Ph.D. Dissertation of Engineering

Approximate Computing for Aging Compensation and Energy-efficient Neural Network

근사 컴퓨팅을 이용한 회로 노화 보상과 에너지 효율적인 신경망 구현

August 2020

Graduate School of Seoul National University

Department of Electrical and Computer Engineering

Heesu Kim

Approximate Computing for Aging Compensation and Energy-efficient Neural Network

지도교수 이 혁 재

이 논문을 공학박사 학위논문으로 제출함
2020년 7월

서울대학교 대학원
전기·정보공학부
김희수

김희수의 공학박사 학위논문을 인준함
2020년 7월

Abstract

Heesu Kim
Department of Electrical and Computer Engineering
College of Engineering
Seoul National University

Approximate computing reduces the cost (energy and/or latency) of computations by relaxing the correctness (i.e., precision) of computations up to the level, which is dependent on types of applications. Moreover, it can be realized in various hierarchies of computing system design from circuit level to application level.

This dissertation presents the methodologies applying approximate computing across such hierarchies; compensating aging-induced delay in logic circuit by dynamic computation approximation (Chapter 1), designing energy-efficient neural network by combining low-power and low-latency approximate neuron models (Chapter 2), and co-designing in-memory gradient descent module with neural processing unit so as to address a memory bottleneck incurred by memory I/O for high-precision data (Chapter 3).

The first chapter of this dissertation presents a novel design methodology to turn the timing violation caused by aging into computation approximation error without the reliability guardband or increasing the supply voltage. It can be realized by accurately monitoring the critical path delay at run-time. The proposal is evaluated at two levels: RTL component level and system level. The experimental results at the RTL component level show a significant improvement in terms of (normalized) mean squared error caused by the timing violation and, at the system level, show that the proposed approach successfully transforms the aging-induced timing violation errors into much less harmful computation approximation errors, therefore it recovers image quality up to perceptually acceptable levels. It reduces the dynamic and static power consumption by 21.45% and 10.78%, respectively, with 0.8% area overhead compared to the conventional approach.

The second chapter of this dissertation presents an energy-efficient neural network consisting of alternative neuron models; Stochastic-Computing (SC) and Spiking (SP) neuron models. SC has been adopted in various fields to improve the power efficiency of systems by performing arithmetic computations stochastically, which approximates binary computation in conventional computing systems. Moreover, a recent work showed that deep neural network (DNN) can be implemented in the manner of stochastic computing and it greatly reduces power consumption. However, Stochastic DNN (SC-DNN) suffers from problem of high latency as it processes only a bit per cycle. To address such problem, it is proposed to adopt Spiking DNN (SP-DNN) as an input interface for SC-DNN since SP effectively processes more bits per cycle than SC-DNN. Moreover, this chapter resolves the encoding mismatch problem, between two different neuron models, without hardware cost by compensating the encoding mismatch with synapse weight calibration. A resultant hybrid DNN (SPSC-DNN) consists of SP-DNN as bottom layers and SC-DNN as top layers. Exploiting the reduced latency from SP-DNN and low-power consumption from SC-DNN, the proposed SPSC-DNN achieves improved energy-efficiency with lower error-rate compared to SC-DNN and SP-DNN in same network configuration.

The third chapter of this dissertation proposes GradPim architecture, which accelerates the parameter updates by in-memory processing which is co-designed with 8-bit floating-point training in Neural Processing Unit (NPU) for deep neural networks. By keeping the high precision processing algorithms in memory, such as the parameter update incorporating high-precision weights in its computation, the GradPim architecture can achieve high computational efficiency using 8-bit floating point in NPU and also gain power efficiency by eliminating massive high-precision data transfers between NPU and off-chip memory. A simple extension of DDR4 SDRAM utilizing bank-group parallelism makes the operation designs in processing-in-memory (PIM) module efficient in terms of hardware cost and performance. The experimental results show that the proposed architecture can improve the performance of the parameter update phase in the training by up to 40% and greatly reduce the memory bandwidth requirement while posing only a minimal amount of overhead to the protocol and the DRAM area.

Keywords: approximate computing, circuit aging, stochastic computing, spiking neural network, in-memory, mixed-precision, neural processing unit

Student Number: 2015-20917

Contents

List of Figures

List of Tables

Chapter 1

Dynamic Computation Approximation for Aging Compensation

1.1 Introduction

1.1.1 Chip Reliability

Recently the chip reliability problem is getting much worse as the process technology scales down. Among others, BTI (Bias Temperature Instability) is a key reliability problem that degrades the chip performance by increasing the threshold voltage and decreasing the drain current [1]. This incurs chip slow-down, and after all, generates timing violation errors. In addition, metal-oxide thin-film transistors suffer fast aging and thus much larger timing variations in low supply voltage systems. Therefore, aging is a more serious problem in low power systems such as Internet-of-Things (IoT) or biomedical devices.

1.1.2 Reliability Guardband

The conventional approach to compensating for this aging-induced timing violation error is assigning a reliability guardband to supply voltage [2]. However, there are many problems in applying this approach to low power design. First of all, it is a too pessimistic approach since it should assign a relatively large guardband that can compensate for the chip slowdown after many years (say 10 years) of aging. This approach wastes extra power/area that is unnecessary during the first 10-years. Second, it is very difficult to determine an accurate guardband at design-time because the chip slowdown by aging depends on operating conditions including supply voltage, temperature, and application scenario. That is, the guardband should be determined by considering the worst-case operating conditions; it is very risky to determine the guardband by considering the statistical or balance conditions because it cannot ensure the normal chip operation in the worst-case conditions. Third, increasing the supply voltage to secure the guardband makes the aging accelerate.

1.1.3 Approximate Computing in Logic Circuits

Instead of increasing supply voltage and/or using faster (but larger) circuit as in the conventional approach while maintaining the computation accuracy, adopting the concept of approximate computing can be a more efficient solution in applications such as image/video processing, where the output quality is less sensitive to small errors. Under the assumption that the quality degradation by the approximation is not large, approximate computing can effectively increase performance and/or reduce power consumption. Therefore, simplified or approximate arithmetic (logic) circuits (adders, multipliers) are widely used to generate acceptable quality results in signal processing applications, especially in image/video processing applications [3, 4, 5, 6].

1.1.4 Computation approximation for Aging Compensation

In this chapter, an approach that enables the system to adapt to aging dynamically is presented. Thus, it does not need to add the pessimistic reliability guardband at design-time. Instead, it monitors the aging-induced delay at run-time and compensate for the increased delay by curtailing the critical path in a way of minimizing the accuracy loss due to the computation approximation. Therefore, it is essential to measure the chip slowdown due to aging and compensate for it adequately at run-time. For the implementation, the chip performance is measured periodically [7] using on-chip monitor since the aging is a very slow process. Ideally, this approach does not need to add any reliability guardbands. Practically, however, a small guardband should be secured to compensate the delay mismatch between the monitoring circuit and the actual critical path delay of the target block. The approach also requires reconfigurable circuits that can adjust the level of approximation. A design methodology is presented to implement reconfigurable adder and multiplier that replace conventional adder and multiplier. Since they are the fundamental arithmetic building blocks for more complex arithmetic circuits, arithmetic circuits consisting of them also can be implemented following such design methodology. Note that all aging-induced delays in the system cannot be compensated by computation approximation. For example, timing errors in control logic cannot be translated to computation error. For those kinds of logic, it should be handled by other means such as employing reliable margin, transistor upsizing, or other forms of redundancy.

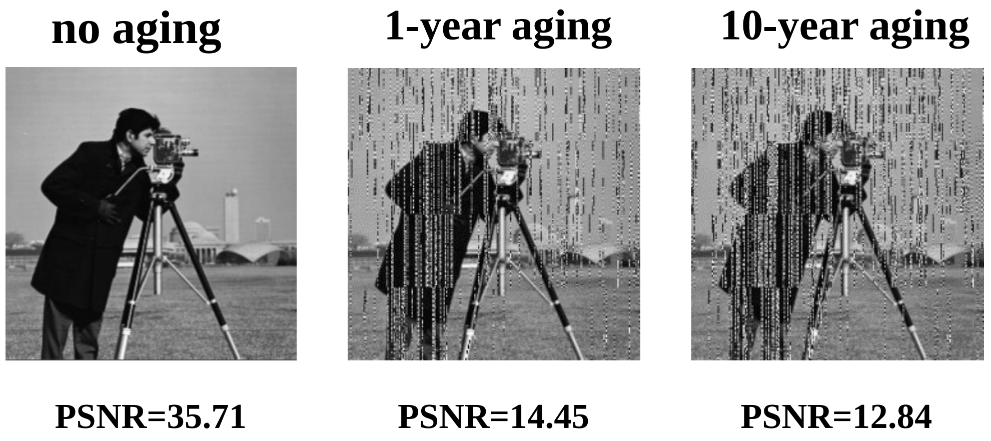


Figure 1.1: Impact of the aging-induced delay on an image processing application.

1.1.5 Motivational Case Study

The main issue in this case study is whether or not a reliability guardband is essentially required in every circuit design. Conventionally, even in error-tolerant circuits, the guardband is indispensably required to gain reasonable outputs. To demonstrate this, the proposed approach that compensates aging-induced delay dynamically is applied to a image processing system that performs Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT). Detailed setup is described in subsection 1.5.1.

To simulate aging effect of circuits, a degradation-aware library (standard cell libraries), which has been recently proposed and made publicly available [5] is used. The library is compatible with existing EDA tool flows like Synopsys and hence one can directly use them to perform static timing analysis of a circuit netlist without requiring any modifications. In practice, the library contains the delay information of standard cells under the effect (i.e., BTI) that aging has on the electrical characteristics of nMOS and pMOS transistors (e.g., threshold voltage and carrier mobility). Note that even though the degradation-aware library [5] only models BTI, the proposed approach is agnostic to any type of transistor aging (i.e., BTI or HCI) since it just measures aging-induced delay, then compensates the increased delay, regardless of its source. In practice, both of BTI and HCI can be modeled jointly during the cell library characterization [8]. In this work, I focus on the worst-case aging estimation provided by [5] in which a continuous DC stress is assumed (i.e., 100% duty cycle) in order to ensure reliability for the entire projected lifetime (10 years). Therefore, the recovery mechanism of BTI that depends on the running workload is not considered to provide upper boundary of aging.

As shown in Figure 1.1, removing the guardband in the circuit design results in a significant quality drop on an image when the circuit first encodes the image and then decodes it. In such a chain of circuits, errors are increasingly accumulated. The errors first occur in the encoder and then they are propagated to the decoder which leads to a larger impact on the quality of the final output image. As process technology advances, this problem becomes worse and worse because the aging-induced delay by BTI and HCI (Hot Carrier Injection) increases [9].

1.2 Previous Work

1.2.1 Aging-induced Delay

Many approaches have been studied to avoid aging-induced timing violation errors. The conventional approach to compensate for the errors is assigning an additional reliability guardband to supply voltage (or to slack of the critical path) [2]. However, it is too pessimistic and it is very difficult to determine an accurate guardband at design-time. In [5], the circuits are optimized against aging through logic synthesis with degradation-aware cell libraries. It enables the optimization process to select cells with most suitable input slew and output load capacitance for each set of operating conditions, considering aging effects. The work in [10] quantifies the impact of aging-induced errors and approximation errors on quality loss when the guardband is removed. It replaces the guardband with an equivalent reduction of precision in approximate computing applications. The aging effects are characterized and applied at design-time. However, such design-time approaches naturally result in an over-design. A more aggressive optimization can be done by run-time measurement of the chip slowdown due to aging and compensation. There is a technique for variation-resilient design that allow timing violation errors and manage the design reliability dynamically [11]. For measuring the chip slowdown, many on-chip aging monitors are presented [12, 13, 14, 15]. However, all these approaches raise the operation voltage to suppress the errors for accurate computing.

1.2.2 Delay-Configurable Circuits

Approximate computing [4] is a concept to trade-off computation accuracy for speed (i.e., delay) or energy, thus it can be exploited to alleviate timing violation errors including those incurred by the aging-induced delay [10]. Especially, approximate arithmetic circuits such as adders and multipliers have been widely studied. The approximate circuits can be divided into two categories: statically configurable and dynamically configurable circuits. The statically configurable circuits [16, 17, 18, 19, 20] have been proposed, and they have configuration parameters which enables the trade-off between the computation precision and the low energy consumption by adjusting approximation level. Some works [18, 19] extensively explore design space for finding an optimal point on the trade-off curve. [18] proposes an approximate adder having fine-grain configuration parameters, providing high flexibility in design space and the trade-off curve. [19] separates the higher significant bits and the lower significant bits, and processes them differently using some design alternatives, which they proposed to avoid errors in the higher significant bits, for better quality-energy trade-off. However, none of these approaches handle run-time effects such as the aging-induced delay since they need re-design or re-synthesis at design-time to change the configuration.

On the other hand, dynamically configurable circuits [21, 22, 23, 24, 25, 26] can be configured to various approximation levels at run-time without re-synthesis, thus being able to handle the run-time effects. The accuracy-configurable adder proposed in [21] changes the accuracy of results by selecting the operation mode at run-time. It basically outputs approximate results by cutting propagation path, and if needed, it corrects the errors for accurate results with multiple overlapping sub-adders. The gracefully-degrading adder in [22] is comprised of fixed structured multiple sub-adder units with selectable length for carry prediction bits, so it can satisfy a computation quality requirement varying at run-time. [25, 26] propose dynamically configurable multipliers that decrease power consumption by disabling switching activities of inactive sections resulting from the reduced input bit-width determined at run-time.

Although those circuits are configurable at run-time, they do not consider timing violation errors incurred by run-time effects, but primarily configure the approximate circuit to achieve energy reductions with acceptable accuracy degradation. Several prior work have specifically targeted timing ef-

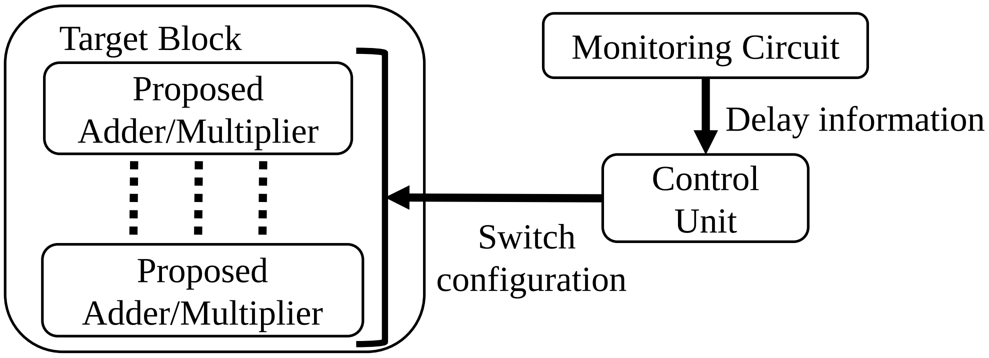


Figure 1.2: Simplified block diagram of the proposed system.

fects [6, 23, 24]. The dynamic voltage accuracy scaling approach proposed in [23, 24] achieves dynamic voltage overscaling by bit-width reduction under timing constraints. [6] proposes quality and delay configurable circuits to replace the temperature guardband at run-time. Neither of these approaches target aging, however.

This work is the first to directly measure and compensate for the aging-induced delay of basic arithmetic circuits such as adders and multipliers at run-time. The proposed approximate circuits truncate its least significant bits (LSBs) to reduce the critical path delay using the measured delay information at run-time with only a couple of gates overhead.

