then the area increase was at most 0.4%. On the other hand, the number of low-Vt (VTG) cells increased. For one circuit optimization for MCMM timing closure takes 8∼13 minutes for Method (A), (B) and (C) under FPI. Therefore, the entire MWVS design flow of (C) takes about six hours.

This work next investigates the impact of false path identification methods on the optimization results. Unfortunately, the MCMM flow with the false-path set of CF-FPI is slow and circuit optimization with CF-FPI could take 6∼7 hours for one run. Therefore, only the final iteration is executed with CF-FPI. Meanwhile, this work considers that the run time varies with different designs and workloads, and CF-FPI flow is not always so slow. Table 3.2 compares the number of commands for false path specification for the cases with and without applying CF-FPI. Roughly speaking, the number of commands is similar. Fig.3.9(a) compares the usage of low-Vt (VTG) cells. FPI specifies fewer false paths, and then the timing closure is more strict, which results in 77 to 78% larger number of low-Vt cells. On the other hand, CF-FPI identified more false paths than FPI so that fewer lower-Vt cells are used, and the increase of VTG cells is suppressed to only 8% to 21%. Although there is minor improvement can be attained for the total power (0.03%) due to the small portion of leakage power to the

Table 3.2: # of false-path specifications in design constraint file for each mode

	Mode		
	1	2	3
FPI	161K	162K	214K
CF-FPI	186K	192K	185K



(a)

(b)

Figure 3.9: Comparison between FPI and CF-FPI for (a) # of low-Vt (VTG) cells (b) leakage powers.

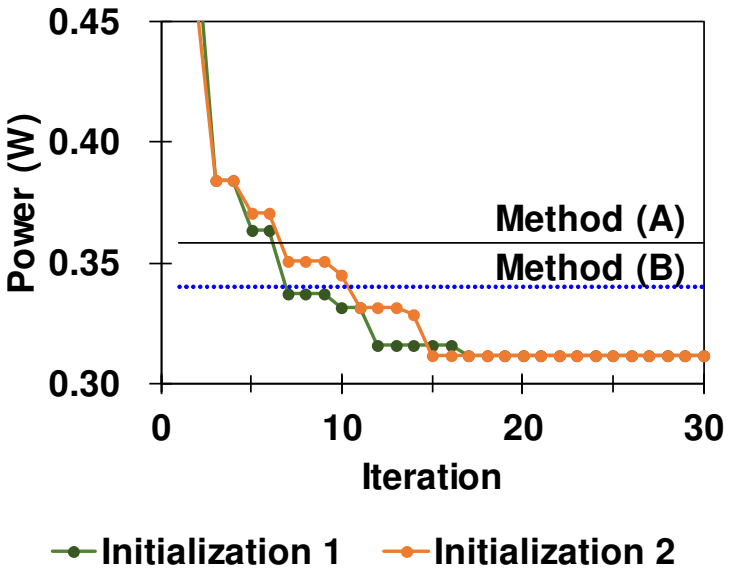
overall power, Fig. 3.9(b) reveals that applying CF-FPI reduces leakage power by 42% in the two-mode case and 31% in the three-mode case. This result indicates that CF-FPI facilitates the timing optimization and reduces the number of lower-Vt cells.

3.5.3 Proposed Methodology Investigation

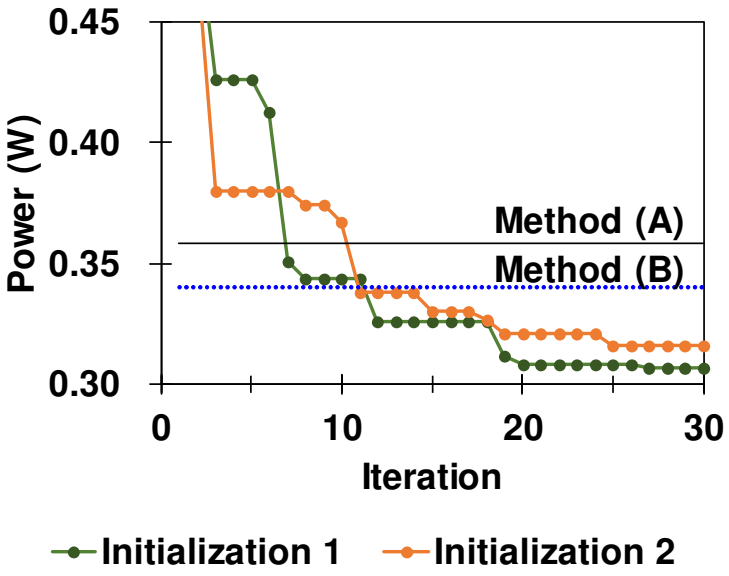
Fig. 3.10(a) and Fig. 3.10(b) show how the solution is becoming convergent, where Fig. 3.10(a) supposes two-mode (1, 3) case of 50:50 duration and Fig. 3.10(b) supposes

three-mode (1, 2, 3) case of 33:33:33 duration. In each figure, two curves are plotted starting from different initial values. Initialization 1 applies the voltage set of (1.0, 1.0), (0.9, 1.0), and (1.0, 0.9) as the start points in the two-mode optimization, while initialization 2 applies (1.0, 1.0), (0.85, 1.0), and (1.0, 0.85). Similarly, the start points are set in the three-mode case. Fig. 3.10(a) shows that both initialization sets of the two-mode case can converge to the same power value after 17 iterations. However, in the three-mode case, the different initial sets would reach different power dissipation values. The reason is that, although DSA is not a completely greedy algorithm, it does not have an explicit hill-climbing capability, and then it can fall into local optimal points. Meanwhile, the objective function is supposed to be somewhat a smooth function since it is a linear sum of the product of the power and duration for each mode, and the power is roughly proportional to the voltage squared. However, the MCMM flow of EDA tool might generate non-continuous space for this objective function once the number of modes increases.

Fig. 3.11(a) and Fig. 3.11(b) show the scatter plots of the voltages in two different modes evaluated in the optimization process, where the red points mean that the timing closure fails in the MCMM flow and the blue points mean that the timing closure succeeds. Fig. 3.11(a) is for two-mode case, and Fig. 3.11(b) is for three-mode case, where one dimension of voltage, i.e., voltage for Mode 3, V_3, is fixed at 0.81 V. Focusing on the two-mode case, a clear bound could be found; the timing closure passed in the MCMM flow when $V_1 \geq 0.73$ V and $V_3 \geq 0.80$ V. Although there are a few outlier points that fail, they are relatively rare, and the DSA works well. However, in the three-mode case, even when fixing one dimension of voltage V_3 at 0.81 V, the plot shows that the boundary is unclear and the red and blue dots are mixed, especially around the point of (V_1, V_2) = (0.73, 0.73). This behavior indicates that if the solution approaches this area, the algorithm might be disturbed by the outlier points and converge at a local minimum point. Actually, since the MCMM flow itself is another complex optimization problem and its complexity also increases according to the number of modes, the stability of the overall solution may become sensitive to the set of initial points. On the other hand, comparing to Method (A) and (B), the proposed methodology provides higher power efficiency. Note that DSA is not the sole solver for this problem, and other methods can be used as long as their performance is better.

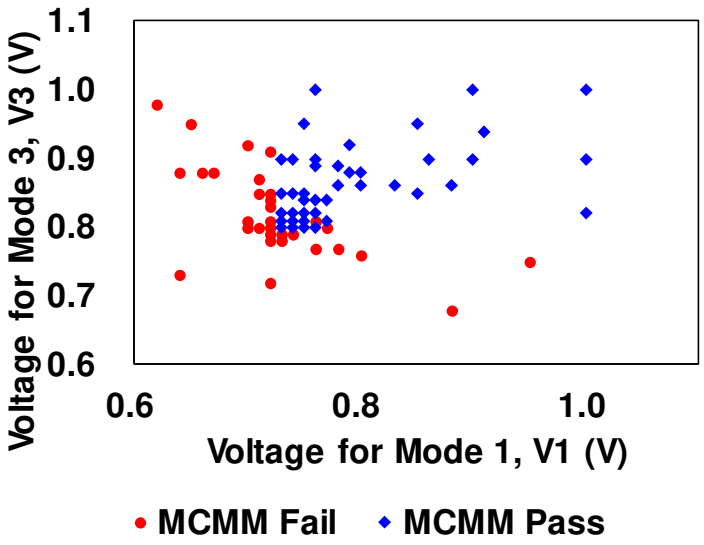


(a)

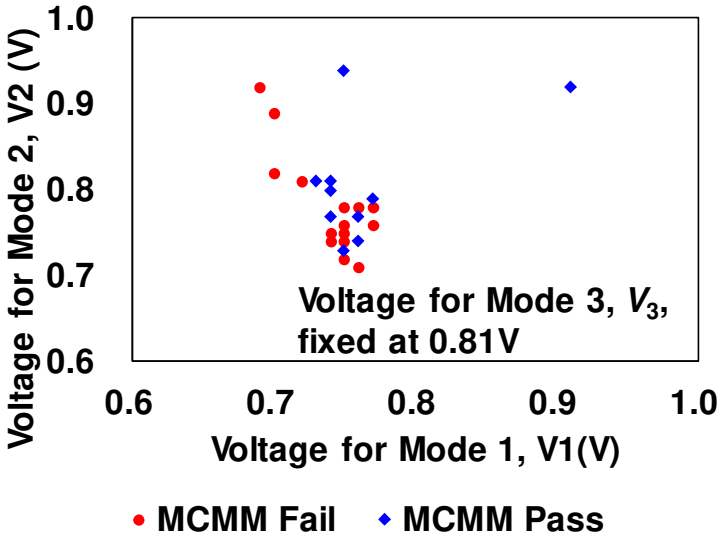


(b)

Figure 3.10: Convergence plots of the optimization for (a) two-mode case (1, 3), and (b) three-mode case (1, 2, 3) starting from two different initial points.



(a)



(b)

Figure 3.11: Timing closure results at various sets of voltages for individual modes. (a) two-mode case (1, 3). and (b) three-mode case (1, 2, 3) where the voltage for Mode 3 is fixed at 0.81 V.

3.6 Conclusion

This chapter proposed a design methodology based on ASA to achieve a design applying to mode-wise voltage-scaling (MWVS) with guaranteeing no timing errors. This work formulated the MWVS design as an optimization problem toward the minimized energy operation by defining operation voltages for individual modes and cell sizes and Vt types as the variables. The proposed method integrated ASA with the MCMM flow in EDA tools, and then applied DSA to solve the problem numerically. This work also introduced a fine-grained identification method of false paths that can be excluded in timing optimization without any risk of timing error. The evaluation based on RISC-V design achieved 20% gain of power efficiency compared with the conventional VS method, where 15% gain comes from the mode-wise idea. It is also mentioned that the fine-grained false path identification facilitated the timing closure and reduced leakage power by more than 30%. Though this study deploys CPU as the platform, GPU-like design should enjoy the benefit from MWVS. The mode-wise concept applies to GPU since GPU applications range, for example, from GPU-based advanced encryption standard (AES) [105] to the popular DNN training. Applying the proposed MWVS scheme to GPU-level applications could be the future work.

Chapter 4

DNN Training with Adaptive-Bit-width-and-Voltage-Scaling (ABVS)

This chapter proposes a DNN training scheme which integrates the approximate computing (AC) and voltage scaling (VS) techniques to strengthen energy reduction, which is called adaptive bit-width and voltage scaling (ABVS). ABVS starts the training from less fraction bit-width (FB) and gradually increase FB regarding the present training quality such as accuracy and mAP. With the fact that shorter FB can achieve shorter latency, the length of FB in hardware is manipulated to functionally operate at each minimum acceptable voltage (MAV) with no timing issue. Therefore, the approximation in this scheme induces the truncation error due to less FB in early training stage but no timing error even at the scaled voltage. Experiments with couple of popular public datasets confirms the efficacy of the proposed ABVS-based training. This chapter also introduces the training scheme to tackle new unknown dataset with ABVS.