1.3 Proposed System

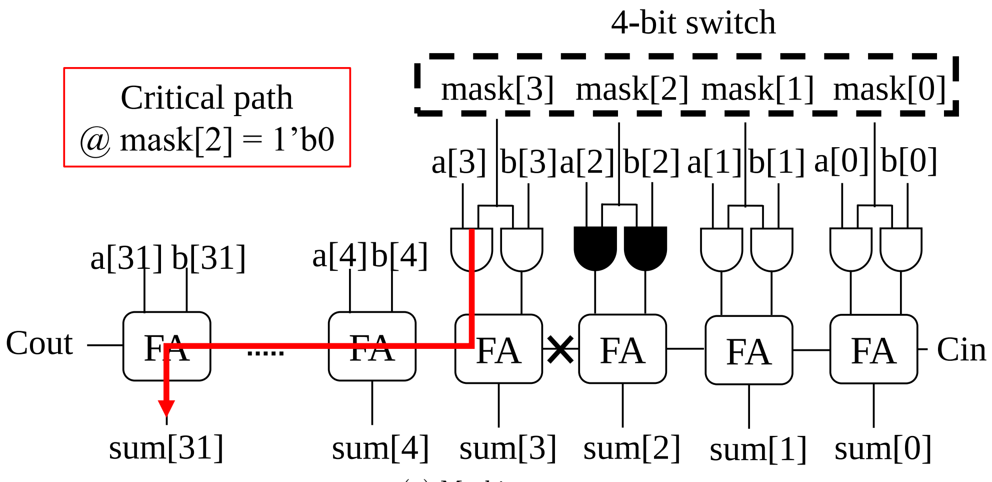
1.3.1 Overview of the Proposed System

Figure 1.2 is the simplified block diagram of the proposed system. It comprises the monitoring circuit, control unit, and the target block. The target block implements a signal processing application such as an image/video codec. it is considered to use the proposed adder/multiplier in the target block. The monitoring circuit outputs the delay information of the adders/multipliers in the target block under the current operating condition. Then, the control unit can reduce the critical path delay of the adders/multipliers by truncating LSBs according to the measured aging-induced delay information. It is performed using the switches of the adders/multipliers to cut-off their carry propagation paths. The overhead of the control unit is not too large because it just translates the 5-bit output of the monitoring circuit into the 4-bit configuration input in a way of counting. As mentioned in the previous section, the proposed adder/multipliers basically operates as an accurate circuit, but becomes an approximate one if some aging is detected by the monitoring circuit. Detailed explanations of each component in Figure 1.2 are given in the following subsections.

1.3.2 Proposed Adder

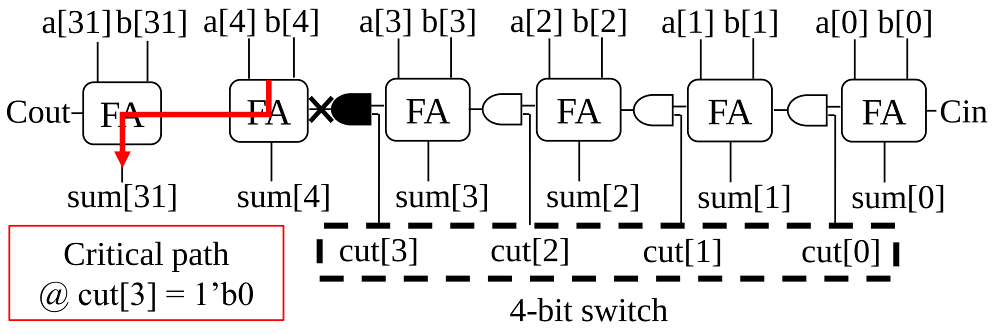
The proposed adder structure is based on ripple carry adder, which is the most cost/power-efficient adder among conventional adders. It shows the lowest power consumption and the best power-delay product metric, compared to other conventional accurate adders [16]. So, it has been widely chosen for the low power design. However, in a conventional ripple carry adder, errors in the most significant part (called MSP errors) are generated when the carry signal cannot be propagated to the MSP positions during one clock period due to the aging-induced delay. Such MSP errors are much more critical than errors in the least significant part (called LSP errors), especially when the sign bit in the 2's complement representation is involved in the errors.

To resolve this problem, two types of adders—masking and cutting— are proposed to prevent MSP errors. Figure 1.3 shows the structure of the two



4-bit switch
Critical path @ mask[2] = 1’b0
mask[3] mask[2] mask[1] mask[0]
a[3] b[3] a[2] b[2] a[1] b[1] a[0] b[0]
a[31] b[31]
a[4] b[4]
Cout
FA
.....
Cin
sum[31] sum[4] sum[3] sum[2] sum[1] sum[0]

(a) Masking type.



a[31] b[31] a[4] b[4] a[3] b[3] a[2] b[2] a[1] b[1] a[0] b[0]
Cout
FA
Cin
sum[31] sum[4] sum[3] sum[2] sum[1] sum[0]
Critical path @ cut[3] = 1’b0
cut[3] cut[2] cut[1] cut[0]
4-bit switch

(b) Cutting type.

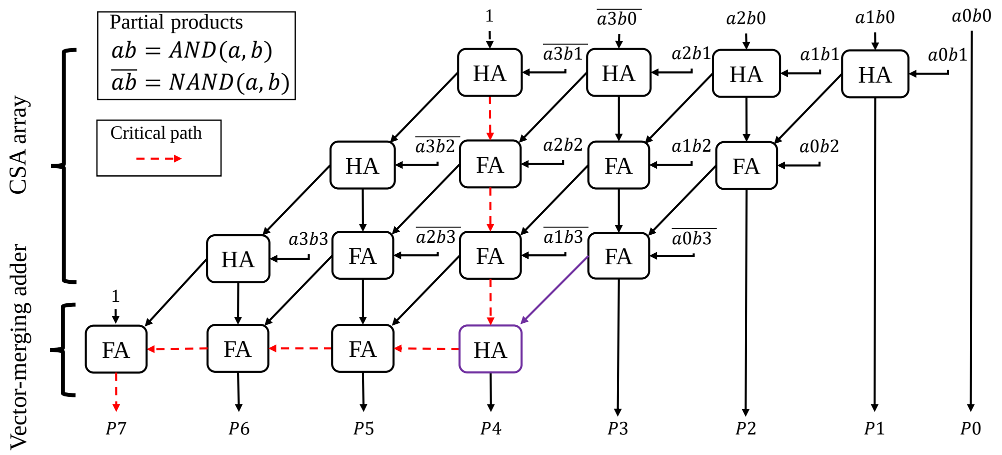
Figure 1.3: Proposed adders (a) masking type (b) cutting type .

proposed adders. The difference from the conventional adder is that the proposed adders have a 4-bit switch to cut-off the carry propagation path. These circuits reduce the critical path delay according to the configuration input value from the control unit. The masking type adder truncates some LSBs of the adder input to cut-off the carry propagation path, while the cutting type adder directly blocks the carry propagation from some LSBs. (those two types of gating are named to "*truncation*" afterwards.) For example, in case of masking type adder, when setting the configuration input to "mask[3:0] = 4'b1011", the carry-out (CO) of the third full adder (FA) is always zero. Then the critical path becomes shorter; the new critical path is from a[3] to sum[31] (red arrow in Figure 1.3a). Also, in case of cutting type adder, when setting the configuration input to "cut[3:0] = 4'b0111", the carry out of the fourth FA is always gated not to propagate it into the next (fifth) FA. They are configured dynamically at run-time, only when the aging-induced delay is detected by the monitoring circuit. The blocking of carry propagation in these examples may generate many LSP errors instead of a few critical MSP errors.

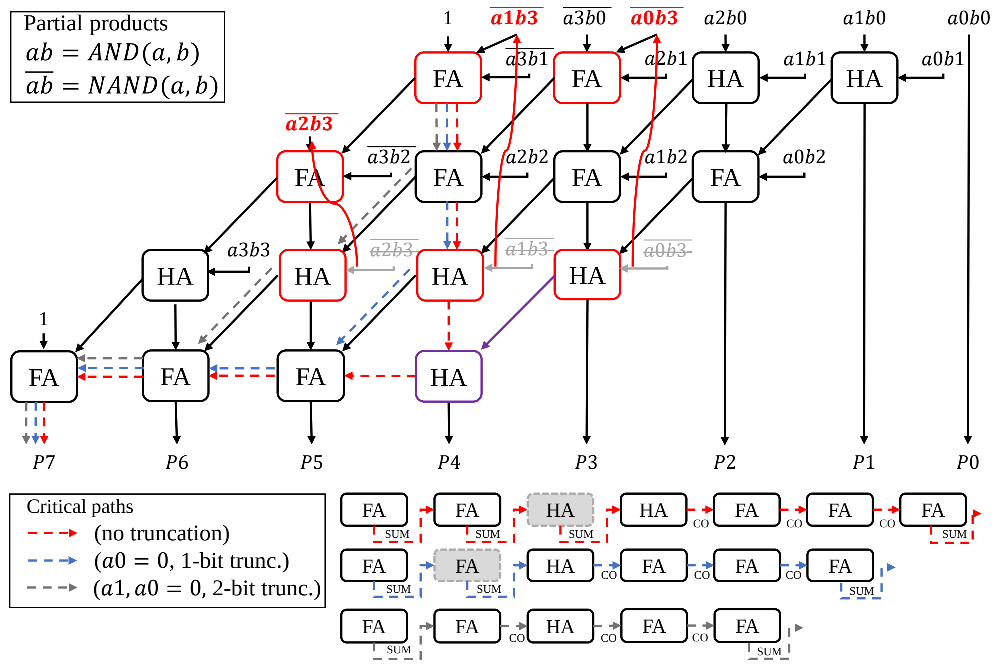
In Figure 1.3, we can see only the 4-bit switch to cut-off the carry propagation path. However, the optimal number of bits depends on the maximum amounts of the delay increase due to aging, which in turn depends on operating conditions and process technology. Detailed explanations determining it are given in Section 1.4.

1.3.3 Proposed Multiplier

The proposed multiplier structure is based on Baugh-Wooley multiplier [27] which supports unsigned and signed parallel multiplication and I call it Carry-Save-Adder (CSA) array multiplier (see Figure 1.4a) in this chapter because



(a) Conventional CSA array multiplier.



(b) Proposed CSA array multiplier.

Figure 1.4: 4-bit (a) conventional and (b) proposed CSA array multiplier. Dashed arrows present critical paths and those in (b) present them before and after the truncation of multiplicand operand. The red-colored operands are position-changed operands (partial products) in the proposed multiplier. The bottom of (b) shows detailed changes of the critical paths.

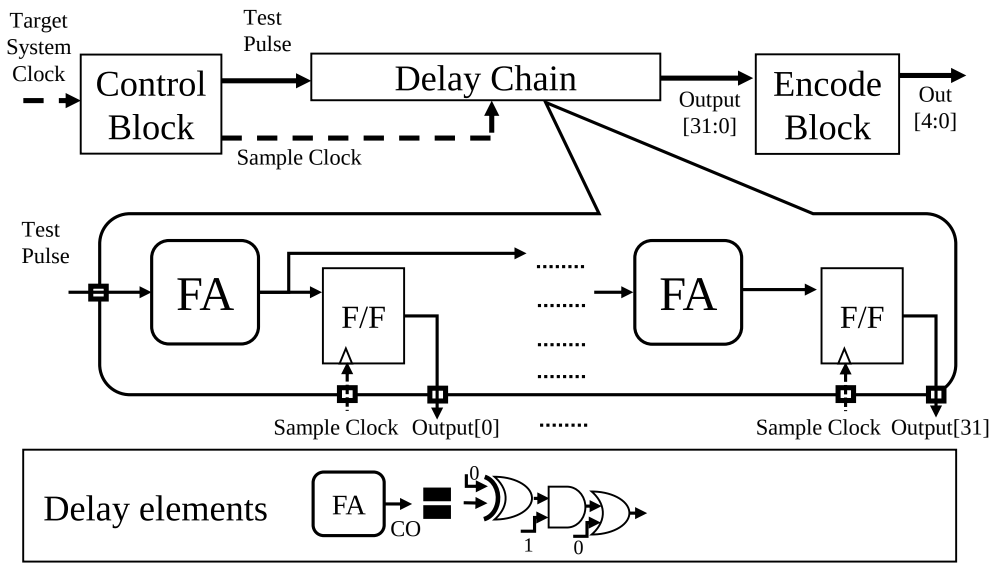
the main part of the multiplier is a CSA array, which has array structure with full adders (FAs) and half adders (HAs). The CSA array performs the partial product additions propagating values from the upper rows to bottom rows through diagonal arrows (carry-out) and vertical arrows (sum). The other part of the multiplier is the vector-merging adder, which performs the carry propagations to transform the outputs of the CSA array into a binary form. I consider this to be a representative multiplier, since the CSA array multiplier is a base structure of many parallel and power-delay optimized multipliers including [28] and [29], which employ [30] and/or Wallace-tree algorithm [31] along with the CSA array multiplier structure.

In order to cut the critical path by truncating the LSBs, two types of truncation, masking and cutting, can be considered as for the adder. However, masking is selected for the proposed multiplier, because there are many wires in between rows of CSA array making delay paths rather complex, thus requiring many AND cells to cut them if cutting type is chosen.

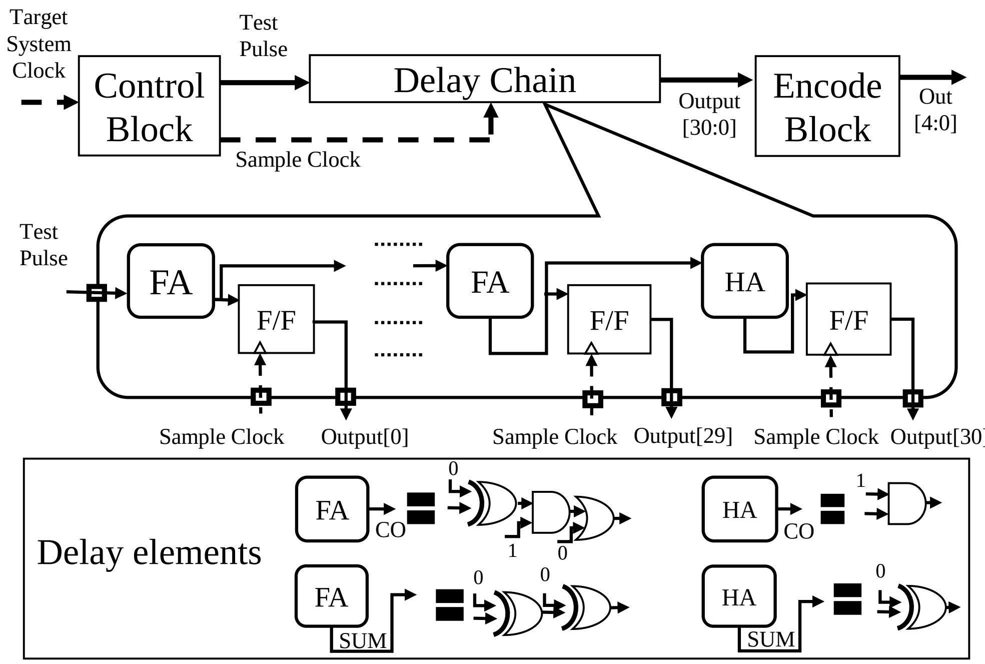
When applying the masking truncation, truncation happens in either multiplicand (ai's in Figure 1.4) or multiplier operand (bi's in Figure 1.4). If truncation occurrs in the LSBs of multiplier operand, it eliminates value propagation delay in the CSA array by forcing the outputs (carry-out and sum) of the corresponding FAs/HAs to be fixed since the truncation fixes the inputs of the FAs/HAs to either "0" or "1". For example, if 2 bits of LSBs of multiplier operand ($b0 = 0$, $b1 = 0$) is truncated as in Figure 1.4a, all outputs (carry-out and sum) of the FAs/HAs of the first row in the CSA array are fixed to either "0" or "1" depending on their inputs fixed for $b0$ or $b1$. Thus, the delay path through the first row is not on the critical path now. In the same way, as the number of truncated bits increases, the critical path delay decreases.

On the other hand, if the LSBs of a multiplicand operand is truncated , the partial products including sign bits ($a3$ or $b3$) in the conventional CSA array multiplier (Figure 1.4a) hinders the critical path reduction. For example, if $a0$ is truncated to be “0”, then $\overline{a0b3}$ is always “1”, and thus the purple arrow in Figure 1.4a can be either “0” or “1” according to the input of vertical arrow (the input of diagonal arrow is always “0” due to $a0 = 0$). Therefore, the critical path delay is not affected by the truncation. If fixing the value of the purple arrow to “0” happens, the critical path delay can be shortened since the carry-out of the purple HA is fixed to “0”. To achieve this, I change the position of operand-bits from a lower row to higher rows as represented by solid red-arrows and red operand-bits in Figure 1.4b, and exchange the FA and HA cells according to the position changes. Note that these changes do not alter the computation results. Now, if 1-bit in LSP is truncated ($a0 = 0$), the purple arrow is fixed to “0”. Hence the critical path is changed to the blue-dashed arrows from the red-dashed arrows as depicted in Figure 1.4b. For 2-bit truncation ($a0 = 0, a1 = 0$), the critical path becomes gray-dashed arrows. Note also that this position changes of operands do not cost any overhead, since the number of FAs and HAs is unchanged.

I choose to truncate multiplicand operand, because it can reduce the critical path delay starting from 1-bit truncation. On the other hand, if I choose to truncate multiplier operand, the minimum number of LSBs that can reduce the critical path delay is 2 bits since 1-bit truncation cannot fix the outputs of every FA/HA on the first row. Therefore, truncating the multiplicand operand shows better efficiency in terms of the trade-off between reduction of critical path delay and errors of computation approximation. It also enables to handle the initial stage of aging with 1-bit truncation.



(a) Generic monitoring circuit corresponding to 32-bit ripple carry adder.



(b) Dedicated monitoring circuit corresponding to 16-bit CSA array multiplier.

Figure 1.5: Structure of the proposed monitoring circuits; (a) generic and (b) dedicated. Delay elements used in each monitoring circuit are presented at bottom of each. Note that cells for generating partial products and switch logic are omitted for clarity.

1.3.4 Proposed Monitoring Circuit

The monitoring circuit is a key component to determine the quality and effectiveness of the proposed system. More accurate aging-induced delay measurement makes the quality degrade more gracefully with approximation. On the other hand, the inaccurate measurement might bring excessive quality degradation from LSP errors or still incur MSP errors, in spite of using the run-time monitoring system. The dynamic approach gives better quality results than static approach under the assumption that the sensor/monitor gives accurate delay information.

Monitoring circuits is designed to be used for the proposed adder/multiplier based on the state-of-the-art monitoring circuit [7]. Figure 1.5 shows the detailed structure of the monitoring circuits designed for the 32-bit adder and the 16-bit multiplier. Both of them consist of three main blocks: delay chain, control block, and encode block. Delay chain is an array of delay elements, each of which represents an element of the critical path of the target circuits.

The monitoring circuit proposed in [7] is a *generic monitoring circuit* which uses unit delay elements to measure increased delay. It computes the ratio of the number of delay elements which are not reachable in a clock cycle over the total number of the delay elements. Therefore, it is applicable to any target circuits including the ones proposed here. However, the resolution of delay adjustment can be a problem. In [7], while they control the continuous value (i.e. supply voltage) to adjust the circuit delay, it handles a discrete value, which is the number of truncation bits in target circuits. Therefore, the delay elements should be designed to be aligned with the unit of truncation. This type of monitoring circuit is called as *dedicated monitoring circuit*. The other parts of the dedicated monitoring circuit except for the comprising delay elements are equal to the generic monitoring circuit.

The monitoring circuits operate as follows. First, the control block generates a test pulse and sample clock signals by using the target system clock. While the test pulse signal propagates through the delay chain, the sample clock samples it to see how many delay elements are propagated through within one clock period. Then the outputs of the flip-flops (a string of 1's followed by a string of 0's) are encoded to 5-bit delay output information. For example (Figure 1.5a), at the initial year (0-year without the guardband), the output [31:0] is always 32'hffff_ffff. However, if the input pulse signal cannot propagate through the whole delay chain within one clock period due to the aging-induced delay, it becomes 32'h7fff_ffff, 32'h3fff_ffff, 32'h1fff_ffff or 32'h0fff_ffff as the amount of delay increases. Based on this output information from the monitoring circuit, the control unit can truncate the LSBs of the target circuit according to the amount of propagation path beyond a clock cycle. Note that some input bits of the delay elements (FAs/HAs) are fixed to either "0" or "1" in order to make them bypass the test pulse.

In the case of the proposed adder, it consists of "FA-CO (FA carry-out)" propagation path only in their critical path, so if FA-CO is chosen as a unit delay element for the monitoring circuit, the dedicated monitoring circuit has a structure of the generic monitoring circuit. Practically, two-input NAND, NOR, XOR cells comprises a delay element and a flip-flop (F/F) is appended to each of the delay element as depicted in Figure 1.5a. As a result, the whole delay chain has completely the same cell composition as the critical path of the proposed adder (i.e., ripple carry adder). That is why the monitoring circuit can measure the aging-induced delay of the adder accurately. The number of delay elements depends on the number of adder bits. In case of the proposed multiplier, as shown in Figure 1.4, the critical path of CSA array multipliers contains four types of propagation paths that exist in FAs and HAs. They are “FA-SUM” path consisting of two XOR cells, “FA-CO” path consisting of an XOR, an AND, and an OR cells, “HA-SUM” path consisting of one XOR cell, and “HA-CO” path consisting of one AND cell as depicted in Figure 1.5b. Therefore, four types of delay elements corresponding to each propagation path are designed, then they constitute a delay chain by connecting themselves in the reverse order of LSBs truncation. For example, as shown in the bottom right of Figure 1.4b, after the 1-bit truncation, the critical path is shortened as the amount of “HA-SUM” delay (upper shaded delay path) and after the 2-bit truncation, the critical path is further shortened as the amount of “FA-SUM” (bottom shaded delay path). Therefore, “HA-SUM” delay element is placed in the end of the delay chain and a “FA-SUM” delay element in front of the “HA-SUM” delay element as shown in Figure 1.5b. Unlike the adder where the generic and the dedicated monitors have the same structure, the generic and the dedicated monitor of the multiplier have different structures, their performance will be compared in subsection 1.5.3. The number of delay elements on the delay chain depends on multipliers number of operand-bits.

Note that in the delay chains, the delay elements for generating partial products (an AND or a NAND) and switch logic (an AND) are omitted. However, they can be considered by placing equivalent delay elements in the delay chain or using a small guardband.

The monitoring circuit is required to have switching activity similar to that of the target circuits for having closely correlated aging characteristics. However, checking whether aging-induced delay incurs timing violation, involving fetching output values of delay elements into flip-flops, is not required to be

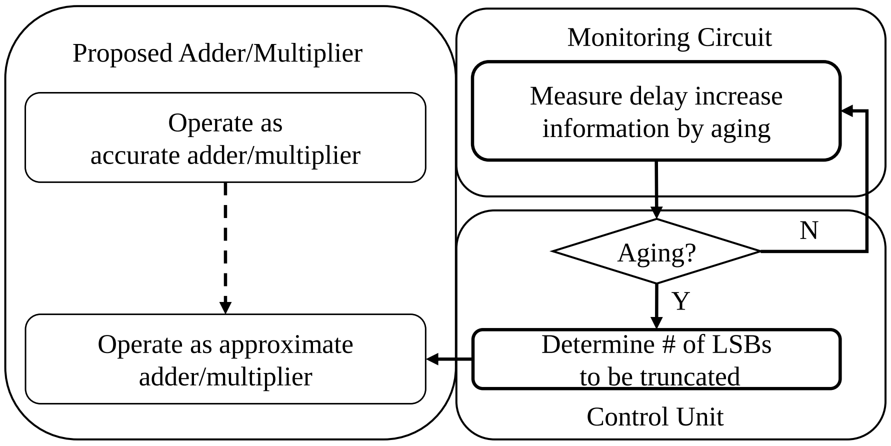

Proposed Adder/Multiplier
Operate as accurate adder/multiplier
Operate as approximate adder/multiplier
Monitoring Circuit
Measure delay increase information by aging
Aging?
N
Y
Determine # of LSBs to be truncated
Control Unit

Figure 1.6: Aging compensation scheme with approximation.

performed at all times, because aging is a very slow process with alternating phases of stress and recovery. Also, the delay measured by the monitoring circuit should be calibrated to get pure aging-induced delay independent to the process, voltage, and temperature variations that are also measured by own monitors [32].

1.3.5 Aging Compensation Scheme

Figure 1.6 illustrates the proposed aging compensation scheme using computation approximation. At 0-year (no aging), the proposed adder/multiplier of target block operate as accurate adder/multiplier. At the same time, the monitoring circuit periodically gives the delay information to the control unit. If an aging-induced timing violation error is detected, the control unit determines the number of LSBs to be truncated by the amount of delay increase due to aging. Then the proposed adder/multiplier operate as approximate ones by truncating some LSBs. When the delay by aging further increases, the control unit conpart1/figures the adder/multiplier to truncate more LSBs. This scheme is automatically operated at run-time.

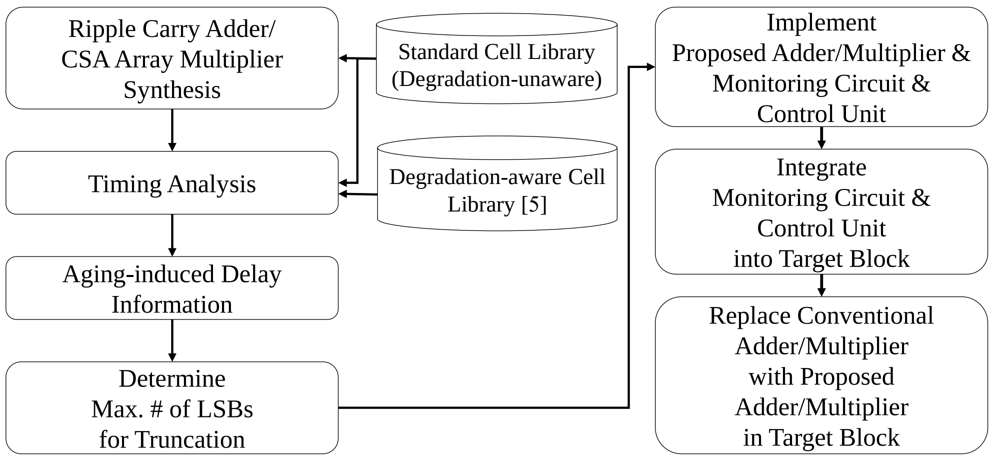


Figure 1.7: Design methodology of the proposed system.

1.4 Design Methodology

In order to apply the proposed scheme, it is required to implement the proposed adder/multiplier and monitoring circuit for a given design specification and process technology. The design methodology of the proposed system is shown in Figure 1.7. The flow chart on the left-hand side shows the *design analysis steps* for the proposed system implementation, and the flow chart on the right-hand side shows the *design integration steps* to integrate the proposed circuits into the target block design. It is very easy to plug the proposed methodology into a conventional design flow.

In the design analysis steps, a ripple carry adder and a CSA array multiplier are synthesized as the reference adder/multiplier structure with general standard (degradation-unaware) cell library, which does not consider any aging effects. With static timing analysis, the aging-induced delay is measured by comparing the critical path delay before and after the aging. The critical path delay after aging is analyzed with degradation-aware library [5] as mentioned in subsection 1.1.5. Note that even though the design methodology requires to develop a degradation-aware library for each technology, it is a one-time effort for each technology and not application-specific. Therefore, it can be reused for any design in the same technology. In addition, during the cell library creation several corners and several operating conditions are typically considered. Therefore, having an extra corner that describes the library behavior after aging would not lead to a significant increase in complexity.

Based on the measured aging-induced delay, the maximum number of LSBs to be truncated for aging compensation can be determined during the expected lifetime. Just one simulation run is enough to determine the maximum number of LSBs (*MaxBits*) to be truncated during the expected lifetime (10 years) of the target circuit since if a few more bits are added on the simulated MaxBits, it can sufficiently cover the whole range of the truncation. That is because MaxBits is the upper bound of the coarse-grain (bit-wise) truncation and a few more bits (2-3 bits) on the switch logic cost only a few more gates, such as ANDs and NANDs. It is noteworthy that the reference circuits do not have a switch logic, and thus the reference designs have shorter critical path delay than the proposed circuits. However, it also can be sufficiently covered by MaxBits since the switch logic has only a gate cell delay.

Next, in the design integration steps, the monitoring circuit, control unit, and the proposed adder/multiplier are implemented according to the specification determined in the previous steps. Then, the monitoring circuit and the control unit are integrated into the target block. Finally, the conventional adder/multiplier is replaced with the proposed adder/multiplier in the target block.

There are some important guidelines to correlate the delay characteristics between the delay chain of the monitoring circuit and the critical path of the proposed adder/multiplier. As the cell delay is mainly affected by process, voltage, and temperature (PVT), it is very important to make the PVT conditions of the monitoring circuit same as that of the adder/multiplier in the target block. This PVT matching directly relates to the accuracy improvement of the proposed system. The guidelines are; First, the monitoring circuit should share the same supply voltage rail with the adder/multiplier in the target block to correlate them closely in terms of delay characteristics and aging process, both of which are voltage dependent. Second, the monitoring circuit should be placed closely to the target block because the delay variation due to on-chip process variation is too large to be ignored in an advanced process technology and the temperature inside one chip can vary over a range of tens of degrees at the worst-case when either of monitoring circuit or target circuit is located at the hotspot region whereas the other is located at the cool (idle) region. These guidelines are mandatory for better delay correlations between the monitoring circuit and the adder/multiplier in the target block, and thus they should be followed carefully when integrating the monitoring circuit into a chip.

In practice, even when following the guidelines, any remaining PVT mismatches between the monitoring circuit and the adder/multiplier in the target block need to be compensated by employing a small guardband. The PVT mismatches can in general result in mismatches in the calibration for the pure aging-induced delay mentioned in subsection 1.3.4 as well as aging degree of the monitor and target circuits. For the case of temperature, which has the largest effect on delays, assuming a worst-case temperature difference of 2 °C between the monitor and target circuits results in a maximum 1.8% difference in delays [6] and a negligible difference in aging degree over 10 years [33] that have to be covered by a guardband. In addition, the switching activity differences, when the monitor circuit is shared across the target circuits, also have to be covered by a guardband.

There are several approaches to estimate and bound such margins and guardbands. To estimate PVT mismatches and guardbands, the result of standard intra-chip PVT variation analysis, which is supported by Process Design Kits (PDKs) with analysis tools and statistical information for varying PVT conditions of the underlying technology, can be utilized. The PDKs and statistical information are provided by foundry companies. Given PVT mismatches, degradation-aware cell libraries characterized for different PVT operating points can be used to analyze both the direct delay impact and the indirect impact on aging and aging-induced delays under different PVT conditions of monitor and target circuit. The designer can apply the estimated guardband by adding the worst-case difference in six or three sigma values obtained from the analysis and/or the statistical data. Note that one does not need to analyze all pairs of target and monitoring circuits in a chip. Estimating mismatches and guardbands can be done conservatively to determine the worst-case scenario over a chip (if necessary, a domain). Finally, to further contain mismatches and required guardbands, one can implement the sensor (i.e., monitor) using standard cells that are the most susceptible to aging. In such a case, the reading and predictions from the sensor will be conservative.

From the hardware overhead point of view, this aging compensation system does not generate a large area overhead. Compared to the conventional ripple carry adder and CSA array multiplier, the overhead of the proposed adder/multiplier can be ignored because a couple of AND and NAND cells are added to cut-off the carry propagation path. The monitoring circuit and the control unit are very simple logics. In addition, this system does not require voltage scaling when the delay increases, because it compensates for the aging-induced delay increase by computation approximation. So, it does not incur additional power consumption due to the voltage scaling for the assignment of the reliability guardband.

For the logic that cannot be compensated by computation approximation (e.g., control logic), the designer can use the degradation-aware cell library to synthesize them for aging compensation [5]. This ensure that the synthesized netlist consistently does not violate timing constraints even at the maximum age that the cell library is characterized for. Since the cell library provides worst-case delay information incurred by aging effects, the synthesis tool will generate a circuit that will meet constraints under such maximum-age delays. Besides, conventional types of aging compensation schemes also can be also used (e.g., up-sizing the transistors and increasing supply voltage).

1.5 Evaluation

1.5.1 Experimental setup

Two experiments are conducted to evaluate the proposed system and scheme. First, two types (masking and cutting) of 32-bit proposed adder and one type (masking) of 16-bit multiplier are implemented and they are evaluated with randomly generated inputs. The 100K random inputs are generated with normal distributions using $dist_normal() functions in Verilog since the random inputs with normal distribution have similar characteristics with the actual input extracted from an image encoder/decoder block [5]. Mean and standard deviation values in [34] are used for this test input generation. Next, the proposed system and scheme with real application environment is evaluated. The experiment with DCT and IDCT circuits, which encode and decode input images that are widely used for image codec applications, is performed. The output port of DCT is directly connected to the input port of IDCT to encode and then decode the input image file and see the impact of aging-induced delay on the image quality. The two different monitoring circuits for adders and multipliers are used to measure the aging-induced delay. Note that the monitoring circuits can be shared by operators (adder/multiplier) of the same kind in the same block while the granularity of blocks varies from design to design.