4.1 Introduction

Many modern applications such as large-scale image classification, object detection, or speech recognition, conveniently, apply DNN to solve those problem. These difficult problems tend to require large scale of dataset and sophisticated model, e.g., very deep layers to be used in DNN, resulting in significant computation cost. Different from inference phase, training phase needs to perform more computations due to:

- Operations for back-propagation and parameter updates
- Ever large scale of training set to strengthen generality
- Several epochs (iterations) to stabilize the model quality

To concurrently approach the most promising solution in reasonable run time, GPU is widely used to train modern DNN model, and 32-bit floating-point (FP32) is the default choice. More complicated problems requiring so-called Big Data mining such as weather prediction [106] or E-commerce [107] further need cloud computing to utilize extensive capability for computation. Since it is claimed that the training for modern DNN models are apt to be susceptible to model inaccuracy [55, 68], the energy-beneficial AC techniques may hinder improving training quality, and degrade the final training quality. Therefore, most of the AC strategies focus on inference rather than training. Besides, in conventional training, the AC techniques are applied to forward propagation only, and back propagation relies on exact computation [32, 38, 65, 66, 69, 70] or additional training stages are required [32, 65, 66]. Although the inference and forward propagation enjoy AC benefits, the efficiency improvement of back propagation in training has less explored despite its importance.

On the other hand, due to back-propagation and weight updates, there are more parameters generated temporarily in training phase, blocking the memory resource. To address this issue as well, bit-width scaling (BWS) is considered to be a promising recipe that can effectively save the memory while smoothly degrading, and thus BWS is widely used in training acceleration. Assuming the conventional FP32 (8-bit for exponents and 23-bit for fractions) is more than necessary, training with a shorter format is explored for training efficiency improvement, and hence FP16 (5-bit for exponents and 10-bit for fractions) or even parts of FP8 are investigated in training [33, 34, 35].

However, as mentioned in Section 1.6.1, there is a concern that the FP16 format may not have representation capability enough for modern DNN training. For certain public datasets, researchers can accommodate DNN training into FP16 but many sophisticated tricks and strategies are demanded [33, 34, 35] and partial computations in FP32 are inevitable [33, 34]. If there is a new unknown dataset that need to be tackled, it is not clear whether we can rely on FP16. Moreover, it is considered very hard to balance the quality and the efficiency with such a limited data format, i.e., FP32 and FP16, and therefore, configurable hardware with FB adjustability could be an alternative paradigm for both efficient and qualifiable DNN training.

This chapter proposes an adaptive bit-width and voltage scaling (ABVS) scheme to achieve energy minimization for DNN training. The key idea is to adopt less FB in the early training stage while the FB is gradually increased depending on the present training quality. This ABVS scheme not only mitigates the computations during training but also saves energy with the hardware architecture supporting configurable FB and voltage scaling. Since the computation with less FB reduces the complexity of the hardware engine and shortens the latency, the extra timing margin can be used for voltage scaling to save power consumption further. Several datasets across different applications, such as CIFAR-10, CIFAR-100, and ImageNet, in image classification, and Pascal VOC in object detection, are used for validating the proposed ABVS scheme.

The contributions of this chapter are:

Table 4.1: Existing works applying AC techniques. FP/BP: forward/back propagation. VS: voltage scaling

Ref.	[32, 65, 66, 69]	[38, 70]	[33, 34, 35, 55, 68]	ABVS
FP-AC	✓	✓	✓	✓
BP-AC			✓	✓
VS		✓		✓

- This work demonstrates that ABVS can achieve comparable training quality (0%-0.5% accuracy loss) against FP32 in DNN training for various datasets in size. This work also provides a guideline, to tackle with a new unknown dataset.
- The hardware evaluation in this work shows that ABVS can achieve energy reduction by 9% to 37% than fixed-bit-width training even with "least sufficient FB" without extra iterations (epochs). If comparing to common FP32 rather than least sufficient FB, ImageNet and Tiny YOLO + Pascal VOC enjoy 57%-62% energy reduction.
- This work discusses why the proposed ABVS is effective in DNN training. Furthermore, this work conducts a comparative study on different rounding methods: conventional round-to-the-nearest-even (RTNE) and round-to-zero (RT0), taking into account the hardware latency, possible voltage, scaling, and training quality. The results reveal that RTNE essentially provides a better trade-off between training quality and energy efficiency.

The rest of this chapter is organized as follows. Section 4.2 reviews previous works that inspire the ABVS concept. Section 4.3 describes the proposed ABVS scheme. Section 4.4 provides experimental results based on software and hardware evaluation through several case studies. Section 4.5 concludes this work.

4.2 Previous Work

Although several researchers apply BWS and voltage-scaling (VS) or even both to DNN, the application strategies are different. Table 4.1 briefly categorizes the existing works. All the works in list already involve BWS or approximate arithmetic unit. The AC techniques are applied to only forward propagation in [32, 38, 65, 66, 69, 70] while the back propagation still relies on FP32 to compensate the approximation error induced in the forward propagation, and [32, 65, 66] execute additional training stages for the compensation. Besides, the general purpose of [32, 38, 65, 66, 69, 70] is to realize an efficient inference engine exploiting approximations for forward computation. Efficiency improvement of back propagation in training is beyond their interests. On the other hand, references [33, 34, 35, 55, 68] address efficiency improvement including back propagation. References [33, 34] adopt training with FP16 and [35] even conducts

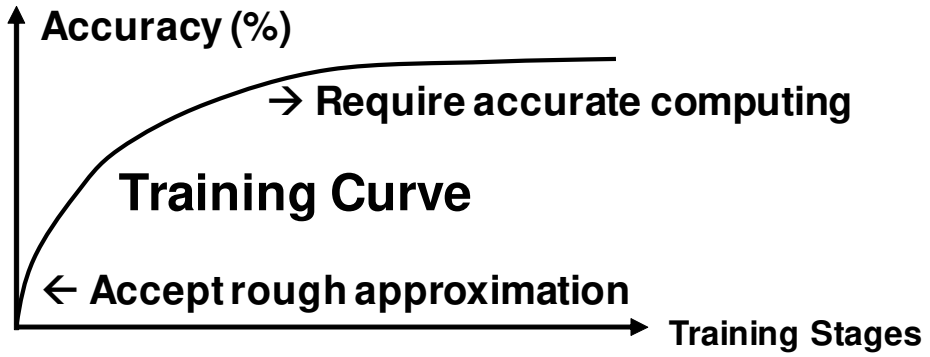


Figure 4.1: Concept of gradual training approximation (GTA).

training with FP8 partially. To enable training with FP16 or even less, sophisticated efforts are required and [35] even demands computations in FP32 partially.

Researches [55, 68] propose a gradual training approximation (GTA) scheme in Fig. 4.1, inspiring us to develop ABVS. They claim that the early stage in training accepts rough approximation while the late stage requires accurate computing. However, their works focus on the approximation for multipliers only and the accumulators remain FP32 accurate type, where FP32 accumulators would spoil VS efficiency. Besides, large-scale datasets and sophisticated applications are not included in their validations. In general, the proposed ABVS scheme is inspired by GTA and this chapter further integrate VS and the adaptive BWS to enhance training efficiency.

4.3 Proposed ABVS Scheme for DNN Training

The proposed ABVS scheme adopts less FB in the early training stage while the FB is gradually increased depending on the present training quality. The proposed scheme supposes a configurable FP hardware unit whose FB can be dynamically changed. Furthermore, the configurable FP unit operates at the minimum voltage at which the FP multiply-accumulate (MAC) result is correct for each FB configuration. In this case, the computation with small FB saves power thanks to fewer signal transitions in the FP unit and lower operating voltage. For minimizing training energy, this scheme should keep the FB as small as possible throughout the training process while achieving high-quality training.

Fig. 4.2 illustrates the procedure of the proposed ABVS scheme. The scheme start the training with a pre-determined smallest FB denoted as FB_{min}, and set the initial A_{check} to 0. For every epoch, the training engine would provide a metric for estimating the training quality, e.g., classification accuracy, and then the scheme assign it to A_{cur}. Then, at a certain epoch assigned for checking quality improvement, the scheme compares the

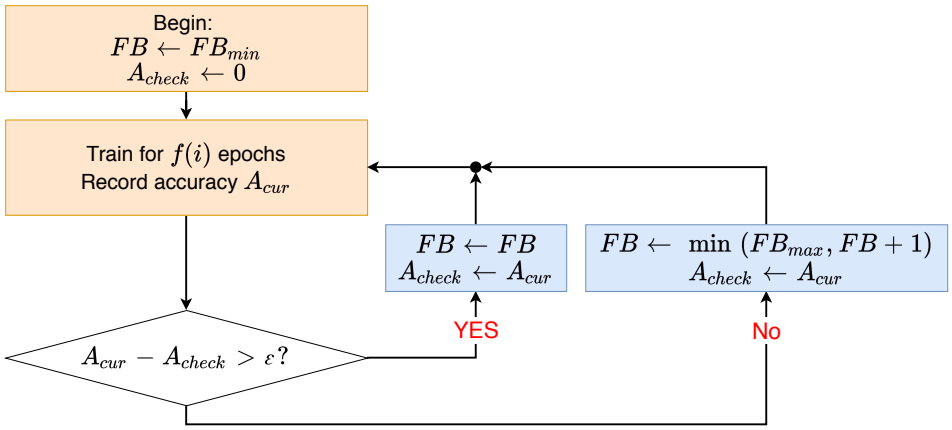


Figure 4.2: ABVS flow chart. Training finishes when i reaches T, but this process is omitted in the figure.

latest A_{cur} with A_{check}. If the difference is smaller than ε, the scheme increases the FB by 1 for the next epoch, until reaching the max pre-determined FB, denoted as FB_{max}. $f(i)$ is a function that determines the schedule of quality checking and gives the checking interval, where i represents the current epoch number. In a simple case, $f(i)$ is constant. T is the total epoch count given to training. This scheme is independent of the training engine architecture while the amount of voltage scaling depends on the architecture and circuit implementation.

When there is preliminary information on the training dataset, the least sufficient FB might be known, where "least sufficient FB" is the FB that can achieve the same quality as FP32. In this case, the least sufficient FB can be assigned to FB_{max}. For a new unknown dataset, on the other hand, the least sufficient FB is unknown. A guideline for this case is to enlarge the range between FB_{min} and FB_{max}, e.g., $FB_{min} = 6$ and $FB_{max} = 23$. As for the checking schedule, when $FB_{max} - FB_{min}$ is large, frequent checking is necessary to make sure FB_{max} is reachable during the training. On the other hand, when $FB_{max} - FB_{min}$ is small, sparse checking is desirable since sparse checking prevents unnecessary FB elevation originating from the noisy metric trend. Considering this tendency, this scheme may suggest

$$f(i) = T/(\alpha \times (FB_{max} - FB_{min})), \tag{4.1}$$

where α is a tuning coefficient. The appropriate α value is experimentally discussed in Section 4.4.2.

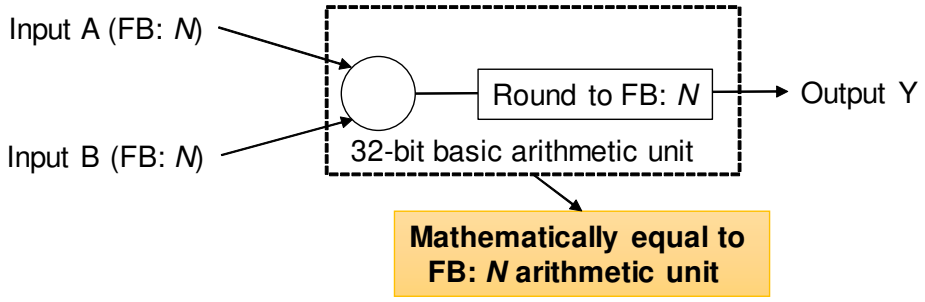


Figure 4.3: Concept of BWS implementation in QPyTorch.