Synopsys Design Compiler (N-2017.09-SP4) is employed with *compile_ultra* option to synthesize the RTL codes of the proposed adder/multiplier with the degradation-aware cell libraries, based on the 45nm Nangate process technology [5]. To prevent the re-arrangement of the gate-level structure by Design Compiler, the bottom-up synthesis methodology (i.e., *Hierarchical design*) is used: Firstly, adder and multiplier are synthesized with the same clock period longer than the point that incurs the structure re-arrangement. Then, a target block (i.e., MAC unit) synthesized by integrating the synthesized adder and multiplier with the target clock period ($2.65ns$), while applying *set_dont_touch* option on the adder and multiplier not to re-arrange the synthesized netlist.

Synopsys Prime Time (J-2014.06-SP3) is employed for the static timing analysis and power estimation of the synthesized netlist under aging. the timing information is analyzed while changing the configuration of the 4-bit switch with the “set_case_analysis” command. Gate-level simulation is performed with Mentor ModelSim, in order to analyze the MSE, NMSE and error rate

Table 1.1: Comparison of 32-bit Ripple Carry Adder and Proposed Adder

32-bit Ripple Carry Adder							
Aging Time	**year 0**	**year 1**			**year 10**		
Dynamic + Static Power (uW)	53.86	53.70			54.43		
Area	153.22						
Critical Path Delay (ns)	1.966	2.065			2.129		
Error Rate	0.00%	1.54%			2.89%		
MSE	0.00	4.85E+16			8.18E+16		
NMSE	0.00	1.85E+07			2.11E+07		
32-bit Proposed Adder - Masking							
Aging Time	**year 0**	**year 1**			**year 10**		
# of Truncated LSBs	**-**	**0**	**1**	**2**	**0**	**2**	**4**
Dynamic + Static Power (uW)	56.21	56.03	55.16	53.78	56.62	54.39	51.47
Area	161.73						
Critical Path Delay (ns)	1.993	2.108	2.044	1.980	2.172	2.040	1.865
Error Rate	0.00	1.54%	75.18%	75.13%	2.89%	75.53%	74.87%
MSE	0.00	4.85E+16	3.85E+16	5.98	8.18E+16	7.68E+16	95.67
NMSE	0.00	1.85E+07	1.39E+07	0.21	2.11E+07	2.17E+07	0.84
32-bit Proposed Adder - Cutting							
Aging Time	**year 0**	**year 1**			**year 10**		
# of Truncated LSBs	**-**	**0**	**1**	**2**	**0**	**2**	**4**
Dynamic + Static Power (uW)	54.71	54.54	53.13	52.14	55.27	52.86	51.01
Area	157.47						
Critical Path Delay (ns)	2.032	2.134	2.053	1.972	2.198	2.033	1.865
Error Rate	0.00%	1.53%	50.55%	49.93%	3.02%	51.18%	49.87%
MSE	0.00	4.71E+16	5.20E+16	7.99	8.31E+16	1.14E+17	127.67
NMSE	0.00	1.79E+07	1.99E+07	0.21	2.09E+07	3.51E+07	0.89

by aging-induced delay. Standard delay file (.sdf) is used to consider the aging-induced delay for the gate-level simulation.

the 256×256 representative images from “video trace library" [35] are used as the input images. The Peak Signal-to-Noise Ratio (PSNR) metric is used for the evaluation of image quality. In the following subsection, image quality changes at year 0 and year 10 with/without the proposed compensation scheme will be presented. These results are generated by gate-level simulation with the degradation-aware cell libraries [5].

1.5.2 RTL component level – Adder/Multiplier

The two types (masking and cutting) of the proposed 32-bit adders and one type (masking) of the proposed 16-bit multiplier, which can be configured to truncate four bits in LSBs at maximum, are implemented. The maximum number of truncated bits in LSBs is so decided because the delay increase by 10-year aging (about 8%) can be sufficiently compensated by 4-bit truncation for the adder and 3-bit truncation for the multiplier. In terms of the accuracy at the component level, the aforementioned error metrics (MSE, NMSE and error rate) are used.

Table 1.1 shows the comparison of power, area, and critical path delay between the conventional 32-bit ripple carry adder and the two types of the proposed adders. The conventional ripple carry adder generates aging-induced timing violation errors when the reliability guardband is not included. The error rate increases up to 1.54% (year 1) and 2.89% (year 10) and MSP errors incur high MSE and NMSE, which is due to the critical path delay increase by about 5.03% (year 1) and 8.29% (year 10). More than half of the aging-induced delay occurs within the first year, which is about 60% of the total delay increase by aging during the expected lifetime.

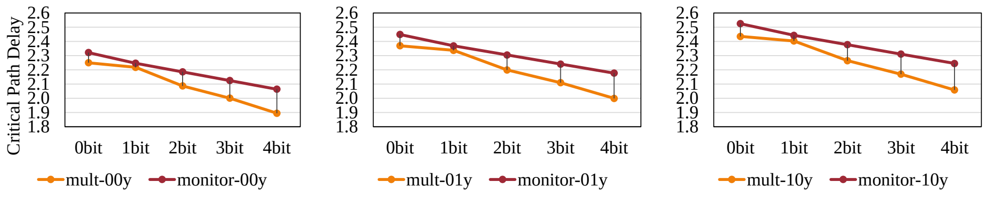
The critical path delay decreases by configuring the switch to cut-off the carry propagation path. At year 1, the critical path delay of 2-bit truncation is smaller than that of no truncation at year 0. It means that the 1-year's aging-induced delay can be compensated by the 2-bit truncation, and it eliminates the MSP errors. On the other hand, 1-bit truncation at year 1 cannot eliminate MSP errors perfectly, so it still shows high error values (i.e., MSE and NMSE). At 10-year, however, it is required to truncate four bits, because 2-bit truncation is not enough to recover the critical path delay. When the delay

Table 1.2: Comparison of 16-bit Carry Save Adder Array Multiplier and Proposed Multiplier

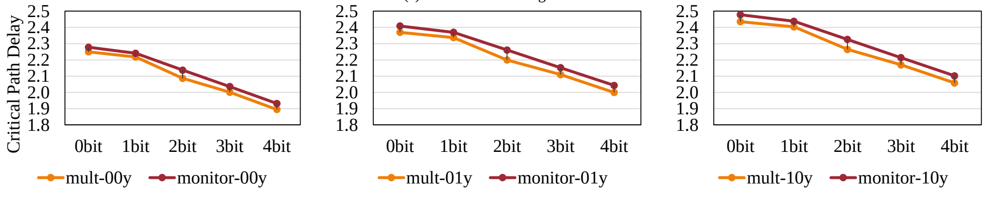
16-bit CSA Array Multiplier							
Aging Time	**year 0**	**year 1**			**year 10**		
Dynamic + Static Power (uW)	661.90	659.40			657		
Area	1409.53						
Critical Path Delay (ns)	2.223	2.342			2.407		
Error Rate	0.00%	4.35%			8.97%		
MSE	0.00	1.35E+17			2.12E+17		
NMSE	0.00	7.63E+15			8.10E+15		
16-bit Proposed Multiplier - Masking							
Aging Time	**year 0**	**year 1**			**year 10**		
# of Truncated LSBs		**0**	**1**	**2**	**0**	**2**	**3**
Dynamic + Static Power (uW)	665.30	662.80	645.20	621.90	660.40	619.90	596.20
Area	1414.32						
Critical Path Delay (ns)	2.250	2.370	2.338	2.200	2.435	2.264	2.169
Error Rate	0.00%	5.72%	47.45%	70.00%	10.55%	70.14%	81.68%
MSE	0.00	1.86E+17	5.84E+16	126.41	2.44E+17	2.19E+16	632.06
NMSE	0.00	7.46E+15	1.77E+15	0.83	1.10E+16	1.79E+14	4.12

is compensated, the error values decrease significantly even though the error rate (rate of inaccurate sum) increases. Note that this error rate mostly comes from the LSP errors, so the error values are not large. This experimental results demonstrate that the proposed adders compensate for the aging-induced delay with computation approximation gracefully.

Table 1.2 shows that the same analysis can be done on the proposed multiplier. Without any truncation, after 1-year and 10-year aging, it shows very high error values due to the timing violation. The increased delay ratios are 5.35% (year 1) and 8.28% (year 10) which are close to that of the adder. However, the multiplier has a higher error rate than the adder; 1.53% vs 5.72% (year 1) and 3.02% vs 10.55% (year 10). It means that the multiplier produces more timing violations than the adder in both of year 1 and year 10. By truncating LSBs, the same benefits as in the adder can be achieved. After 1-year aging, the multiplier effectively recovers their functionality by truncating two bits with acceptable approximation error, 0.83 in NMSE. That is because the



(a) Generic monitoring circuit



(b) Dedicated monitoring circuit

Figure 1.8: Critical path delays of the proposed multiplier and two types of monitoring circuits: (a) generic and (b) dedicated. The shorter distance between the critical path delays means the lower delay mismatch and the better monitoring performance.

critical path delay becomes shorter than the delay of no truncation at year 0. After 10-year aging, however, the multiplier needs one more bit (i.e., three bits in total) to be truncated for acceptable error compensation.

Static power is always reduced due to aging since the aging increases the threshold voltage of transistors. However, dynamic power depends on the circuit design and application, and thus it can increase or decrease by aging [36]. In the experiment, the total power of the adders slightly decreases at year 1 but increases at year 10. However, for the multipliers, total power consistently decreases as aging proceeds. In case of the proposed adder/multiplier, a couple of additional AND or NAND cells are inserted for the switch logic. Due to these small overheads, the delay, area, and power consumption increase a little bit compared to the reference circuit. However, as more LSBs get truncated, the power consumption gradually decreases since the cut-off paths stop switching after the truncation.

1.5.3 RTL component level – Monitoring circuit

the two types of monitoring circuit are introduced in subsection 1.3.4. One is the generic monitoring circuit which contains homogeneous delay elements. The other is the dedicated monitoring circuit which contains target-dependent heterogeneous delay elements. Because the adder contains homogeneous delay components (FA-CO) on its critical path, its delay can be accurately measured by a generic monitoring circuit. On the other hand, because the multiplier contains heterogeneous delay components (FA-CO, FA-SUM, HA-CO, and HA-SUM) on its critical path, it is inefficient to use a generic monitoring circuit to measure the its delay. To demonstrate this, critical path delay of the proposed multiplier is measured with the two types of monitoring circuit.

The requirement of feasible monitoring circuit is that its delay should be longer than that of the target circuit at every aging stages and for any number of truncated bits. That requires the monitoring circuit to have some amount of margin. However, if monitoring circuits have an excessive margin than is necessary, it can cause unnecessary truncation in the target circuit. To make matters worse, the excessive margin may impose slow circuit. Therefore, tight margin is essential for better monitoring circuits.

Figure 1.8 shows the critical path delays of the multiplier and the two types of monitoring circuits while increasing the number of truncated bits at year 0, year 1, and year 10 of aging using the degradation aware library [5]. Figure 1.8a is for the generic monitoring circuit, and Figure 1.8b is for the dedicated monitoring circuit. If comparing the two graphs, we can notice that the differences of the critical path delays between the multiplier and the monitoring circuit are bigger for the case of the generic monitoring circuit in Figure 1.8a.

The generic monitoring circuit has a line with a constant negative slope

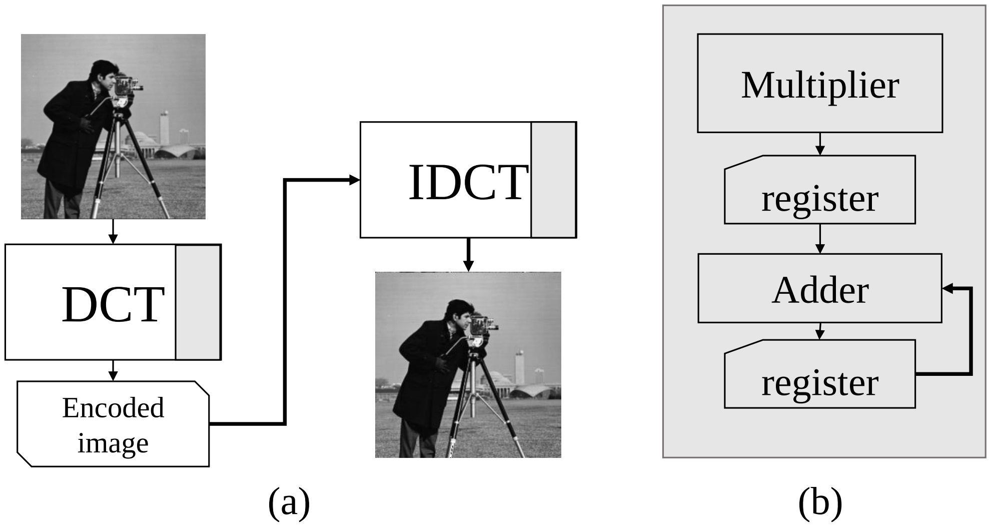


Figure 1.9: (a) DCT/IDCT codec blocks and (b) matrix multiplication unit in DCT and IDCT blocks.

as it contains homogeneous delay elements. To meet the monitoring circuit requirement described above, in Figure 1.8a, the line of monitoring circuit should be placed above the line of the multiplier when 1-bit is truncated. As a result, it imposes excessive margin for the other number of truncated bits.

On the other hand, in case of the dedicated monitoring circuit, as shown in Figure 1.8b, the delay curve of the dedicated monitoring circuit follows that of the multiplier very closely on every number of truncated bits and aging years while keeping the requirement described above. As a result, it requires less margin than that of the generic monitoring circuit.

1.5.4 System level

To evaluate the proposed system and scheme at the system level, DCT/IDCT codec blocks (See Figure 1.9) which encode and then decode the images from “video trace library" [35] are used. With this experiment, the feasibility of the proposed system in a real image processing application are shown. In

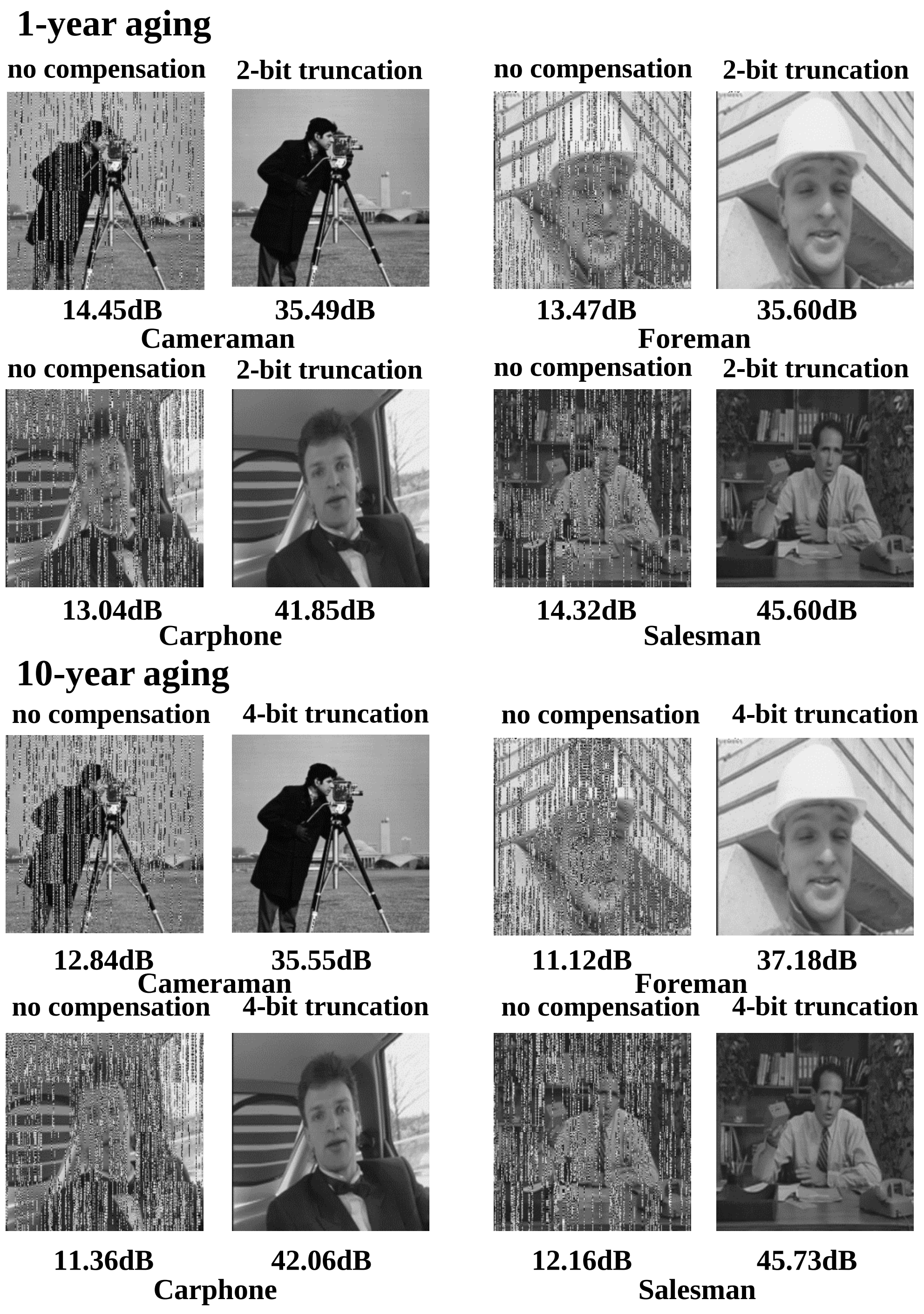


Figure 1.10: Evaluation of aging compensation of proposed adder with approximation in image processing application. The value under the images are PSNR.

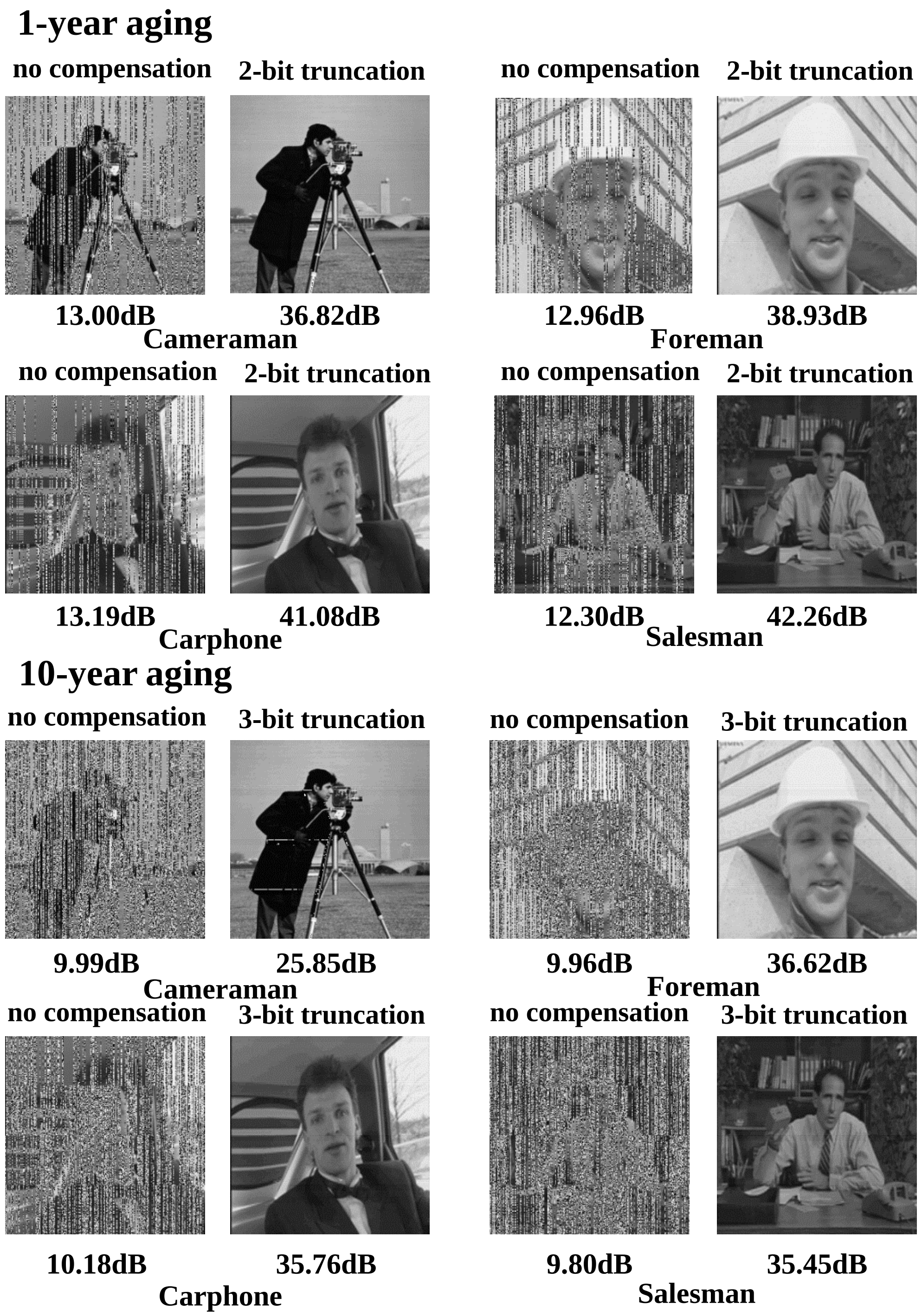


Figure 1.11: Evaluation of aging compensation of proposed multiplier (and proposed adder) with approximation in image processing application. The value under the images are PSNR.

the matrix multiplication unit of the blocks, each of a multiplier and an adder occupies a pipeline stage. different monitoring circuits to each of the adder and the multiplier are integrated. Experiments for the adder and the multiplier are performed independently, then a experiment for the combination of the two, which is a MAC unit, is performed.

Figure 1.10 shows the experimental results for the adder. Note that in this experiment, a delay-optimized multiplier (having high power consumption) instead of the proposed multiplier employ is utilized so that the adder is on the critical path. As shown in the figure, the four images show similar degradation due to the aging. For example, PSNR of “Cameramen" is degraded down to 14.45dB after 1-year aging and 12.84dB after 10-year aging, respectively. It means that without the proposed scheme the reliability guardband is essentially required even in error-tolerant applications, such as image processing, which do not immediately fails by timing violation at the cost of quality degradation. The fact that most of the aging occurs within the initial 1 year of the expected lifetime is also confirmed as observed in the RTL component level analysis.

In the proposed system, the aging-induced delay can be compensated for with computation approximation. For example, in case of 1-year aging, PSNR of “Cameramen" is recovered to 35.49dB by 2-bit truncation as shown in Figure 1.10. In case of 10-year aging, more bits should be truncated in order to recover the image quality. As shown in the figure, 4-bit truncation can recover PSNR of “Cameramen" to 35.55dB. These results indicate that the proposed system can dynamically compensate for the aging-induced delay with computation approximation at run-time, based on the delay information from the monitoring circuit (e.g. 2-bit truncation after 1-year aging and 4-bit truncation after 10-year aging).

Next, a experiment for the 16-bit proposed multiplier is performed. Figure 1.10 shows the experimental results for the multiplier on the four images. Without compensation, multiplier's degradation in PSNR is more severe than that of the adder at equal aging stages, because the rate of timing violations in the multiplier is higher as mentioned in subsection 1.5.2. The compensation recovers PSNR up to acceptable levels by the computation approximation at both 1-year and 10-year aging stages, except for the 3-bit truncation for the "Cameraman" at 10-year aging, whose glitches are caused by the LSP errors of truncation. The reason why these glitches appear only in the case of the multiplier and not in the case of the adder, is that the truncation of the multiplier is directly applied to input images since the multiplications are performed earlier than the additions as shown in Figure 1.9b. The input images are represented as 8-bit integers, therefore truncation of 3 bits out of 8 bits may incur large approximation errors. On the other hand, truncation of the adder is performed on the results of the multiplier which have less significance in their LSBs.

After the independent experiments for each component, the adder and the multiplier, the experiment on the combination of the two, which is a MAC unit, is performed. In the DCT/IDCT codec block, the pipeline stage of the multiplier is directly followed by that of the adder in a MAC unit as shown in Figure 1.9b. The experimental results for the MAC unit, where aging effects on both the adder and the multiplier are considered, are equal to those of the multiplier in Figure 1.10. There are two reasons for this. First, as you can see in Figure 1.9b, a MAC operation operates in the order of multiplication and addition, therefore, the changes in the multiplier output affect the input of the adder directly. For example, if a monitoring circuit truncates two bits of

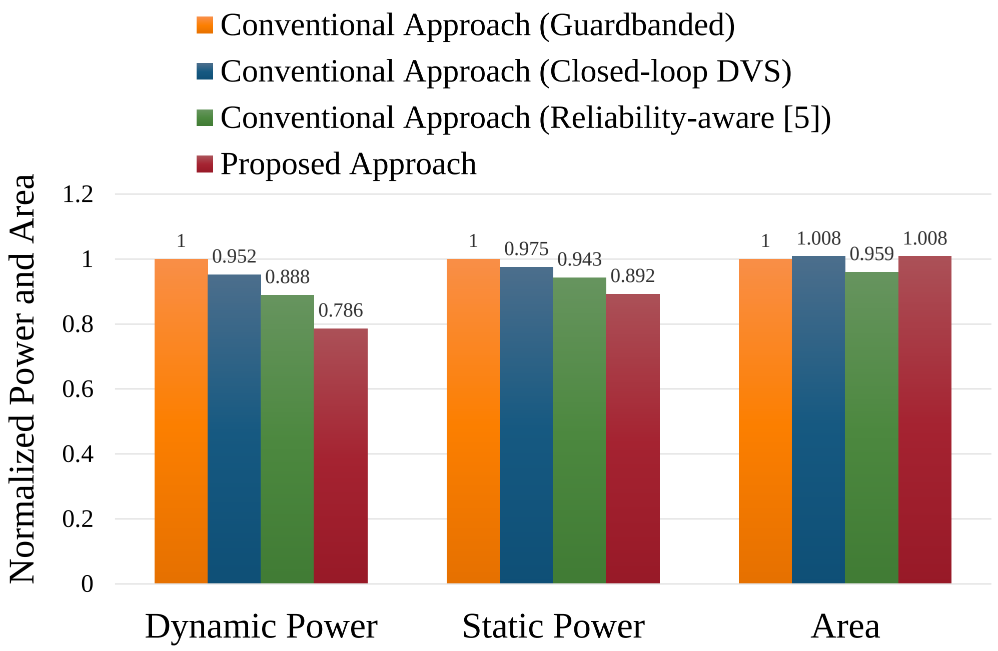


Figure 1.12: Power and area comparison of DCT/IDCT codec block for conventional and proposed approaches. The proposed approach includes the proposed adder and multiplier.

the multiplicand operand of the multiplier, the output of multiplier always has two “0" bits in their LSBs which is conceptually equal to 2-bit truncation for the following adder. Therefore, in this case, the truncation of the adder up to two bits does not have any effect. Second, the adder has shorter critical path delay than the multiplier in the proposed system as shown in Table 1.1 and Table 1.2, and thus the number of truncation bits of the adder is always fewer than that of the multiplier. By such two reasons, the compensation effect of the MAC unit is determined by that of the multiplier in the proposed system.

Figure 1.12 summarizes the DCT/IDCT codec block's power and area comparisons over the four approaches: Guardbanded, Closed-loop Dynamic Voltage Scaling (DVS), the Reliability-aware synthesis from [5], and the proposed approaches. First, here, the way of employing guardband is increasing operating voltage instead of decreasing the clock frequency. Therefore, all circuits including the proposed approach operate at the same clock frequency. The Reliability-aware approach is synthesized with the degradation-aware library (at 10-year) [5] while the others including the proposed approach are synthesized with the degradation-unaware library, which is a subset of the degradation-aware library at 0-year. Note that the proposed approach is better when it is synthesized with the degradation-unaware library since the degradation-aware library implicitly contains aging guardband in their delay that is not required in the proposed approach. The pessimistic guardband at design-time is employed in the Guardbanded approach, and the reduced guardband estimated as in [5] in the Reliability-aware approach. By contrast, the Closed-loop DVS and the proposed approaches dynamically compensate the aging-induced delay by raising operating voltage and adjusting computation approximation, respectively.

The proposed approach reduces the dynamic and static power by 21.45% and 10.78%, respectively, compared to the Guardbanded approach. As the delay increases by aging, the Closed-loop DVS approach consumes more power than the proposed approach since it dynamically raises voltage to prevent the aging-induced timing violation errors. The Reliability-aware approach shows lower power consumption than the Closed-loop DVS approach because the former allows the circuit to operate at lower voltage compared to the latter during almost all of the lifetime, but it still shows worse result than the proposed approach which basically does not need guardband. The proposed approach and the Closed-loop DVS have only 0.8% of the area overhead due to the monitoring circuits and control units. In conclusion, in this system, the proposed approach achieves large power reduction with negligible area overhead and acceptable image quality degradation.

1.6 Summary

In this chapter, it is proposed that a novel aging compensation scheme and system, consisting of monitoring circuits, control units, and configurable adder/multiplier. It dynamically reduces the precision of the adder/multiplier by monitoring the aging-induced delay at run-time. The proposed adder/multiplier turns numerically significant MSP errors to less significant LSP errors. That is why the proposed system avoids significant image quality degradation without the costly reliability guardband in an image processing application. In addition, the proposed design methodology can be applied to any other circuits as well as ripple carry adders and CSA array multipliers, provided that they can be configured to trade-off the accuracy with delay. In an advanced process technology, the effects of aging on a circuit delay can be much more serious and thus such a dynamic compensation method for mitigating the reliability problem becomes more important. The proposed system can effectively resolve the problem with acceptable computation accuracy degradation, while maintaining low power consumption.

The design methodology currently relies on applying *set_dont_touch* on individual adders and multipliers constraints during synthesis. This reduces flexibility during global optimization and can result in sub-optimal results especially in final layouts obtained after place and route (P&R). Properly dealing with this issue is one of the future works. To mitigate this issue, hierarchical design, which combines block-level implementation and optimization, cloning of blocks from a master block, and systemic methods for assembling the blocks at the whole chip level with well-designed algorithms (block-aware re-routing and aligning blocks at top design), can be utilized. Such hierarchical design has been used widely in industry for scalability reasons and is officially supported by industry-level commercial P&R tools (e.g., Cadence Innovus).

Chapter 2

Energy-Efficient Neural Network by Combining Approximate Neuron Models

2.1 Introduction

2.1.1 Deep Neural Network (DNN)

Recent researches on DNNs (Deep Neural Networks) have demonstrated unprecedented performance in various fields, such as computer vision, speech recognition, and so on, which was not possible decades ago due to insufficient computing power available at that time. Over the last decade, however, the computing power has increased dramatically, and computer systems of today provide an enough level of throughput to realize quite complex DNNs. In particular, GPUs (Graphics Processing Units) [37] are the most powerful device to compute floating-point matrix operations that occupy the majority of DNN operations. However, high power consumption of GPUs is still a problem in power-constrained systems such as embedded systems.

2.1.2 Low-power designs for DNN

In order to provide high computation throughput with low cost and power, many studies have proposed specialized hardware implementations of DNNs. For example, hardware implementations with ASIC [38] or FPGA [39] have tried to reduce their cost by using fixed-point operations, which require smaller area and lower power than floating-point operations. The drawback of such approaches is that it can cause overflow or underflow problems during computation because of the narrow dynamic range compared with floating-point operations, and thus degrade the inference accuracy. Another research [40] has adopted a dynamic fixed-point concept for DNNs, which dynamically changes the fixed-point range (i.e., radix point) according to overflow rate. In their work, the radix point of each group is determined during training phase.