4.4 Experimental Results

4.4.1 Evaluation Strategy and Experimental Setup

This section evaluates the reduction of energy for training in the following two steps. The first step is to check whether adaptive FB scaling keeps the training quality and how much FB can be reduced in the training process. This evaluation is performed with software emulation in Section 4.4.3, where the emulation method is explained below. The second step, in Section 4.4.3, estimates the energy reduction supposing a FB configurable FP unit in which the same FP format is shared by multiplication and addition.

The experiments apply the proposed ABVS scheme to a framework called Darknet [12]. Darknet is one of the most popular frameworks for image classification and object detection. An advantage of Darknet is that it supports GPU acceleration, and it is easy to apply in-depth modifications since it is fully developed by C and CUDA based programming. Therefore, it is easy to implement the BWS rounding algorithm in it and enjoy the GPU acceleration through Darknet environment.

In the experiments, adaptive FB scaling is emulated such that the rounding is applied after each basic floating-point computation, where this implementation is leveraged by QPyTorch in [108] and the schematic is shown in Fig.4.3. With this manipulation, this evaluation can perform the computations that fully reproduce the hardware behavior at any arbitrary FB on GPU. Thus, it is claimable that this training result is identical to that obtained with the proposed scheme. In the following, the round-to-the-nearest-even (RTNE) rounding method is applied. A comparison with a more straightforward rounding method will be presented in Section 4.4.4.

Fig. 4.4 and Table 4.2 show the DNN structures and the datasets used in the experiments. CIFAR-10, CIFAR-100 [109], and ImageNet [9] are for image classification, and Pascal VOC [110] is for object detection. The DNN structure used in the experiments for CIFAR-10 is the one recommended by [12], and the same structure is used for CIFAR-100. The selected DNN structures for ImageNet and Pascal VOC are named Darknet reference [12] and Tiny YOLO [111]. Tiny YOLO is constructed based on

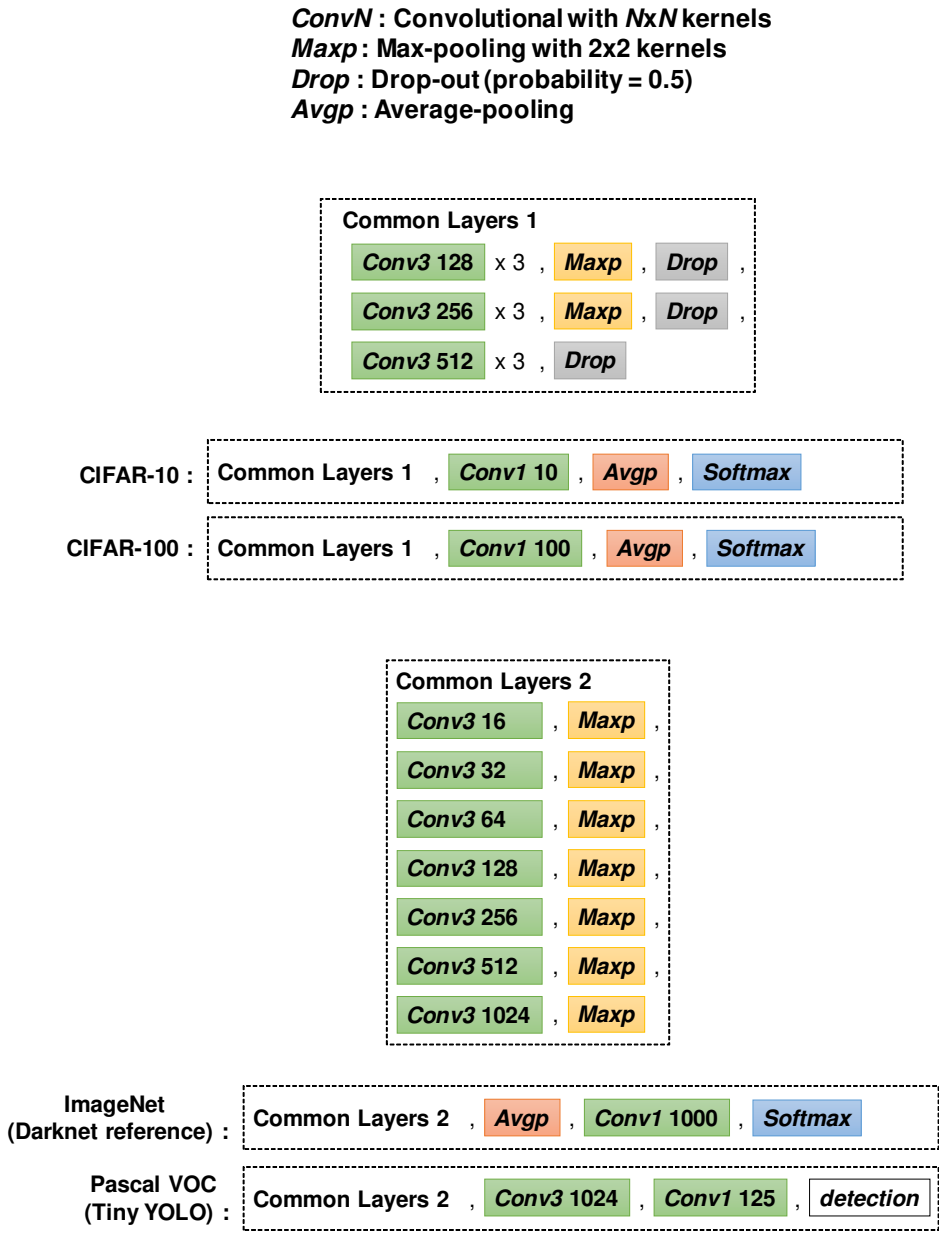


Figure 4.4: DNN structures used in the experiments.

Darknet reference, where most of the composition of the convolutional layers are identical, and only the last layers are replaced from classifier to detector. The developer of Tiny YOLO adopts transfer learning to improve the training quality of YOLO. Therefore, the weights are initialized in the convolutional layers using the pre-trained weights for ImageNet.

All training cases are performed with 50 epochs in total. Learning rate decay is ap-

Table 4.2: # of images in training and testing sets.

Dataset	Training set / Testing set
CIFAR-10 & CIFAR-100	50000 / 10000
ImageNet	1281167 / 50000
Pascal VOC	16551 / 4952

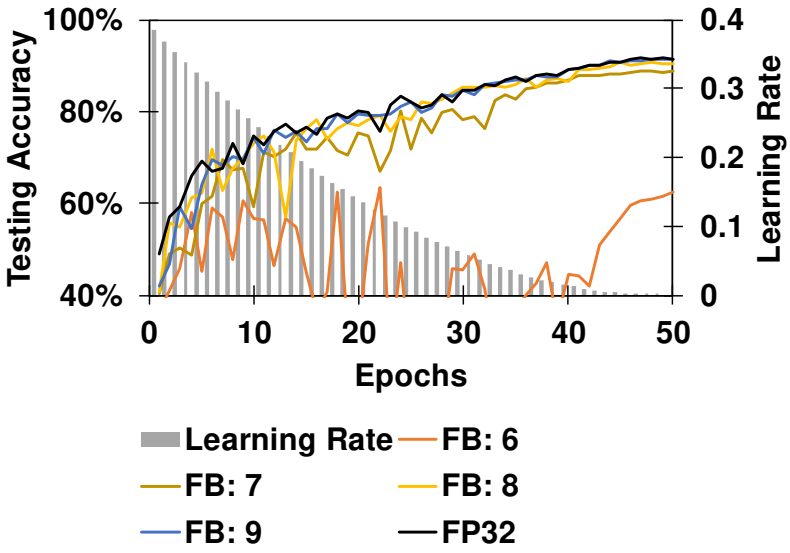
plied with a poly-nominal formula, $r_{init} \times (1 - i/50)^P$, where r_{init} is the initial learning rate and i is the current epoch. P is set to 2 for image classification and 1 for object detection. The learning rate becomes 0 once it reaches $i = 50$. Therefore, with this setup of the learning rate, training beyond 50 epochs is meaningless. For applying the ABVS scheme, The "accuracy" is adopted as the metric of training quality and $\varepsilon = 0.005$ for image classification, and the "mAP" and $\varepsilon = 0.01$ are utilized for object detection. Both accuracy and mAP are computed for the testing datasets. In most experiments, FB_{max} is set to the associated least sufficient FB. With this setup, this experiments can demonstrate that the ABVS scheme reduces computation and energy further even compared with the baseline training with the least sufficient FB.

For reproducing the situation that prior information is available for each dataset, this work trained the NNs with various FBs as preliminary experiments. Fig. 4.5(a) and Fig. 4.5(b) show the training curves of CIFAR-10 and CIFAR-100, respectively, where FB is swept. The left Y-axis shows the accuracy of the testing dataset, and the right Y-axis indicates the learning rate. The results show that CIFAR-10 requires at least 9 bits for fractions while CIFAR-100 may need 10 bits to reach the same quality level as FP32. Similar experiments are performed for other datasets. The obtained FB information is used to determine FB_{min} and FB_{max} for the experiments in the following sections.

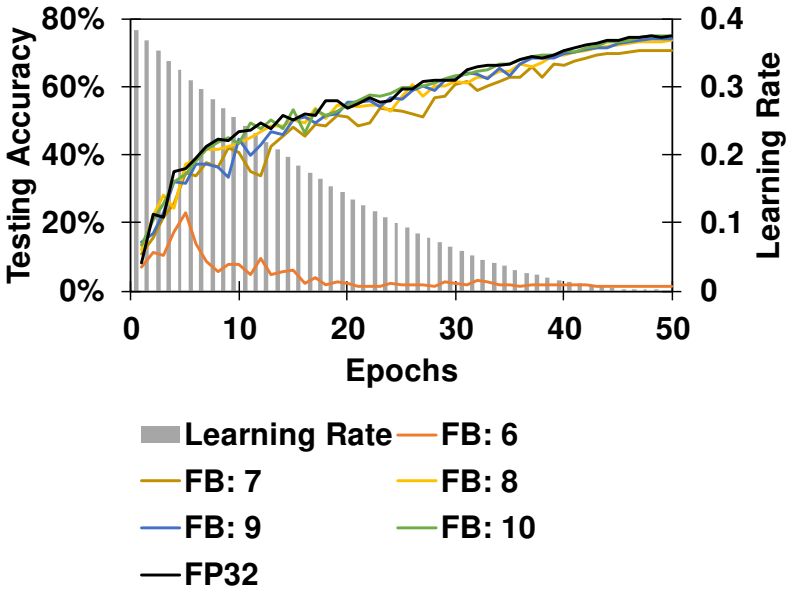
4.4.2 FB Reduction by ABVS

Fig. 4.6 and 4.7 show the results of the proposed ABVS scheme for CIFAR-10 and CIFAR-100, respectively, with two checking schedules described above the figures. Here, FB_{max} is set to 9 for CIFAR-10 and 10 for CIFAR-100, and the FB_{min} is 6 in both cases. The FB varies along the training stage, which corresponds to the right Y-axis. The results present that the duration where FB is lower than 9 in Fig. 4.6(b) is longer than in Fig. 4.6(a). Then, the average FB across the entire training process in Fig. 4.6(b) is 7.66 and smaller than Fig. 4.6(b) despite the second schedule decreases the testing accuracy by 0.3%. Similar observations are found in the results of CIFAR-100 of Fig. 4.7. Smaller average FB involves a small penalty of the accuracy, but the drop is not significant. Even while the training in Fig. 4.7(b) ends with 9 bits, the accuracy difference is only 0.4%. The energy reduction obtained from a smaller average FB will be discussed in the next section.

Next, consider the situation that the dataset is new and unknown. The training with ABVS is carried out using $FB_{min} = 6$ and $FB_{max} = 23$. Fig. 4.8 shows the results



(a)



(b)

Figure 4.5: Trainings w/ fixed FB for (a) CIFAR-10 (b) CIFAR-100.

for CIFAR-10 and CIFAR-100 with different checking schedules. You can see that checking for every two or three epochs works well since the training quality is the same

or almost comparable as FP32 one while the FB is smaller. These results indicate that α is between 1 and 2. This work would suggest such $f(i)$ in Eq. (4.1) to apply ABVS for training a new dataset.

ImageNet: Next, this work evaluates ABVS on ImageNet [9]. Fig. 4.9 shows the results, where FB_{min} and FB_{max} are set to 11 and 14, respectively. The training curve of ABVS traces that of FB = 14, and 0.5% accuracy loss is considered acceptable. The result shows ABVS works even for a large-scale dataset.

Pascal VOC: Figs. 4.10(a) and 4.10(b) demonstrate the results of the first trial with different checking schedules, where FB_{min} and FB_{max} are 10 and 12, respectively. Both results are similar to those for other datasets, but the mAP degradation of 0.5% might be larger compared with other cases. It is found that the maximum mAP of YOLO may not appear at the end of iterations but in the middle. This mAP gap is found even with other checking schedules.

Besides, adding a warmup stage, which increases the learning rate and locates at the beginning of training, is proposed by [25] to achieve better training, and it is adopted in YOLO training. This work empirically found that applying larger FB during the warmup stage in YOLO training improved mAP, as shown in Fig. 4.11. The left chart shows the given learning rate. The maximum mAP improved and is comparable with that of training with 12 bits of FB.

Discussion: Let us discuss the reason why ABVS achieves a similar training quality with fewer FB. K. You, et al. point out that a large learning rate in the early training stage perturbates the training, which prevents the network from memorizing noisy data and results in better generality [112]. On the other hand, BWS injects noise originating from the FB truncation error, especially in the earlier stage with smaller FB. Meanwhile, the large learning rate at the beginning may tolerate larger noise, which provides high compatibility with ABVS.

Through the case studies above, this work has confirmed that the proposed ABVS scheme is promising in various scales of datasets across different applications. The next section evaluates power reduction using the results shown above.

4.4.3 Energy Reduction by ABVS

This section evaluates the power reduction thanks to ABVS. This work supposes, in DNN training, the majority of power consumption comes from MAC computation, especially in convolution layers. Thus, this work prepares a hardware unit with configurable FB, which is shown in Fig. 4.12 The input of "FB mode select" determines the FB for MAC operation. The hardware unit is implemented it with verilog and synthesized with Nangate 45nm VTG cell library [92] at 1.0 GHz by Synopsys Design Compiler.

An important design consideration is that the FP-MAC should be able to operate at the lowest voltage for each FB configuration. For this purpose, the FP-MAC consists of separate circuits with different FBs that are synthesized individually at the lowest

voltage achieving the same operating frequency. Fig. 4.13(a) shows the architecture of each FP-MAC, where $N = FB + 9$ since sign and exponent bits are also included. To include optimized floating-point multiplier and adder modules, this work used Synopsys DesignWare IP in synthesis. The minimum acceptable voltage (MAV) for each N-bit FP-MAC is defined as the voltage at which the circuit can be synthesized with 0 worst negative slack (WNS). The results are shown in Fig. 4.13(b). The MAV ranges from 0.85V to 1.02V.

For energy estimation, a test input pattern is prepared representing convolution computation. The power of each N-bit FP-MAC is reported by Synopsys PrimeTime using the logic simulation result. Then, the training energy is estimated as

$$Energy = T_{epoch} \times (\sum_{i} (P^{(i)} \times N^{(i)}_{epoch})), \tag{4.2}$$

where T_{epoch} is the computation duration for one epoch, $P^{(i)}$ is the power of i-bit FP-MAC and the number of epochs in which i-bit FP-MAC is applied (denoted as $N^{(i)}_{epoch}$).

Fig. 4.14(a) shows the energy reductions, which correspond to Figs. 4.6, 4.7, 4.9, and 4.11. The values in Fig. 4.14(a) are normalized by the energy consumed by the training with the least sufficient FB. The results show that even comparing with the training with the least sufficient FB, the proposed ABVS can achieve 9% to 37% energy reduction.

Let us assume another case that only either of FP16 and FP32 is choosable. In this case, the proposed ABVS scheme provides larger values of energy saving. Fig. 4.14(b) shows the energy reduction, where the energies of CIFAR-10 and CIFAR-100 are normalized by those of FB = 10 (same precision as FP16), and the energies of ImageNet and YOLO are normalized by those of FP32. The ABVS achieves larger energy reduction. Especially for ImageNet and YOLO, 57%-62% energy reduction is achieved. On the other hand, if the voltage scaling is not applied, i.e., only BWS is applied, the energy reduction becomes less, as shown in Fig. 4.14(a). There is a 15% difference in CIFAR-100. Thus, simultaneous FB and voltage scaling in the proposed scheme are effective.

Finally, this section demonstrates the energy reduction for CIFAR-10 and CIFAR-100 with unknown dataset treatment. Fig. 4.14(c) shows that the proposed ABVS can reduce training energy by 25% to 69% even for unknown datasets.

Although this configurable design provides significant energy saving with BWS and VS, it is the fact that the circuits with different FBs implementing in parallel are too naive and would result in huge penalty of area. Compared with a solo 32-bit FP-MAC design, the area overhead of the implemented configurable unit is 10X. Note that this area overhead could be exacerbated if considering additional margin to accommodate manufacturing and environmental variations. On the other hand, multiple voltage regulators, multiple voltage rails or level shifters are not necessary since the entire circuit is operating at the same voltage. The possible way to supply the voltage is with a small

controller to control the flow in Fig. 4.12 and determines the number of FBs. Also, each FB is associated with the operating voltage, which might be stored in a look-up table. In this case, referring the table, the controller can control the operating voltage. When constructing the table, the environmental variation might be considered. On the other hand, the evaluation in the dissertation did not consider the environmental variation since it is applicable to any design, i.e., not unique to the configurable FB design, and hence it is beyond the scope of this work. A more sophisticated and area-efficient configurable FP-MAC design methodology is intended to be the future work, where a possible design method is going to be introduced in Section 4.4.5.

4.4.4 Rounding Methods

Lastly, this chapter discusses the rounding method suitable for the proposed ABVS. IEEE 754 adopts round-to-the-nearest-even (RTNE) as the rounding method. On the other hand, rounding requires more operations and thus results in longer timing paths, which prevents voltage scaling. Instead, round-to-zero (RT0) is the most hardware friendly method, and it might achieve more power saving from voltage scaling. Hence, this work evaluated the MAVs for RTNE and RT0, which are plotted in Fig. 4.15. RT0 can roughly reduce the MAV by 25 mV compared with RTNE.

Next, the impact of rounding methods on the training quality is evaluated. Fig. 4.16 shows the training curves for CIFAR-10 and CIFAR-100 with different rounding methods. The results show that RT0 requires 2 or 3 more bits to achieve comparable training quality against RTNE. On the other hand, about 2- to 3-bit difference corresponds to more than 30 mV in operating voltage. These results indicate that RTNE provides a better energy-quality trade-off, and it is superior to RT0. Therefore, this chapter applied RTNE to all the experiments except in this section.

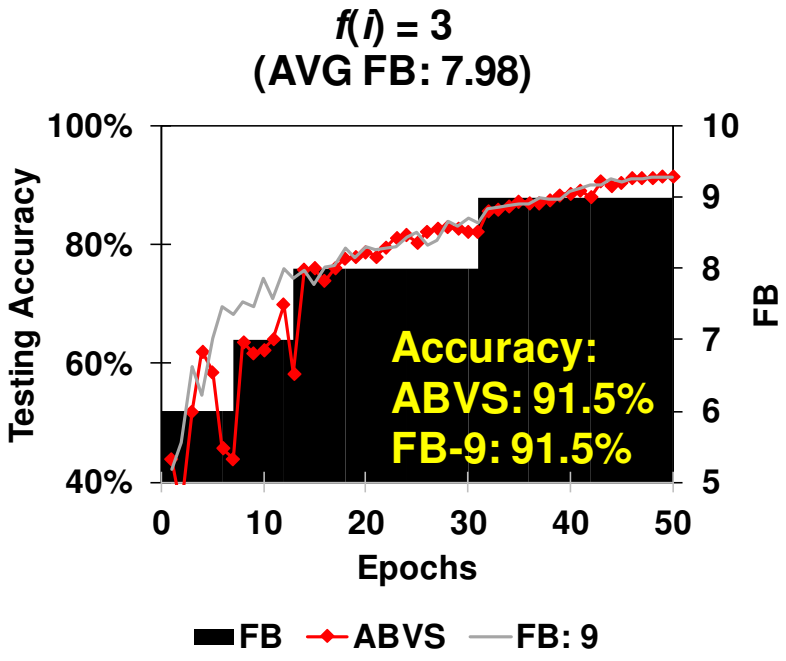
4.4.5 Potential Solution for More Area-efficient FB-configurable FP-MAC Design

This section introduces an implementation approach that might be available to acheive FB-configurable FP-MAC design with less area overhead.

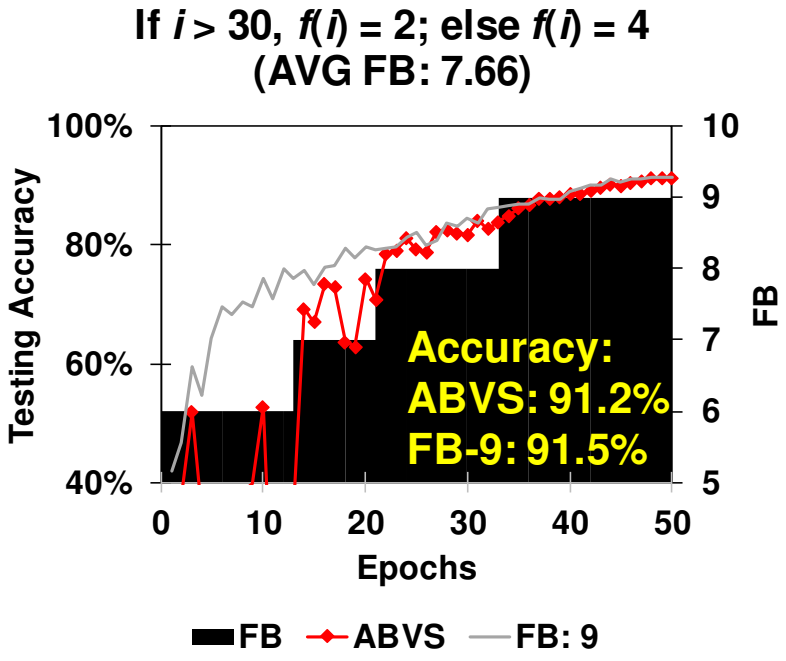
The main idea is to apply round-to-the-nearest (RTN) scheme instead of RTNE scheme, where RTN relies on fewer bits during rounding process. This difference enables shorter latency and facilitates FB configurable design, which will be explained later. Fig. 4.17 shows the comparison of the algorithms between RTNE and RTN. The first column in the truth tables shows the rounding results, where the rounding is performed between the m-th and (m-1)-th values. You can see the RTNE uses the m-th to 0-th values whereas the RTN depends on only the $(m-1)$-th value. There is one row that outputs different values between RTNE and RTN. The condition of the different outputs is that the m-th value is 0, $(m-1)$-th is 1, and all the values from the $(m-2)$-th to 0-th

are 0. The occurrence probability of this condition is rare, and hence it is conceivable that training results adopting RTN could yield similar results as that adopting RTNE. Fig. 4.18 shows that RTN performs almost the identical results as RTNE. Therefore, RTN can be exploited in the new FB-configurable FPU design.