2.1.3 Stochastic-Computing Deep Neural Network

SC-DNN (Stochastic-Computing DNN) [41] is another promising alternative to low-power DNN hardware implementation. It represents values using stochastic numbers in a bitstream form, corresponding to binary numbers of other conventional hardware. Stochastic numbers typically take one of two different encodings: *unipolar* and *bipolar* encoding (see Figure 2.1). The unipolar encoding represents a value as the probability of a bit in the bitstream to be '1', and thus the value has range [0, 1]. On the other hand, the bipolar encoding represents a value as (probability of a bit to be '1') − (probability of a bit to be '0'). Thus the value has range [-1, 1]. The benefit of using stochastic number representations is that traditional binary logic implementations of arithmetic

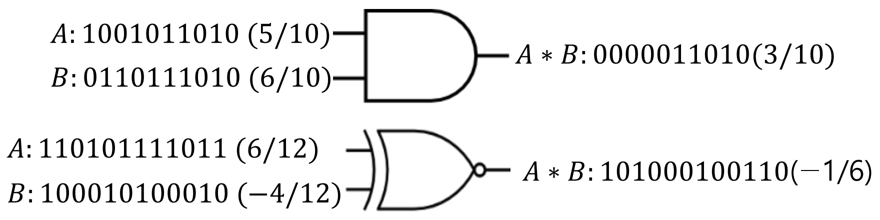

A: 1001011010 (5/10)
B: 0110111010 (6/10)
$A * B$: 0000011010(3/10)
A: 110101111011 (6/12)
B: 100010100010 (−4/12)
$A * B$: 101000100110(−1/6)

Figure 2.1: Example of stochastic number and their multiplication. AND and XNOR gates are used for unipolar and bipolar encoding, respectively.

operations such as multiplications and additions can be replaced with much simpler stochastic logic implementations. For example, a stochastic multiplier can be implemented by using a single AND (or XNOR) gate for unipolar (or bipolar) encoding. In unipolar encoding, for example, a value is represented by the probability of observing '1's in a bitstream and an AND operation of two stochastic numbers (bitstreams) gives the probability of both stochastic number having '1' (i.e., it gives the product of the two probability values, provided that the two bitstreams are independent), as can be seen in Figure 2.1.

Another important benefit of using stochastic number representations in an SC-DNN is the flexibility of precisions, or *progressive precision* [42]. The floating- and fixed-point implementations have predefined precisions determined at design time. On the other hand, the precision of stochastic computing (both unipolar and bipolar) is flexible because it is determined by the length of the bitstream for a stochastic number. In other words, the precision of stochastic computing can be increased/decreased in proportion to the increase/decrease of the bitstream on the same hardware hardware design. Therefore, SC-DNNs can change their precision dynamically without additional hardware cost [41].

2.1.4 Spiking Deep Neural Network

It has been shown that the behavior of an IF (Integrate-and-Fire) spiking neuron (spiking neuron in short throughout this chapter) can be modeled as a ReLU (Rectified Linear Unit) activation neuron [43]. The IF spiking neuron can be formulated in two steps as follows:

(Step 1. Update membrane voltage)

$$V_{mem,j} = V_{mem,j} + \sum_i W_{ij} \times \delta_{ij}$$

(Step 2. Check firing condition)

$$V_{mem,j} = \begin{cases} V_{reset}, & \text{if } V_{mem,j} > V_{th} \\ V_{mem,j}, & \text{otherwise} \end{cases}$$

where $V_{mem,j}$ is membrane voltage of the j-th neuron, i is an index of neurons in the preceding layer, W_{ij} is the weight of a synapse connecting the i-th neuron of the preceding layer to the j-th neuron in the current layer, and δ_{ij} indicates the incoming spike through the corresponding synapse. V_{reset} and V_{th} represent reset voltage and threshold voltage, respectively. the value of V_{reset} is set to $V_{mem,j} - V_{th}$ as was done in [44]. In the first step, the spiking neuron takes input spikes through synapses multiplied by their weights and then accumulates those values into the membrane voltage. In the second step, if the membrane voltage exceeds the predefined threshold voltage, the neuron fires a spike to their output synapses. After firing the spike, the membrane voltage resets to V_{reset}. Those operations are performed in a continuous time domain in biological neurons, but in this chapter, spiking neurons are modeled to operate in a discrete manner.

The spiking neuron processes the inputs δ_{ij} having value '0' or '1'. '0' means no spike, and '1' means spike firing. It is similar to unipolar encoding in stochastic computing. Therefore, each discrete time slot can be matched to a bit of each stochastic number, and set the length of a stochastic number the same as the number of time slots. a spiking layer is made with spiking neurons, and it constructs a *SP-DNN (SPiking DNN)* by stacking themselves.

2.2 Hybrid of Stochastic and Spiking DNNs

2.2.1 Stochastic-Computing vs Spiking Deep Neural Network

Compared to SC-DNN, SP-DNN has some advantages. First, it has shorter latency to reach the same level of accuracy. While a stochastic neuron processes only a bit at a time, a neuron in SP-DNN processes multiple bits at a time (a single bit can be used for an incoming spike, but multiple bits are used for internal IF operations.) According to the experiment in [41], SC-DNN requires bitstream length of 1024 to archive accuracy close to that of a floating-point DNN (i.e., test error 2.41% for the MNIST dataset). In contrast, it is observed that SP-DNN processes a spike represented by a single bit as if it had a multiple-bit value, since every time a spike comes into a neuron, a multiple-bit weight value is accumulated to the membrane voltage. Therefore, the latency required to achieve the same level of accuracy is shorter than that of SC-DNN.

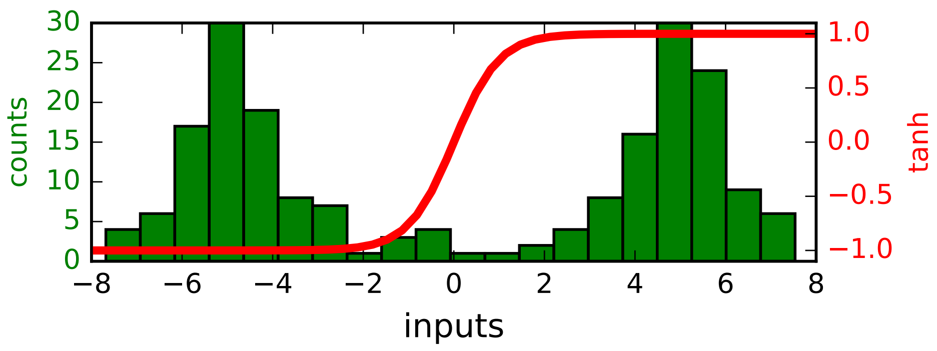


Figure 2.2: The histogram shows the distribution of input values to the activation function in the first layer of SC-DNN for a test image. The activation function of *tanh* is drawn on top of the histogram to show that most of the inputs exist in the saturation regions of the function.

Secondly, SP-DNN provides a better interface to the sensors that generate raw input data. Typical bio-inspired event-driven sensors (a.k.a. neuromorphic sensors) [45] generate event data in the form of spikes, so as to enable spiking-based device to process them immediately. The work in [46] made neuromorphic handwritten dataset (i.e., N-MNIST) by recording the MNIST handwritten dataset using a neuromorphic sensor called ATIS [45]. It is reasonable to consider the MNIST dataset as ideally recorded image and the N-MNIST as raw input data. A major difference between the two datasets is their value range. The MNIST has [0, 255] value range of gray-scale in the original dataset but it is normalized to [0, 1] by dividing the values with 255. However, in reality the value is concentrated in a very narrow range. In the case of N-MNIST, for example, all the values are in the range of [0, 0.17]. This is due to the characteristics of actual sensors.

The neuromorphic sensor generates data as a sequence of spikes, which looks like unipolar encoding for stochastic computing. However, the SC-DNN is better implemented using bipolar encoding to represent negative weights.

Thus the input in the form of unipolar encoding should be converted to bipolar encoding when SC-DNN is to be used. That conversion is linear mapping through mapping function $f(x) = 2x - 1$. It transforms the original value range of [0, 1] to [-1, 1]. The problem of this approach is that the values after the transformation are severely biased toward -1 (in the case of N-MNIST, the values will reside only in [-1, -0.66]). As shown in Figure 2.2, the problem still exists at the input of the activation function (tanh is used for the activation function) after multiplication with weights and accumulation. Most of the inputs are located in the saturation region of the activation function, which makes the training phase stick in a local minimum.[1]

2.2.2 Combining Spiking Layers and Stochastic Layers

Based on the observations discussed in subsection 2.1.4, a spiking layer is introduced in SC-DNN to shorten the latency and avoid accuracy drop due to input data conversion. Basically, both of them have relatively low power compared with other conventional DNN implementations (e.g., GPU based DNN). As mentioned in subsection 2.1.3, in stochastic computing, the fundamental arithmetic operations can be conducted efficiently with simple gates. The operations in a spiking neuron do not require a multiplier. Instead, it just accumulates the synapse weight on incoming spike and then sends out an outgoing spike when the accumulated value (i.e., membrane voltage) exceeds the threshold. The proposed scheme combines spiking neurons and stochastic neurons for a synergy effect, while trying not to lose their advantages. In order to maintain the efficiency, only the first layer in SC-DNN is replaced with a

[1]Indeed, floating-point DNN for N-MNIST with spike frequencies converted to floating-point numbers gives accuracy of 96.25%, but if floating-point numbers are transformed to $f(x) = 2x - 1$, then the accuracy is degraded down to 92.09%.

spiking layer as depicted in Figure 2.3. That is because the spiking layer is more expensive than a stochastic layer in terms of power consumption. Moreover, it is also observed that only a single layer placed in the front is enough to alleviate the problem caused by the data format conversion for the raw input data as discussed in subsection 2.2.1. Placed at the most front end of the network, the spiking layer extracts features of input data and sends the output spikes to the following stochastic layers, which enables the Spiking and Stochastic DNN (SPSC-DNN) to have shorter latency than SC-DNN.

The work in [41] proposes the *EDT (Early Decision Termination)* method, which exploits probabilistic independence of each bit composing a stochastic number, i.e., a fraction of the bitstream for a stochastic number also holds the value information of the number. For example, a stochastic number '11010011001101110100' represents '0.55' (i.e., 11/20) in the unipolar encoding. When only the first 10 bits are shown , '1101001100', it represents '0.50' (i.e., 5/10) which is close to '0.55'. Thus, it is sometimes good enough to evaluate only a fraction of a stochastic number depending on the accuracy requirement. The proposed scheme can also exploit the EDT method to further shorten the latency.

Another motivation is the similarity of encoding. As mention in subsection 2.1.4, the sequence of spikes is similar to unipolar encoding, no explicit conversion is needed. The implicit conversion to bipolar encoding is explained in the following subsection.

2.2.3 Encoding Mismatch

Since they use different types of encoding, there is an *encoding mismatch* in that they translate bitstreams as value. Therefore, there should be a converter

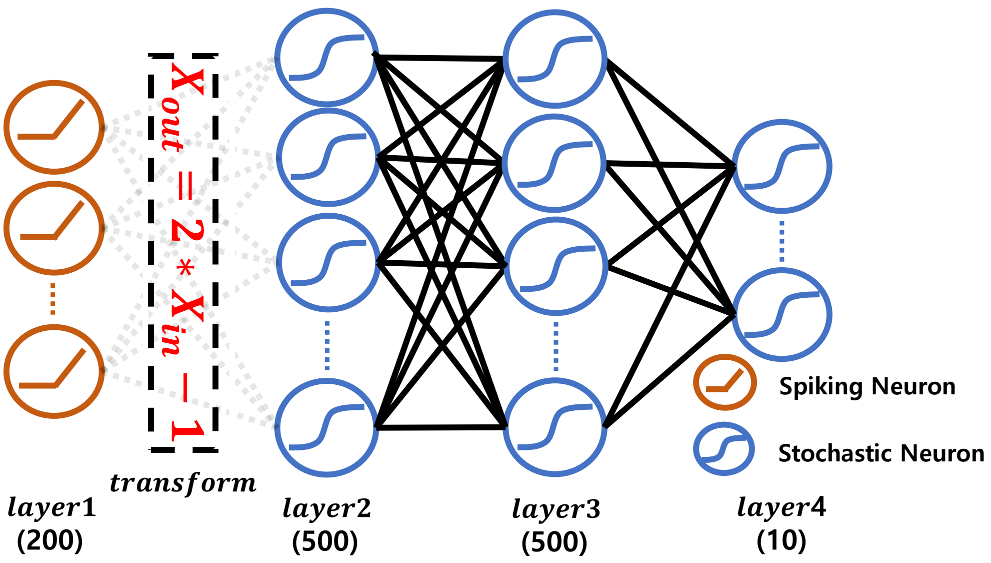


Figure 2.3: Topology of SPSC-DNN. The transform layer is placed to emulate encoding mismatch between the spiking neuron and stochastic neuron in training phase. SC-DNN and SP-DNN also have the same topology, except for different type of neuron. Each layer has 200-500-500-10 neurons, respectively.

that coordinates the mismatch. For example, if a spiking neuron sends out '1011001101' bitstream meaning 0.6 in unipolar encoding, but the stochastic neuron will translate it as 0.2 as follows:

$$\begin{aligned} unipolar : P(X = 1) = 6/10 &= 0.6 \\ bipolar : P(X = 1) - P(X = 0) = 6/10 - 4/10 &= 0.2 \end{aligned}$$

where $P(X = k)$ is probability of a bit X to be 'k'.

In order to avoid additional hardware, inconsistency is resolved by adjusting the synapse weights, which is done in the training phase. The proposed approach is inserting a *transform layer* between the spiking layer and following stochastic layer as can be seen in Figure 2.3. The transform layer is designed based on the relationship between the two encodings, which can be expressed as $P(X = 1) - P(X = 0) = 2P(X = 1) - 1$, which equals to relationship as in subsection 2.1.4. Since the relationship is a linear transformation, the transform layer can be easily integrated in the training phase. According to the experiment, it emulates the inconsistency between the different encodings quite well. Since it is introduced only for training phase, it should be removed after the training. As a result, it will not exist in actual hardware implementation.

2.3 Evaluation

All experiments are conducted on SC-DNN, SP-DNN and the proposed SPSC-DNN. They are synthesized with identical topology (i.e., same number of stacked layers and same number of neurons within each layer), except for the type of each neuron.

2.3.1 Latency and Test Error

Since the precision is proportional to the length of bitstream for stochastic and spiking DNNs, the *test error* (i.e., the rate of failed classification on test dataset) decreases as the bitstream becomes longer. the DNNs are evaluated on both of MNIST and N-MNIST as shown in Figure 2.4. As expected, the test error of SP-DNN decreases faster than other two. SPSC-DNN decrease slower than SP-DNN, but much faster than SC-DNN. Among the three implementations, SPSC-DNN achieves the lowest test error, 1.65% and 2.90% on MNIST and N-MNIST, respectively. As already explained, SC-DNN shows degraded accuracy to raw input data like N-MNIST.

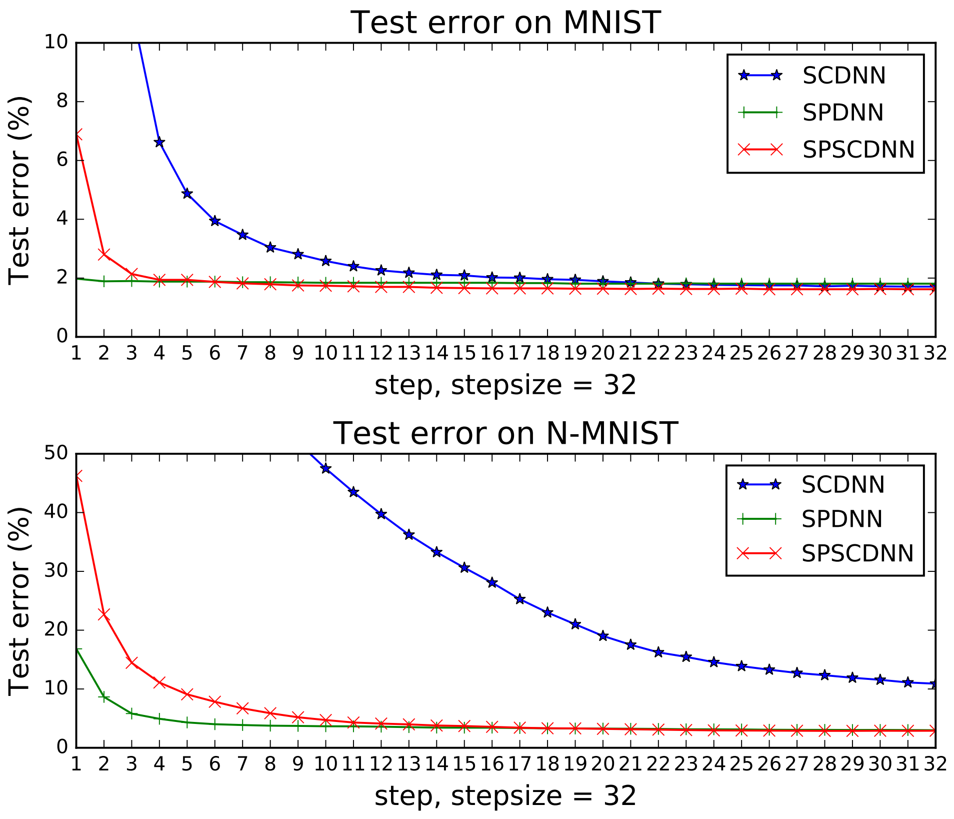


Figure 2.4: X-axis values are in number of steps; each step corresponds to 32 bits in the input bitstream (thus the total length of a bitstream is $32 \times 32 = 1024$ bits).

In order to compare the three schemes while considering the length and test error simultaneously, the target test error is set to 2.00% and 1.81% (SC-DNN could not achieve the error rate of 1.81%) for MNIST and 3.08% (again, SC-DNN could not achieve the error rate of 3.0%) for N-MNIST, and then compare the length (i.e., latency) which is required to achieve the target test error. Those lengths are shown in Figure 2.5.

2.3.2 Energy Efficiency

In order to evaluate energy efficiency, a spiking neuron and a stochastic neuron are synthesized for each layer. Synopsys Design Compiler and TSMC 45nm technology library are used for the synthesis. The spiking neuron consists of MUXs controlled by the incoming spikes (if there is a spike, the weight is added; otherwise, 0 is added) and an accumulator for membrane voltage. On the other hands, the stochastic neuron consists of SNGs (Stochastic Number Generators) for converting binary numbers to stochastic numbers, XNORs for multipliers, a simple counter to add stochastic numbers, and an FSM for tanh function. The SNGs dominate in power consumption within a stochastic neuron. They consume about 70% of the overall energy according to the synthesis results. The recent work [47] proposed *Spintastic* which exploits spintronic technology to implement an energy-efficient SNG that is seven times better than the conventional SNG based on CMOS. It is used in experiments. For synapse weights, 10 bits fixed-point numbers are used. After the synthesis, their power (see Table 2.1) is measured, and the energy consumed in each of the networks approximately is calculated by extrapolation as follows:

$$E_{SC} = (P_{stoc,1}L_1 + P_{stoc,2}L_2 + P_{stoc,3}L_3 + P_{stoc,4}L_4) \times T_{SC}$$

$$E_{SP} = (P_{spike,1}L_1 + P_{spike,2}L_2 + P_{spike,3}L_3 + P_{spike,4}L_4) \times T_{SP}$$

$$E_{SPSC} = (P_{spike,1}L_1 + P_{stoc,2}L_2 + P_{stoc,3}L_3 + P_{stoc,4}L_4) \times T_{SPSC}$$

where E_{SC}, E_{SP}, and E_{SPSC} denote the energy of SC-DNN, SP-DNN, and SPSC-DNN, respectively. $P_{stoc,i}$ and $P_{spike,i}$ denote the power of stochastic and spiking neuron, respectively, in the i-th layer. L_i denotes the number of neurons consisting of the i-th layers and T_{SC}, T_{SP}, and T_{SPSC} denote the latency until they reach the target test error. Both neurons are synthesized with the same frequency (i.e., 1GHz), and thus the relative comparison of latency can be done by the length. The calculated energy of them is shown in Figure 2.5. The power consumption is highest in SP-DNN, but it shows the best performance in terms of latency. SC-DNN consumes the lowest power but suffers from the longest length. Those observations say that there is a trade-off between the power consumption and length. SPSC-DNN is standing in the middle of them. Energy is measured considering the power consumption and the length simultaneously, For the relatively high test error target of 2.0% on MNIST, SP-DNN reaches the target much faster than the others, so SPSC-DNN shows slightly lower energy efficiency when the EDT is applied. However, for 1.81% on MNIST and 3.08% on N-MNIST, SPSC-DNN shows the best energy efficiency.

Table 2.1: Power (mW) Measured on Stochastic and Spiking Neuron of Each Layer Where The Number of Inputs of The Neurons in The 1st, 2nd, 3rd, and 4th Layers is 784, 200, 500, and 500, Respectively.

Neuron Type	$P_{.,1}$	$P_{.,2}$	$P_{.,3}$	$P_{.,4}$
Spike ($P_{spike,.}$)	37.60	12.69	22.02	22.02
Stochastic ($P_{stoc,.}$)[1]	9.85	2.45	5.19	5.19

[1] with *Spintastic* SNG.

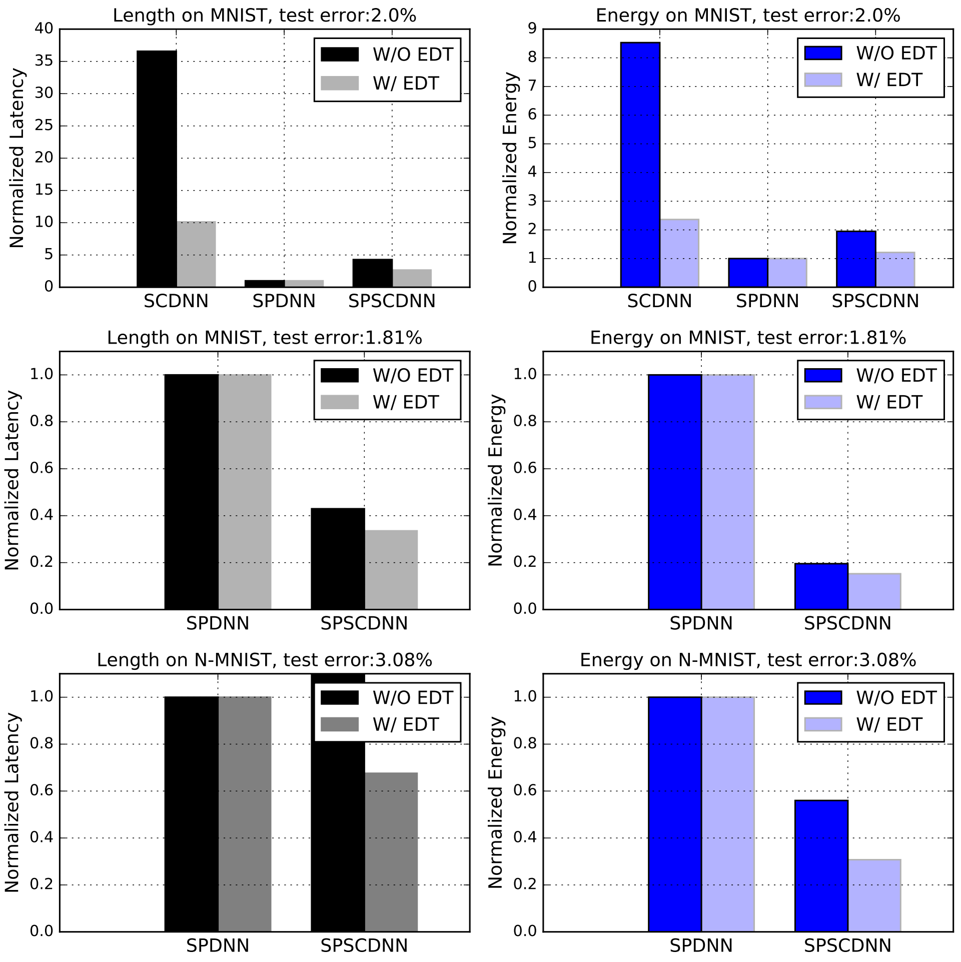


Figure 2.5: Latency (i.e., length) and energy consumption in SC-DNN, SP-DNN, and SPSC-DNN for different dataset and target test error. The effect of EDT is also evaluated. Since SC-DNN cannot reach the target test error for 1.81% on MNIST and 3.08% on N-MNIST, it is not included in the graphs on the second and third rows.

2.4 Summary

We notice in this work that SP-DNN has shown much shorter latency than SC-DNN. Furthermore, they have similar encoding for the representation of values using bitstreams. However, to combine the two schemes for a better performance, the issue of encoding mismatch should be addressed. By inserting a transform layer, they can be combined successfully without accuracy drop, and the combined DNN (SPSC-DNN) shows shorter latency than SC-DNN and lower energy consumption than SP-DNN.

Chapter 3

GradPIM: In-memory Gradient Descent in Mixed-Precision DNN Training

3.1 Introduction

3.1.1 Neural Processing Unit

Training a deep neural network (DNN) is a time-consuming process. For instance, training the widely-used ResNet-50 [48] on ImageNet dataset [49] requires about a week of GPU time for training. Building a specialized neural processing unit (NPU) is a promising approach for achieving both speedup and energy efficiency. As DNN emerges as one of the most significant applications of the era, a number of NPU designs are being proposed for both inference and training [50, 51, 52, 53, 54, 55, 56].

One important issue in designing an NPU is minimizing its memory bandwidth requirements. By naively executing DNNs on hardware, one would suffer from the heavy traffic caused by repeated memory accesses on the same data. Hence, a large stream of work on NPUs is focused on maximizing data reuse. For example, many accelerators [50, 52, 57, 58, 53, 54] propose different dataflow models that handle data reuse problem by loop reordering and careful address mapping to the PE array. Minibatch serialization [59], BN fission and fusion [60], and layer fusion [61] fall into the class of inter-layer optimizations that finds opportunities of data reuse by applying inter-layer optimizations to increase the amount of data reuse.

3.1.2 Mixed-precision Training

One major direction that NPU technology is heading toward is mixed-precision training [62, 63, 64, 65, 66]. Mixed precision training performs numeric operations in low precision while keeping a copy of their master weights in high precision. It can preserve training accuracy while reducing computational costs and memory bandwidth consumption. NVIDIA TensorCore [67] is a direct example of a mixed-precision training algorithm that is being applied to GPUs today. It performs 16-bit operations and 32-bit accumulation to obtain a performance boost. Such techniques are being extensively used in many cases [68, 69], contributing to the reduction in DNN training cost. There are also a few attempts aimed at further reducing training data precision to 8 bits [62, 63]. By applying techniques such as custom floating-point representation and chunk-based accumulations, quantization error can be mitigated to achieve almost no accuracy loss with 8-bit datapaths.

However, the common consensus is that the master weights need larger precision, while the gradients are the most forgiving to the quantization errors, as demonstrated in [66] with 16-bit training with 32-bit master weights. [62] is one popular work which uses 8-bit floating point format for all executions, but keeps the master weight at 16 bits to mitigate the swamping effect [70]. In [62], techniques called chunk-based accumulation and floating point stochastic rounding are introduced, where the former organizes the order of the additions to reduce the average gap of range between the two add operands, and the latter alleviates the loss coming from a smaller number being added to a larger number. Later, [64] suggested improved rules for loss scaling and stochastic rounding, and [63] proposes a custom floating point format that provides a better dynamic range for the training. In addition, there is work on using a hybrid of floating point and fixed point precision [65] because low precision fixed point format has better precision than floating point format but lacks the dynamic range.

In this work, the proposed NPU adopts 8-bit floating point training while maintaining 32-bit fixed point master weight, which stably provides a good accuracy with relatively low hardware cost. Also, it adopts widely used techniques such as chunk-based accumulation, and custom floating point formats for DNN training.

3.1.3 Mixed-precision Training with In-memory Gradient Descent

The proposed NPU is configured to process 8-bit floating-point MAC operations along with a best-effort minimization of memory bandwidth consumption by using on-chip memories and applying inter-layer optimizations. Counter-

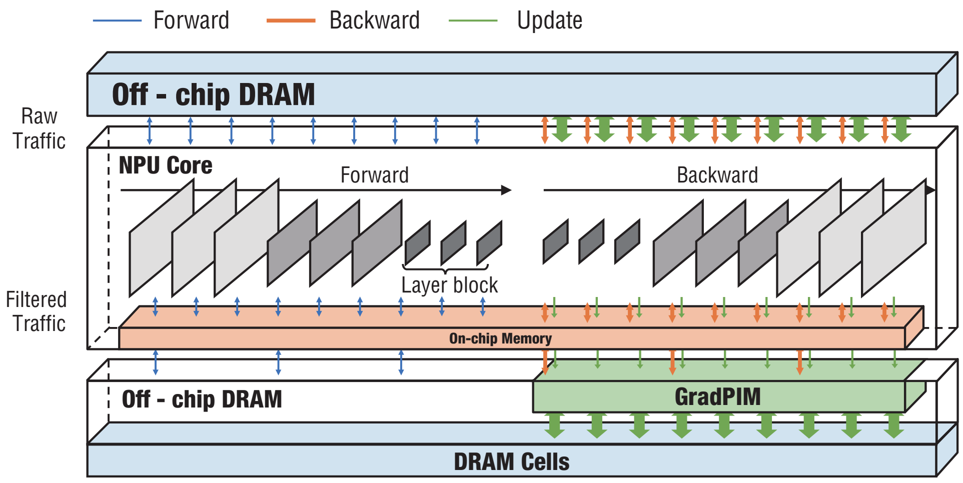


Figure 3.1: Mixed-precision training with GradPIM.

intuitively, after applying the state-of-the-art data reuse techniques, the parameter update phase of the training is identified as the bottleneck of DNN training. From reading and writing high precision master weights and the associated parameters, the parameter update phase consumes up to about 50% of the memory bandwidth consumption and 25% of the execution time. Moreover, even though the computational intensity is very low, there has to be a dedicated high precision datapath for the high-precision parameter updates in the NPUs. Keeping the master weight in a fixed-point is one way to reduce the hardware cost [71], but becomes a burden to the NPU in both processing and transferring of high-precision data. Unlike the other parts of the training process, the operations involved within the update phase have been overlooked in many designs. Those operations are relatively simple, and there is not much room for further optimizations. However, being both simple and memory-intensive makes this an excellent target for processing-in-memory.

To this end, GradPIM is proposed, a fixed-function PIM with parameter-update logic within the DRAM die. Conceptually, it is attempted to isolate the memory traffic associated with the parameter update phase within the memory with GradPIM (Figure 3.1), similar to the forward and backward traffics being isolated within the NPU by data reuse techniques.

The bank group I/O gating is identified as the ideal place for placing the parameter update logic. By placing registers next to the bank group I/O gating, each bank group is effectively decoupled from the global DRAM structure. With careful data arrangements, a set of DDR-based PIM operations, which can perform parameter updates using the internal parallelism of the DRAM, is implemented.

GradPIM is designed to be a simple extension from the existing DDR4 protocol [72]. The design is non-invasive to the DRAM cell arrays and places only a small additional module along with the peripherals. Those modules are fully controlled by the memory controller using a reserved command (i.e. RFU), and thus GradPIM can be considered a DDR-compatible device rather than functioning as an independent accelerator.

3.1.4 DNN Parameter Update Algorithms

Majority of DNNs rely on a family of gradient descent as the parameter update algorithm. In the simplest form of SGD (stochastic gradient descent), all parameters in the network are updated towards the negative direction of the gradient at the end of each minibatch execution. The update can be formulated as below:

$$\theta_{t+1} = \theta_t - \eta g_t \tag{3.1}$$

where g_t is the gradient vector over the current minibatch and η is the learning rate. To gain faster convergence and/or better accuracy, many advanced parameter update algorithms have been proposed. For example, SGD with momentum [73] is formulated as below.

$$v_t = \alpha v_{t-1} - \eta g_t \tag{3.2}$$

$$\theta_{t+1} = \theta_t + v_t \tag{3.3}$$

where α is a momentum decaying factor and v is the momentum. The momentum works as a damping factor, and makes the convergence faster. If weight decay term β is used in addition, Eq 3.2 becomes

$$v_t = \alpha v_{t-1} - \eta(\beta\theta_t + g_t) \tag{3.4}$$

There are a few more parameter update algorithms worth mentioning such as Adam [74], AdaGrad [75], NAG [76] or RMSprop [77]. For example, RMSprop is formulated by

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{3.5}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{3.6}$$

These algorithms all exhibit element-wise computations and has a relatively low number of computations per element. However, they usually require higher precision especially due to their small hyperparameters (e.g., $\eta = 0.01$). Thus, this can easily become the bandwidth bottleneck in the mixed-precision training.