From a circuit design perspective, RTN can simplify the structure and has a higher compatibility with the configurable design. Fig. 4.19 illustrates the scenario of new design. The new design is basically a 32-bit FP design. No matter what the FB mode is specified and some bits in input A and B are masked, the calculation is performed based on this design and then the calculated result is rounded according to the specified FB. In order to achieve larger VS, the paths starting from the FFs representing different bits of Input A and B are manipulated to have different timing margins, where the achievable margin could vary with the bits. In theory, the paths from the bits getting closer to the most significant bit (MSB) are expected to have larger achievable timing margin. Therefore, when the number of FB becomes smaller, the larger timing margin brings larger VS without timing errors. Since RTN has less function complexity compared with RTNE, more timing margin can be squeezed. Here, RTNE needs to consider the 0-th bit whatever the FB-mode as discussed above, where the paths starting from 0-th bit are thought to be the most timing critical. Therefore, RTN is expected to provide larger VS. This design basically includes one 32-bit design, and then it is expected that the area overhead is significantly smaller than the current implementation shown in Section 4.4.3. The detailed implementation of the FB-configurable design is going to be a future work.

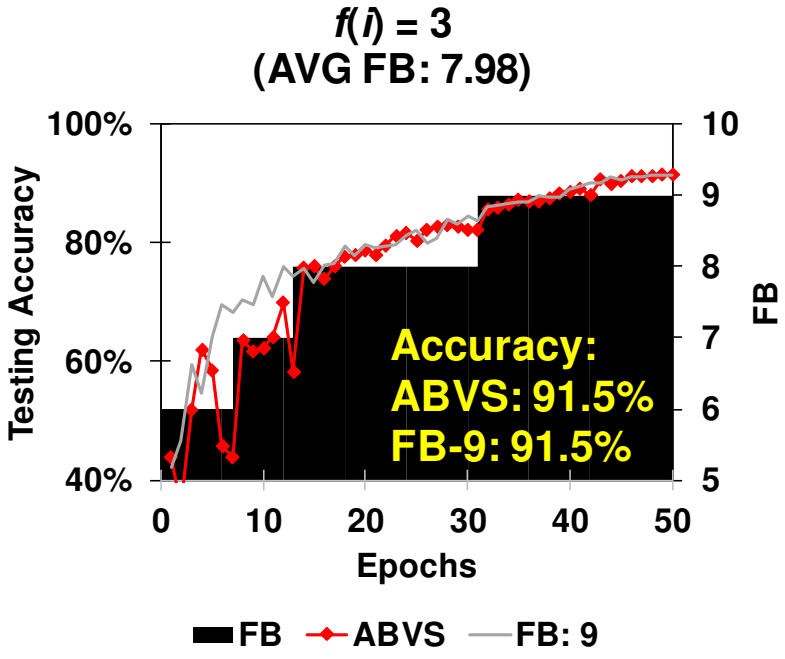


(a)

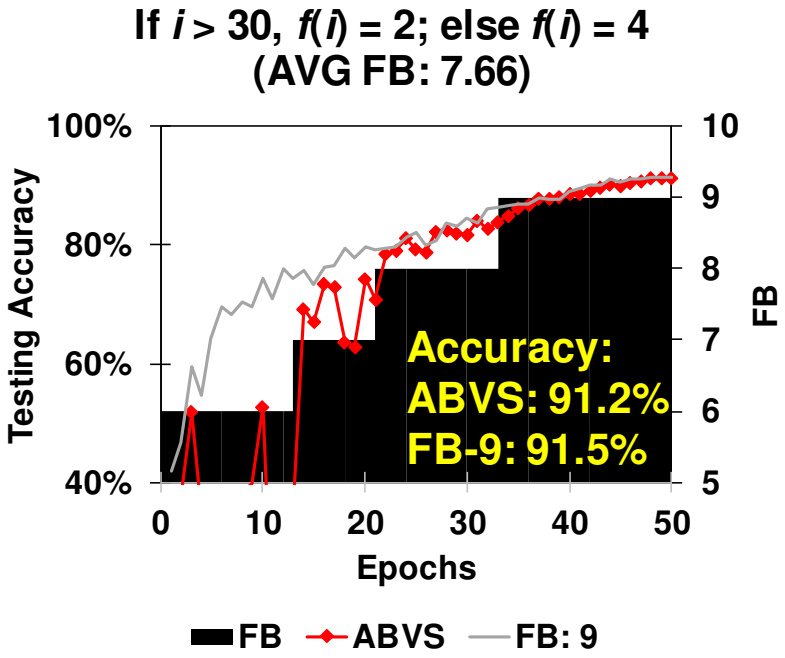


(b)

Figure 4.6: Trainings w/ ABVS for CIFAR-10 with two checking schedules.



(a)



(b)

Figure 4.7: Trainings w/ ABVS for CIFAR-100 with two checking schedules.

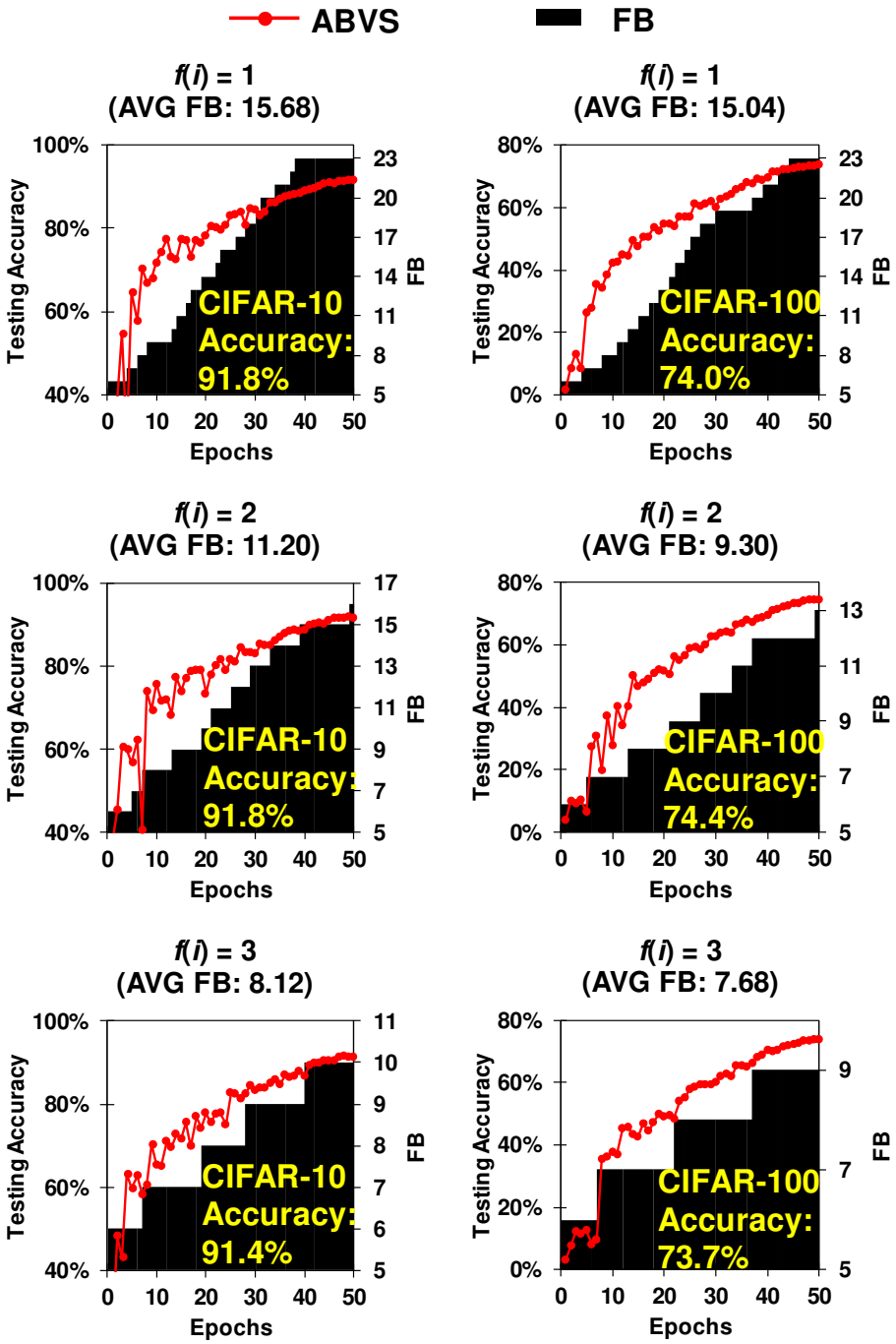


Figure 4.8: Applying ABVS flow with $FB_{max} = 23$ for CIFAR-10 and CIFAR-100.

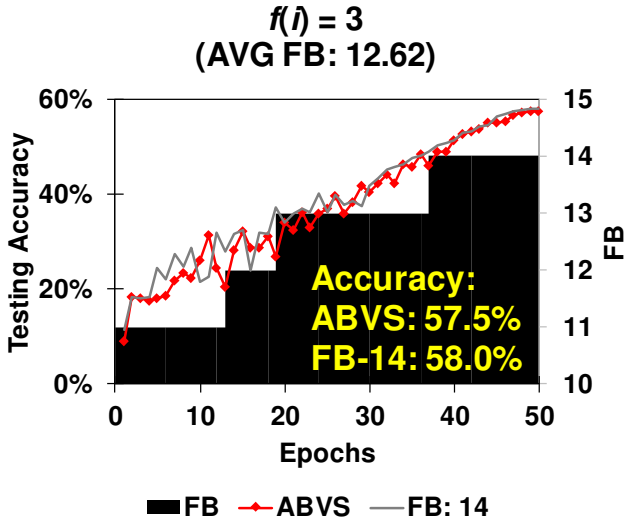
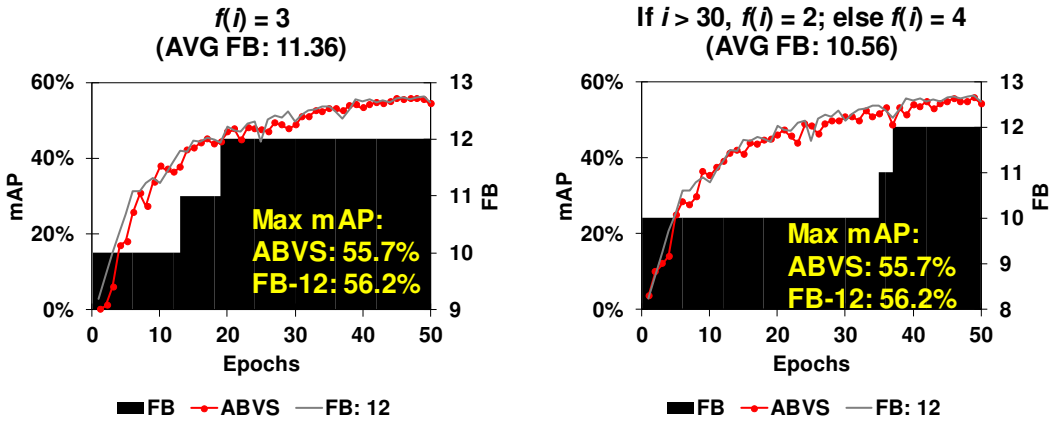


Figure 4.9: ABVS results (ImageNet).



(a) (b)

Figure 4.10: ABVS results w/ two checking schedules (Pascal VOC).

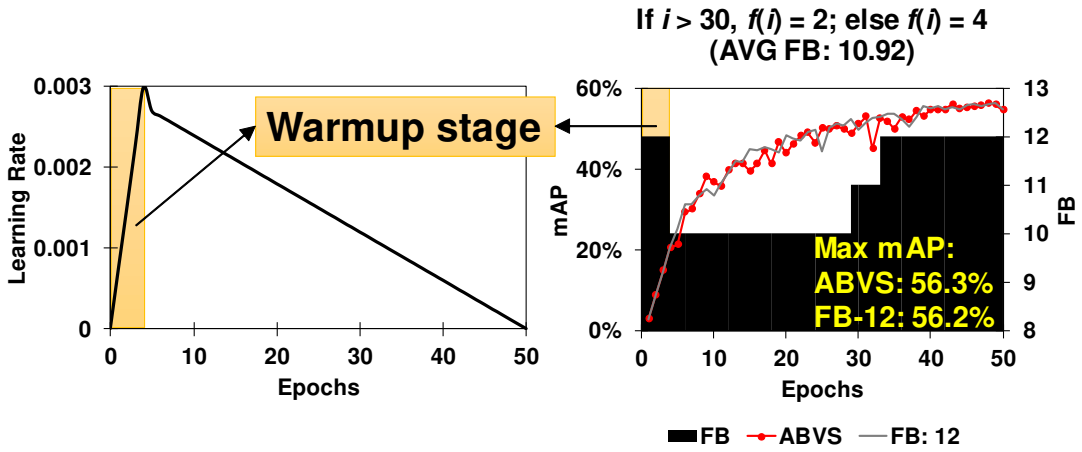


Figure 4.11: ABVS results considering warmup stage (Pascal VOC).

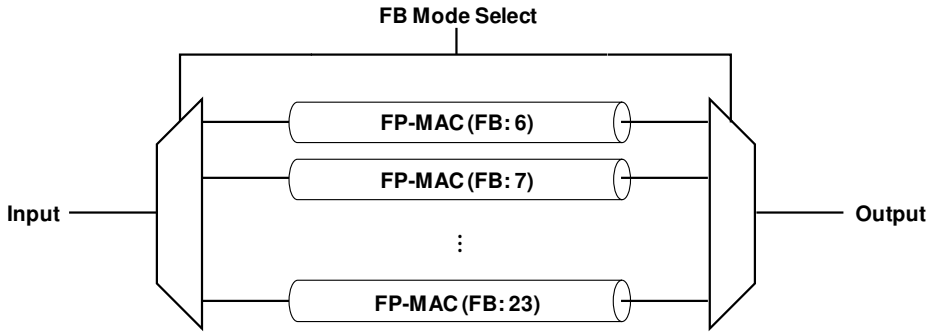
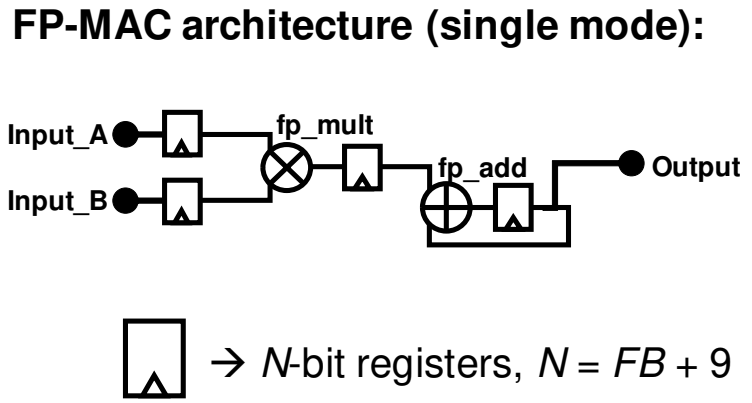
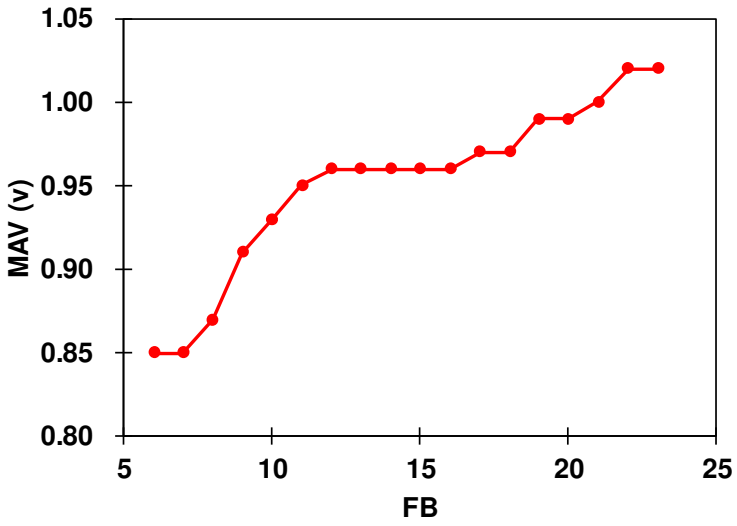


Figure 4.12: Assumed FP-MAC with FB configurability.

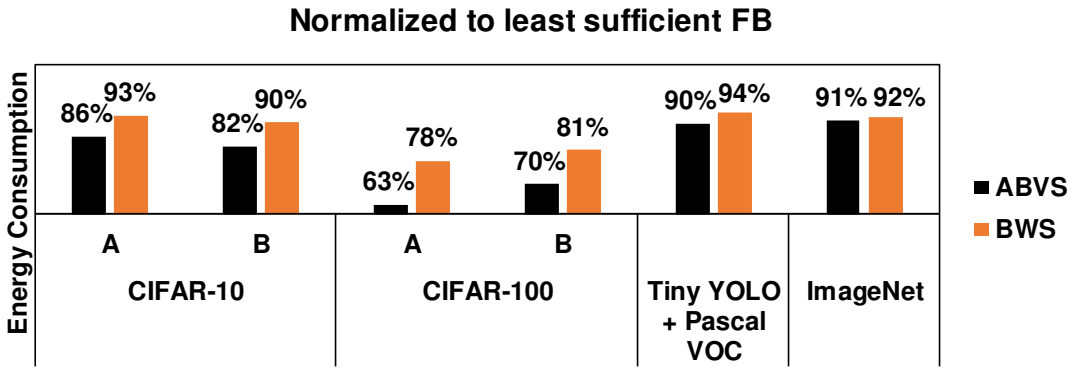


(a)

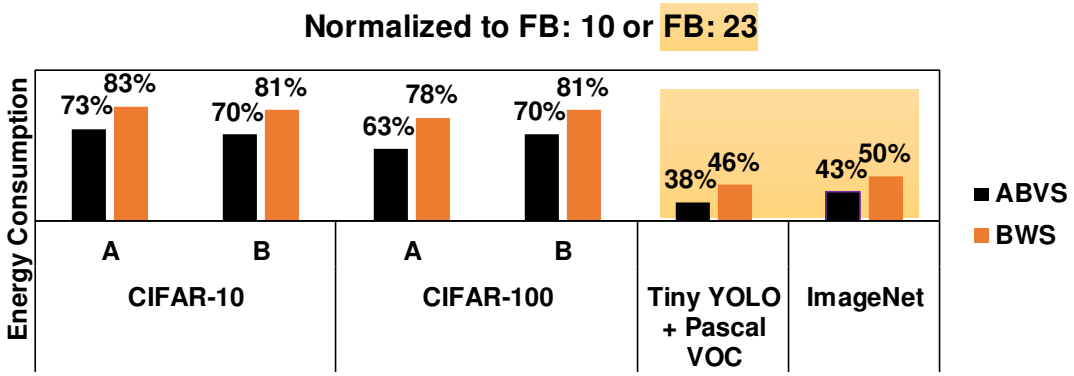


(b)

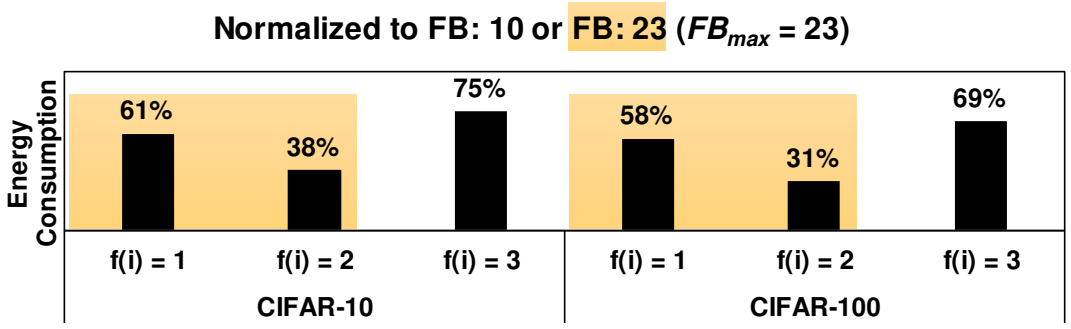
Figure 4.13: (a) Schematic of a single FP-MAC with N-bit. (b) MAV results for each FP-MAC with different FBs.



(a)



(b)



(c)

Figure 4.14: Energy of ABVS training (a) normalized by that of least sufficient FB (b) normalized by that of FB: 10 or FB: 23 (c) normalized by that of FB: 10 or FB: 23 for CIFAR-10/CIFAR-100 with unknown dataset treatment.

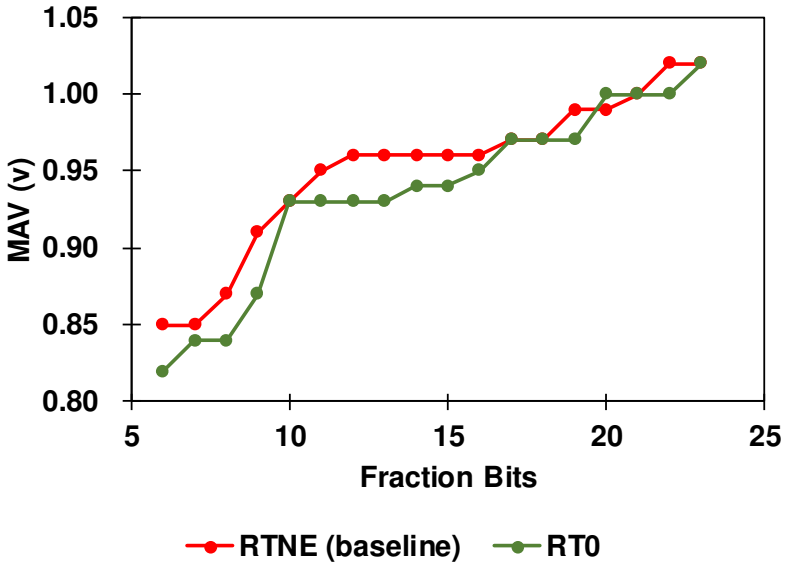


Figure 4.15: MAV comparison between FP-MACs with RTNE and RT0.

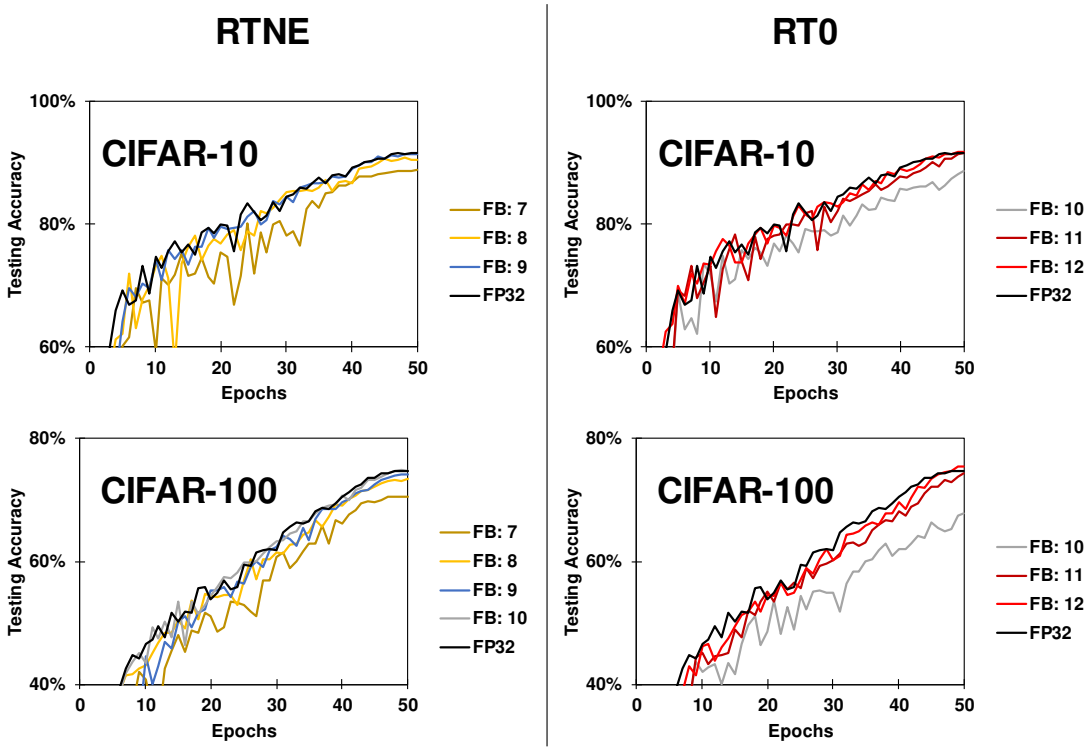
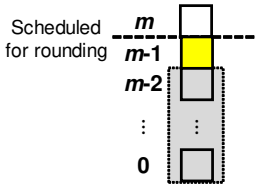


Figure 4.16: Comparison in training quality between RTNE and RT0.



Truth table to determine the results of rounding (1 or 0)

RTNE	*m*	*m*-1	*m*-2 | *m*-3 | … | 0
0	0	0	0
0	0	0	1
0	0	1	0
1	0	1	1
0	1	0	0
0	1	0	1
1	1	1	0
1	1	1	1

RTN	*m*	*m*-1	*m*-2 | *m*-3 | … | 0
0	X	0	X
0	X	0	X
1	X	1	X
1	X	1	X
0	X	0	X
0	X	0	X
1	X	1	X
1	X	1	X

Figure 4.17: Truth-table-based algorithm comparison between RTNE and RTN.

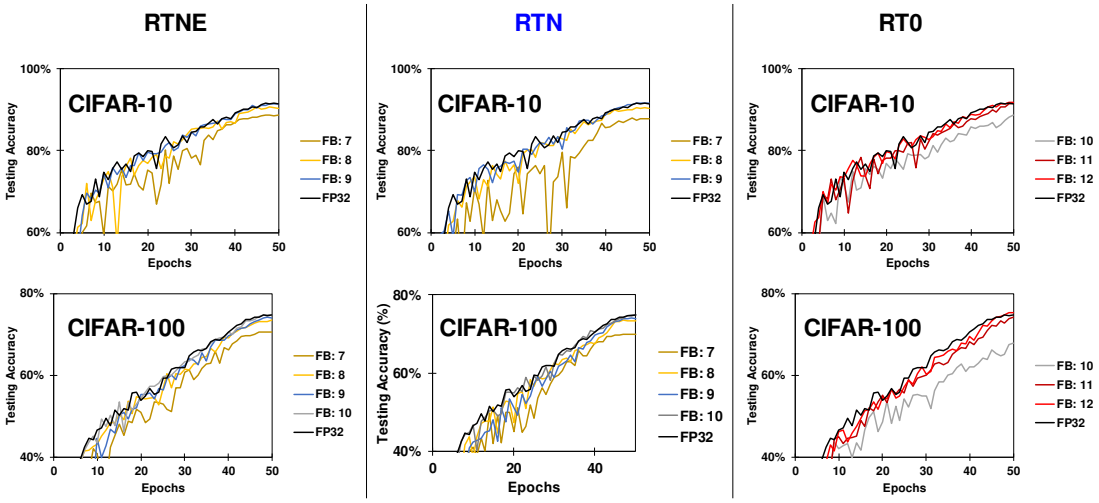


Figure 4.18: Comparison in training quality between RTNE, RTN and RT0.

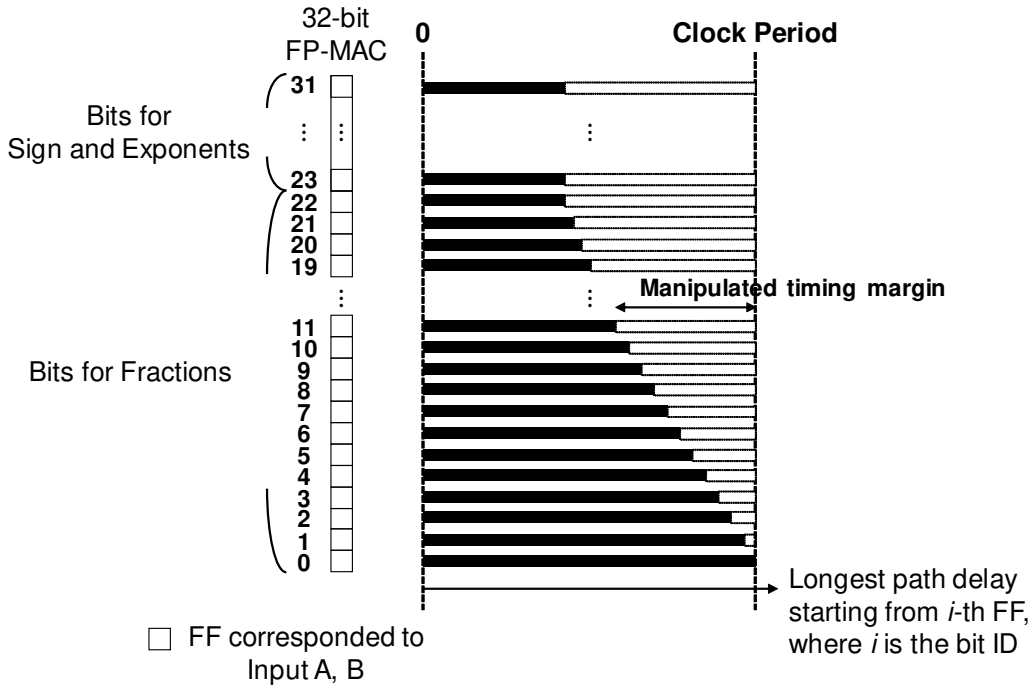


Figure 4.19: Proposal for a FB-configurable FP-MAC design with less area-overhead.

4.5 Conclusion

This chapter proposed the ABVS scheme for DNN training to minimize energy consumption. With a hardware unit with FB configurability, the proposed scheme can concurrently perform bit-width and voltage scaling during training, which enlarges energy saving. The proposed ABVS is proved to apply to various datasets across different applications with negligible quality loss (at most 0.5%) while saving at most 62% energy comparing and up to 37% energy is reduced even comparing with training in the least sufficient FB.

Chapter 5

Conclusion

This dissertation aims at providing a promising solution to achieve an efficient yet high-quality DNN training engine to facilitate both high-end GPU designs and server- or device- level designs. The main countermeasures to achieve efficient training are two fields; (1) approximate computing (AC) and (2) voltage scaling (VS). While either one of them can contribute to incredible power/energy efficiency, the two countermeasures applied concurrently provide a significant synergy effect for the energy efficiency improvement in DNN training.

For (1), AC, a LAM-based floating-point NN training is introduced in Chapter 2. LAM approximates the floating-point multiplication to fixed-point addition, which achieves shorter delay, less gate, and power consumption to mitigate primary MAC computation in NN training. Besides, LAM is highly compatible with conventional BWS technique. Even when BWS is already implemented in training, LAM is still applicable and the power efficiency can be further enhanced. In Chapter 2, there are two platforms selected to demonstrate the efficacy of LAM-based neural network training, where one is dedicated hardware (ASIC) and another one is GPU-level processor. Experimental results show that through dedicated training hardware, LAM-based training achieves 10% speed-up and 2.3X power reduction and the same ratio of area saving as well when training a 2-D classification dataset, where the improvement is compared with training with exact multipliers synthesized at the same speed. Training NNs with LAM + BWS, about 4.9X energy efficiency, where 2.2X originates from LAM, is attained while the accuracy has no more than 1.0% discrepancy, sometimes even better. This work then quantified LAM performance through an open-source GPU design and evaluated the power reduction based on FPGA hardware measurement. The experiment results confirmed 28% power efficiency improvement in the LAM-embedded GPU design compared with GPU design with regular 32-bit floating-point multiplier, and 41% for GPU design with LAM + BWS. The final experiment in Chapter 2 experimentally qualified the applicability of LAM and LAM + BWS for NN with up to 4 hidden layers, and confirmed that LAM can sustain the training quality even with aggressive BWS.

As for (2), VS is a classical yet powerful technique to achieve power/energy reduc-

tion for many low-power applications, including NN engine. ASA is a state-of-the-art technique that can enhance VS efficiency. This dissertation presents a design methodology based on ASA to achieve a mode-wise voltage scalable design with guaranteeing no timing errors in Chapter 3. The MWVS design flow can be formulated as an optimization problem toward the minimized energy operation for identifying the operation voltage for individual modes as well as the cell sizes and Vt types in the design. To solve the formulated problem, the ASA is integrated with MCMM flow in EDA tools giving the identified false paths, and DSA is applied in this work as an effective algorithm to search the optimal solution space. As the results, the design after mode-wise ASA enables mode-wise voltage scaling, and the temporal bias of the mode usage could be exploited to contribute to overall power saving. Chapter 3 also introduced a fine-grained identification method for false paths that can be applied in MWVS without any concern for timing error and glitch issues. The evaluation results based on RISC-V CPU design show that the proposed MWVS methodology can reach 13%-20% overall power gain compared with the conventional VS method, where 8%-15% gain originates from the mode-wise idea. The introduced fine-grained false-path identification also benefited the timing closure and successfully reduced the leakage power by 31%-42%.

This dissertation then introduced adaptive bit-width and voltage scaling (ABVS) scheme in training in Chapter 4, which can be considered as a naturally integrated solution for (1) and (2) to intensify energy reduction for DNN training. The BWS, one of the conventional AC technique introduced in Chapter 2 and the mode-wise voltage scaling scheme mentioned in Chapter 3 form the mechanism of Chapter 4. The main idea is to leverage a hardware unit with fraction bit-width (FB) configurabililty, and then starting from less FB, and gradually increase the FB according to the present training quality. The nature of achievable shorten latency for less FB, which is exploited in MWVS design, contributes to an efficient tuning knob of scaled voltage, and hence this scheme can concurrently perform bit-width and voltage scaling to realize efficient training. Various public datasets were chosen for validating ABVS efficacy. The negligible quality loss (at most 0.5%) yet at most 62% energy saving and up to 37% energy reduction even comparing with training in the least sufficient FB were attainable. Besides, the proposed ABVS flow can also apply to new unknown datasets even without pre-understanding its baseline accuracy. The reason why ABVS works in modern DNN training and a benchmarking for hardware quality-efficiency trade-off between two rounding methods are also addressed in Chapter 4.

Overall, this dissertation aims at providing useful techniques to realize efficient DNN training, and confirms approximate computing and mode-wise voltage scaling are applicable and effective strategies. However, the study in this dissertation is just a part of the area to achieve energy-efficient training, and many questions are still not well-investigated. Many countermeasures relevant to DNN are mainly based on empirical studies and the effects for tackling new models or new datasets always have concern. Regarding Chapter 2, the efficacy for LAM in more sophisticated DNN structures or

larger size of dataset is worthy to be evaluated. Also, training with other data representations such as fixed-point or log-domain could be involved into the benchmarking for comparing their trade-off between quality and efficiency. Besides, the performance evaluation through more recent training hardware with the FB-configurable FPU design introduced in Chapter 4 is also challenging yet worthy of further study. Developing a good hardware evaluation environment for DNN training inspires many attractive studies, such as involving other AC techniques or layer-wise approximation into training.

The MWVS design methodology introduced in Chapter 3 has the potential for generality to apply on several applications demanding low-power. Therefore, the application for MWVS on other platforms is worthy to be investigated. Also, MWVS is not necessary to rely on DSA to search the solution space, and there are many combinations of detailed implementation for achieving MWVS, which leaves many valuable directions for the next scope.

Bibliography

[1] Statista, *Artificial intelligence market size revenue comparisons*, 2020, https://www.statista.com/statistics/941835/artificial-intelligence-market-size-revenue-comparisons/.

[2] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," pp. 2295–2329, 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[6] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015. [Online]. Available: http://arxiv.org/abs/1512.02595

[7] B. Zhang, A. Davoodi, and Y. H. Hu, "Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 836–848, 2018.

[8] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[10] Y. Tao, R. Ma, M. Shyu, and S. Chen, "Challenges in energy-efficient deep neural network training with fpga," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 1602–1611.

[11] Statista, *Worldwide global datasphere real-time data annual size*, 2018, https://www.statista.com/statistics/949144/worldwide-global-datasphere-real-time-data-annual-size/.

[12] J. Redmon, "Darknet: Open source neural networks in c," 2013-2016. [Online]. Available: http://pjreddie.com/darknet.

[13] Wenlai Zhao, Haohuan Fu, W. Luk, Teng Yu, Shaojun Wang, Bo Feng, Yuchun Ma, and Guangwen Yang, "F-cnn: An fpga-based framework for training convolutional neural networks," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2016, pp. 107–114.

[14] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, "A framework for acceleration of cnn training on deeply-pipelined fpga clusters with work and weight load balancing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 394–3944.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[16] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[17] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, p. 37–42, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2831347.2831354

[18] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *Proc. VLDB Endow.*, vol. 11, no. 12, p. 2046–2049, Aug. 2018. [Online]. Available: https://doi.org/10.14778/3229863.3236256

[19] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, 2017, pp. 1–2.

[20] Y. Deng, "Deep learning on mobile devices - A review," *CoRR*, vol. abs/1904.09274, 2019. [Online]. Available: http://arxiv.org/abs/1904.09274

[21] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv e-prints*, vol. abs/1412.7024, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.7024

[23] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[24] Y. Wang, Z. Jiang, X. Chen, P. Xu, Y. Zhao, Y. Lin, and Z. Wang, "E2-train: Energy-efficient deep network training with data-, model-, and algorithm-level saving," *CoRR*, vol. abs/1910.13349, 2019. [Online]. Available: http://arxiv.org/abs/1910.13349

[25] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[26] Y. You, Z. Zhang, C. Hsieh, and J. Demmel, "100-epoch imagenet training with alexnet in 24 minutes," *CoRR*, vol. abs/1709.05011, 2017. [Online]. Available: http://arxiv.org/abs/1709.05011

[27] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *CoRR*, vol. abs/1802.04680, 2018. [Online]. Available: http://arxiv.org/abs/1802.04680

[28] X. Chen, X. Hu, H. Zhou, and N. Xu, "Fxpnet: Training a deep convolutional neural network in fixed-point representation," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2494–2501.

[29] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[30] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *CoRR*, vol. abs/1707.01083, 2017. [Online]. Available: http://arxiv.org/abs/1707.01083

[31] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[32] J. Kung, D. Kim, and S. Mukhopadhyay, "A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 85–90.

[33] P. Micikevicius, S. Narang, J. Alben, G. F. Diamos, E. Elsen, D. García, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *CoRR*, vol. abs/1710.03740, 2017. [Online]. Available: http://arxiv.org/abs/1710.03740

[34] D. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019. [Online]. Available: http://arxiv.org/abs/1905.12322

[35] N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," *CoRR*, vol. abs/1812.08011, 2018. [Online]. Available: http://arxiv.org/abs/1812.08011

[36] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5900–5904.

[37] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, 2016. [Online]. Available: http://arxiv.org/abs/1603.01025

[38] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1–6.

[39] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[40] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, Mar. 2016. [Online]. Available: https://doi.org/10.1145/2893356

[41] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[42] D. Kozen, "Semantics of probabilistic programs," *Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 328 – 350, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022000081900362

[43] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, 2010, pp. 25–34.

[44] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 124–134. [Online]. Available: https://doi.org/10.1145/2025113.2025133

[45] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 505–516.

[46] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *International Symposium on Code Generation and Optimization (CGO 2011)*, 2011, pp. 85–96.

[47] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11. New York,

NY, USA: Association for Computing Machinery, 2011, p. 164–174. [Online]. Available: https://doi.org/10.1145/1993498.1993518

[48] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting partially-forgetful memories for approximate computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, 2015.

[49] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *SIGARCH Comput. Archit. News*, vol. 39, no. 1, p. 213–224, Mar. 2011. [Online]. Available: https://doi.org/10.1145/1961295.1950391

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

[51] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[52] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 130–137.

[53] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC Design Automation Conference 2012*, 2012, pp. 820–825.

[54] M. Horowitz, "Energy table for 45nm process." Standford VLSI wiki.

[55] M. Imani, P. Wang, and T. Rosing, "Deep neural network acceleration framework under hardware uncertainty," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, 2018, pp. 389–394.

[56] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient mitchell's approximate log multipliers for convolutional neural networks," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 660–675, 2019.

[57] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 9, pp. 2856–2868, 2018.

[58] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18, New York, NY, USA, 2018.

[59] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "Quest: A 7.49tops multi-purpose log-quantized dnn inference engine stacked on 96mb 3d sram using inductive-coupling technology in 40nm cmos," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 216–218.

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

[61] R. DiCecco, L. Sun, and P. Chow, "Fpga-based training of convolutional neural networks with a reduced precision floating-point library," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 239–242.

[62] D. Shin, W. Choi, J. Park, and S. Ghosh, "Sensitivity-based error resilient techniques with heterogeneous multiply–accumulate unit for voltage scalable deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 520–531, 2019.

[63] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6.

[64] D. Kim, J. Kung, and S. Mukhopadhyay, "A power-aware digital multilayer perceptron accelerator with on-chip training based on approximate computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 164–178, 2017.

[65] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 27–32.

[66] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 701–706.

[67] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, p. 661, 2019.

[68] M. Imani, M. Masich, D. Peroni, P. Wang, and T. Rosing, "Canna: Neural network acceleration using configurable approximation on gpgpu," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 682–689.

[69] Y. Mishchenko, Y. Goren, M. Sun, C. Beauchene, S. Matsoukas, O. Rybakov, and S. N. P. Vitaladevuni, "Low-bit quantization and quantization-aware training for small-footprint keyword spotting," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 706–711.

[70] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 488–493.

[71] A. Tgarguifa, T. Bounahmidi, and S. Fellaou, "Optimal design of the distillation process using the artificial neural networks method," in *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, 2020, pp. 1–6.

[72] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 4, pp. 589–600, 1998.

[73] A. Dunkels, J. Eriksson, N. Finne, F. Österlind, N. Tsiftes, J. Abeillé, and M. Durvy, "Low-power ipv6 for the internet of things," in *2012 Ninth International Conference on Networked Sensing (INSS)*, 2012, pp. 1–6.

[74] M. Magno, L. Benini, C. Spagnol, and E. Popovici, "Wearable low power dry surface wireless sensor node for healthcare monitoring application," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013, pp. 189–195.

[75] P. Pawar, R. Nielsen, N. Prasad, S. Ohmori, and R. Prasad, "Hybrid mechanisms: Towards an efficient wireless sensor network medium access control," in *2011 The 14th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2011, pp. 1–5.

[76] J. Nagayama, Y. Masuda, M. Takeshige, Y. Ogawa, M. Hashimoto, and Y. Momiyama, "Activation-aware slack assignment (asa) for mode-wise power saving in high-end isp," in *ACM/ESDA/IEEE Design Automation Conference (DAC) Designer/IP track*, 2019.

[77] Kangmin Lee, Se-Joong Lee, and Hoi-Jun Yoo, "Low-power network-on-chip for high-performance soc design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 2, pp. 148–160, 2006.

[78] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning dvs processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, 2006.

[79] J. Park and J. Abraham, "A fast, accurate and simple critical path monitor for improving energy-delay product in dvs systems," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 391–396.

[80] Y. Kunitake, T. Sato, and H. Yasuura, "A replacement strategy for canary flip-flops," in *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, 2010, pp. 227–228.

[81] Y. Masuda, M. Hashimoto, and T. Onoye, "Critical path isolation for time-to-failure extension and lower voltage operation," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.

[82] Y. Masuda, T. Onoye, and M. Hashimoto, "Activation-aware slack assignment for time-to-failure extension and power saving," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2217–2229, 2018.

[83] S. Ghosh, S. Bhunia, and K. Roy, "Crista: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 1947–1956, 2007.

[84] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2010, pp. 825–831.

[85] A. B. Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano, "Axnoc: Low-power approximate network-on-chips using critical-path isolation," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1–8.

[86] Y. Masuda, T. Onoye, and M. Hashimoto, "Performance evaluation of software-based error detection mechanisms for supply noise induced timing errors," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100.A, no. 7, pp. 1452–1463, 2017.

[87] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-dnn: Energy efficient low-precision deep neural networks by compensating quantization errors," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[88] J. Johnson, "Rethinking floating point for deep learning," *CoRR*, vol. abs/1811.01721, 2018. [Online]. Available: http://arxiv.org/abs/1811.01721

[89] K. Roy and A. Raghunathan, "Approximate computing: An energy-efficient computing technique for error resilient applications," in *2015 IEEE Computer Society Annual Symposium on VLSI*, 2015, pp. 473–475.

[90] M. Gao and G. Qu, "Energy efficient runtime approximate computing on data flow graphs," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 444–449.

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

[92] 2009, http://www.nangate.com/index.php.

[93] C. Chang and C. Lin, "fourclass," 1996. [Online]. Available: https://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/binary.html.

[94] D. Anguita, A. Ghio, X. P. L. Oneto, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2013.

[95] D. Dheeru and G. Casey, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[96] P. M. Ciarelli and E. Oliveira, "Agglomeration and elimination of terms for dimensionality reduction," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, 2009, pp. 547–552.

[97] K. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[98] A. Tisan, S. Oniga, D. Mic, A. Buchman, and V. Babe, "Digital implementation of the sigmoid function for fpga circuits," in *ACTA TECHNICA NAPOCENSIS Electronics and Telecommunications*, 2009.

[99] J. Bush, "Nyuziprocessor: Source code," 2015. [Online]. Available: https://github.com/jbush001/NyuziProcessor.

[100] D. Flynn, *Low power methodology manual: For system-on-chip*, 1st ed. Upper Saddle River, NJ: Springer Science & Business Media, 2007.

[101] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Enhancing the efficiency of energy-constrained dvfs designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 10, pp. 1769–1782, 2013.

[102] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. USA: Cambridge University Press, 2007.

[103] L. Baudzus and P. M. Krummrich, "Modified downhill simplex method for fast adaption of optical filters in optical communication systems," *Electronics Letters*, vol. 55, no. 8, pp. 471–473, 2019.

[104] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3–14.

[105] K. Iwai, T. Kurokawa, and N. Nisikawa, "Aes encryption implementation on cuda gpu and its analysis," in *2010 First International Conference on Networking and Computing*, 2010, pp. 209–214.

[106] S. Madan, P. Kumar, S. Rawat, and T. Choudhury, "Analysis of weather prediction using machine learning big data," in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 2018, pp. 259–264.

[107] B. Wang and J. Tang, "The analysis of application of cloud computing in e-commerce," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, 2016, pp. 148–151.

[108] T. Zhang, Z. Lin, G. Yang, and C. D. Sa, "Qpytorch: A low-precision arithmetic simulation framework," *CoRR*, vol. abs/1910.04540, 2019. [Online]. Available: http://arxiv.org/abs/1910.04540

[109] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[110] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, Jun. 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[111] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[112] K. You, M. Long, M. I. Jordan, and J. Wang, "Learning stages: Phenomenon, root cause, mechanism hypothesis, and implications," *CoRR*, vol. abs/1908.01878, 2019. [Online]. Available: http://arxiv.org/abs/1908.01878