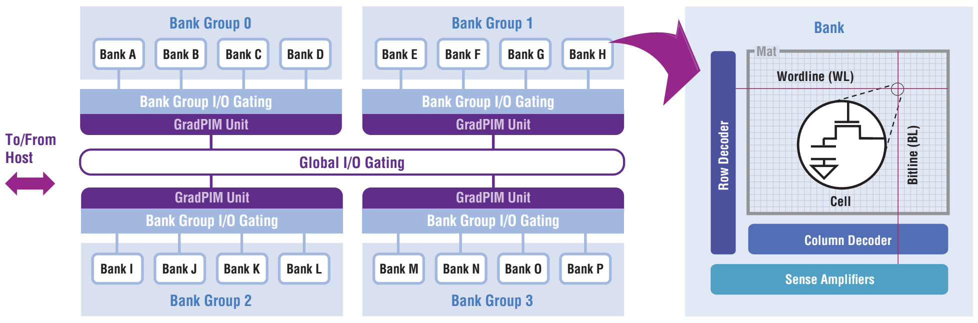


Figure 3.2: Modern DDR4 SDRAM internal architecture.

3.1.5 Modern DRAM Architecture

The modern DRAM architecture is a result of multi-decade effort of continuously increasing cell density and bandwidth at the same time. Figure 3.2 shows the modern DDR4 SDRAM architecture, except for the 'GradPIM Unit's that is proposed to be added. a DRAM is composed of multiple banks, which are 2D arrays of 1T1C cells. To increase the area efficiency, a row of cells in a bank share a wordline (WL), which is used to select the row that should be activated. A vertical set of cells in a bank share a pair of bitlines (BL, BLB) that is used to deliver the data from/to the cells. When a row is activated, the small charge stored within each cell flows out to the bitline and is caught by a row of sense amplifiers. The sense amplifiers restore the cell capacitor to its original values, and this takes tRAS time units to complete.

To read data from the activated row, a few bits (column) are chosen. After tRCD from the beginning of an activation, data could be read from the activated row. A few bits (columns) are chosen and are propagated through the I/O gating and to the off-chip data bus. Even though each bank can operate independently, the I/O gating circuitry is shared among them, and each column read command occupies the I/O gating for tCCD time units.

Therefore a back-to-back column read command has to be spaced with tCCD. Also, the data occupies the off-chip data bus for tBURST time units. tCCD and tBURST are usually set to be 4 cycles, providing 64 bytes of data in a burst[1]. After tCL of latency from the assertion of a read command, the data burst starts on the data bus. In the case of a column write, the data flows in the opposite direction. The memory controller can place the data on the bus tCWL time units after the write command assertion, and the I/O gating is again occupied after tCCD time units.

If a different row has to be accessed for read or write commands, the current row has to be deactivated (precharged) and a new row has to be opened. However, if a previous read or write command is on-going on a column, the row has to remain open for tRTP or tWR time units after the previous read/write command respectively, because the row has to provide the data for a read, or has to wait for the data to be restored for a write.

An important concept introduced since DDR4 [72] is *bank groups*. To keep up the internal fetch speed with the increasing off-chip data rate, multiple banks (2,4, or 8) form a bank group, and the I/O gating is partitioned into bank group I/O gating and global I/O gating. The result is that if consecutive column reads/writes are asserted to two different bank groups, the accesses only share the global I/O gating, and still can be spaced with 4 cycles (=tCCD_S) as in previous generations (i.e., DDR3 [78]). However, if those accesses are to the single bank group, the data occupies both the bank group I/O gating and the global I/O gating, and the two accesses now have to be scheduled with a longer interval (=tCCD_L) in-between. Usually, tCCD_L is 25% to 100% longer than tCCD_S.

[1]In fact, multiple chips co-operate as a rank to provide 64 bytes in a 4 cycle burst. the details are omitted for brevity.

With the introduction of bank groups, there are two kinds of opportunities for in-DRAM processing-in-memory (PIM) technologies to make use of. First, bank-level parallelism exploits the fact that each bank can be independently activated, and a single bank alone can provide more than half the bandwidth of the off-chip data bus (the ratio depends on tCCD_L). Thus, a DDR4 SDRAM chip with 16 banks has more than 8x bank-internal bandwidth, multiplied by the number of ranks per channel. On the other hand, each bank group can also provide more than half the bandwidth of the off-chip bus. Therefore there is more than 2x (for DDR4) or 4x (for DDR5) bank group-internal bandwidth, again multiplied by the number of ranks. Each bank group has access to multiple banks and therefore provides an opportunity to work with multiple open rows in separate banks, which would not have been possible when working only with bank parallelism.

3.1.6 Motivation

subsec:3.1.1.motivation Most NPU designs are focusing on maximizing MAC utilization and minimizing memory traffics during the forward and the backward passes. In full-precision training, the traffics from activation values dominate the entire memory traffics and the execution time, making the above strategy promising. However, with mixed-precision training, the portion of the memory accesses from parameter updates dramatically increase because the parameter update is the only phase that reads and writes the high-precision master parameters from/to the memory, which is 4x the amount of low-precision values (in the 8bit/32bit mixed-precision case).

Furthermore, the state-of-the-art data reuse techniques [61, 59, 60] can reduce the off-chip memory accesses during the forward and backward pass.

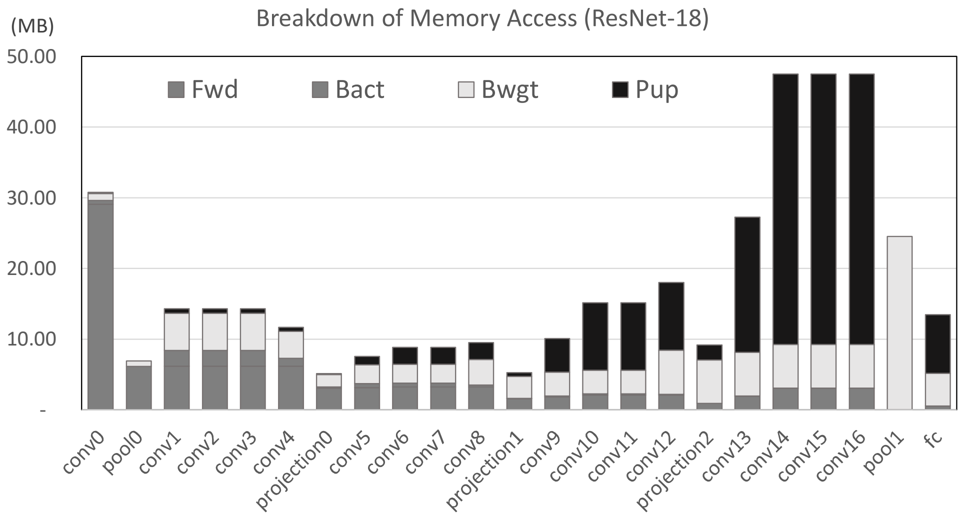


Figure 3.3: Breakdown of the memory access of ResNet-18 layers with mixed precision training.

With such techniques, the portion of the parameter update memory traffics rises to nearly half of the entire training. Figure 3.3 shows the breakdown of the amount of memory accesses in training a batch of ResNet-18, in MB. The bars are divided into forward pass (Fwd), backward pass activations (Bact), backward pass weights (Bwgt), and parameter updates (Pup). For maximizing the data reuse, MBS (MiniBatch Serialization) [59] and BNFF (Batch-Normalization Fission and Fusion) [60] are applied to reduce the inter-layer data traffic. As the network advances to later layers, the portion of the memory traffic caused by parameter updates becomes significant. In the last block, the weight parameter updates take as much as 80.5% of memory traffics. The total portion of the update phase is 45.9%, which is where GradPIM is targeted at. This is in line with the analysis in [79] which reports that offloading packing and quantizing the data to a PIM module can bring a great amount of speedup and energy gain on various applications including TensorFlow Mobile [80]. GradPIM isolates most of the update traffics within the bank group, and the NPU only needs to send the 8-bit scaled gradient to the memory. This leads to a significant amount of execution time and energy reduction, as will be demonstrated in Section 3.5. The portion of traffic in the update phase depends on the ratio of activations and the weight parameters. As the application fields of DNNs expand towards non-CNN workloads such as AlphaGo [81] (playing boardgames) or MLP [82] (general classification and regression problems), we have found that the portion of the weight parameters rises especially for those emerging non-vision DNN applications.

3.2 Previous Work

3.2.1 Processing-In-Memory

Placing a logic inside DRAM is not entirely new, and in fact has a long history of work accumulated over a few decades. Starting from execube [83] in the 1990s, there has been much work on integrating processor and memory on a single die [84, 85, 86, 87], but without commercial success. The idea of PIM revived in mid-2010s, with the aid of 3D stacking technology. By placing a logic die underneath a few memory dies, processing logic could be placed together with the memory cells without having to share the same silicon technology of the memory that slows down the processing speed [88, 89, 90, 91, 92, 93]. Meanwhile, there were attempts to exploit the inherent structure of the existing DDR family to perform a certain class of executions [94, 95, 96, 97, 98, 99]. UpMem [100] has fabricated a processor within a memory die and claims that it can obtain a multifold speedup of the various applications.

3.2.2 Co-design Neural Processing Unit and Processing-In-Memory

DNN as a surging application is another strong driving force towards memory with computing capability. For example, Neurocube [101] is a PIM for executing Neural network using a 3D stacked memory and Tensor-DIMM [102] organizes a specialized DIMM targeting gather and reduction to speedup embedding lookups and tensor manipulations. One major stream of work is at modifying the cell array structure to perform massively parallel operations needed for DNNs. Such an idea has been proposed for DRAM [103, 104, 105, 106, 107], SRAM [108, 109, 110, 111], and emerging memories [112, 113, 114, 115, 116, 117, 118, 119]. However, these technologies all require changing the cell structure itself, and implementing on top of the legacy technologies is complicating (e.g., DDR4/5 protocol).

Compared to most of the PIM solutions above, the GradPIM is a cheaper, easily realizable solution. Solutions such as [103, 104, 105, 106, 107] involve redesigning the DRAM core cell array. While these solutions are certainly better at performance, they are much harder to realize as a product. On the other hand, GradPIM only alters the datapath at the global I/O gating, requiring only a small amount of change in the circuitry. We believe approaches similar to GradPIM has more potential to be accepted to the industry in the nearer future as they fit more smoothly to the current DRAM standards.

BGLP [120] is another work worth mentioning since it exhibits a related idea with the proposal. While not essentially a processing-in-memory work, it decouples the bank group from the other parts of the DRAM. The difference with this work is that they use the buffers to relax the scheduling restrictions, and tries to increase the utilization of the internal banks.

This work is similar to TensorDIMM [102], which organizes a specialized DIMM targeting embedding lookups and tensor manipulations. However, TensorDIMM requires the use of buffer chips to the memory channel, and utilizes only the rank internal bandwidth and requires adding more ranks for the speedup. GradPIM can achieve more speedup as shown in Section 3.5 by utilizing the bank group level parallelism.

3.2.3 Low-precision Computation in NPU

For efficient execution of DNNs, NPU designs are starting to adopt low-precision executions into the datapaths. Nvidia TensorCore [67] is one direct example of a hardware capable of mixed-precision execution that can be found on the market. [121] proposes an interesting idea where most of the computation is operated within the low-precision units, and a few outliers that require high-precision are specially handled to achieve high energy efficiency. Binarized neural networks (BNNs) are also popular targets for efficient executions at the lower extreme, and there have been many NPUs [122, 123, 124], and PIMs [103, 104, 106, 105, 113] specially designed for BNNs. This work also encloses 8-bit datapaths to perform low-precision MACs, but the proposal has a novelty in that it co-design the NPU along with the PIM memory, and split the computation for efficiently isolating the traffic within the NPU and the memory.

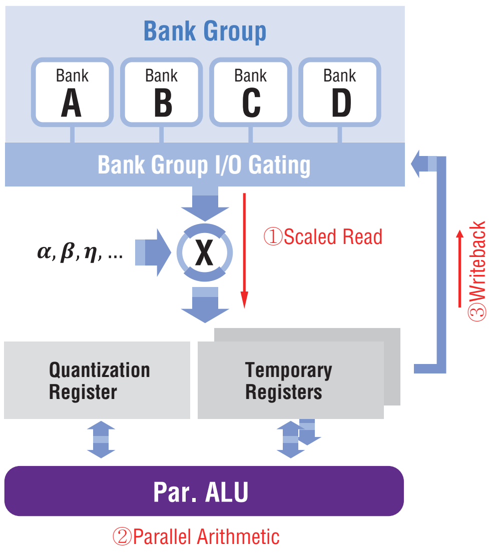


Figure 3.4: DRAM internal architecture of GradPIM unit.

3.3 GradPIM

3.3.1 GradPIM Architecture

Figure 3.4 shows the architecture of GradPIM unit. a GradPIM unit is placed at each bank group, next to the local I/O gating. At the heart of GradPIM is the temporary registers next to the local I/O gating. The temporary registers effectively decouples the local I/O gating and the global I/O gating, hence enabling the bank-group level parallelism. Data is read to or written from the temporal registers, instead of having to occupy the global I/O gating or the external data bus. Also, the vector operation required for the update phase is performed in executing DNNs.

Another benefit of placing the GradPIM unit next to the bank group I/O gating is that it has access to multiple banks. That means, unlike a few previous PIM approaches [96, 95] which performs operations within a bank, GradPIM can operate on multiple rows concurrently at a time. This is essential for working with multiple arrays of variables, which would otherwise cost expensive row activation each time a column of different array has to be accessed.

GradPIM logic mainly includes three components: Registers, scaler, and parallel arithmetic unit.

- *Registers* are used to store intermediate results and have the same width of the global sense amplifiers (i.e., 64 Bytes in total for a rank). Two temporary registers are placed per GradPIM unit to be used for source and destinations of the arithmetic operations, and one quantization register exclusively for storing the 8bit quantized values.
- *Scaler* is used to scale the data with pre-defined hyperparameters, e.g., default learning rate. The scaler is placed between the bank group I/O and the registers, and performs element-wise multiplications.
- *Parallel arithmetic unit* is used to perform element-wise computations within the update phase, such as the additions in Eq. 3.4. In the current version of GradPIM, it supports simple additions and subtractions.

3.3.2 GradPIM Operations

With the components, the operations that GradPIM perform are classified into three categories as below:

1. **Scaled read** loads a column of data into a register from the cells through the sense amplifiers. While loading to the registers, those values are scaled by certain hyperparameter, such as η, α or β as in Equation 3.4. Since those scaling values are mostly fixed constants that have broadly accepted values in practice, four scaler values are pinned to an id to each value. To simplify the scaler, the scaler values are approximated in

$2^n \pm 2^m$ and implemented with combinations of shifters and adders. The values of n and m assigned to each opcode can be programmed with MRW (Mode Register Write) command in case the user needs to use different set of values.

2. **Parallel operations** performs arithmetic operations from the registers and puts the result into another register similar to AVX-512 VPADDD [125] instruction. Currently, GradPIM supports add, sub, quantization, and dequantization to create the partial terms of Equation 3.4 or execute conversion between 8-bit and 32-bit values. Quantization and dequantization processes either read data from one of the temporary registers and write to the quantization register (quantization), or the other way around (dequantization). This decision is made since the 8-bit values stay four times longer in the register, and using a register exclusively for storing them greatly simplifies the data and control path circuit design (see Figure 3.5 for details).

3. **Writeback** After the operations for the optimizers are complete, the result has to be written back. This corresponds to the second half of the DDR column write command, where the register data is written to the global sense amplifier and to the cells.

3.3.3 Timing Considerations

To allow the memory controller to schedule the GradPIM commands along with the existing commands, each GradPIM command has to mingle with the timing parameters. timings are kept for each command as below.

The scaled read is similar to the column read operation in the ordinary

DDR protocol [72], but the data is placed to one of the registers instead of the data bus and therefore does not limit the scheduling of other commands with tBURST. In an attempt to maintain close consistency with the existing DDR commands, the memory controller regards the operation as complete after tCCD_L. In DDR protocol, tCCD_L represents the bandwidth that a bank or a bank group is capable of providing. Because the scaled read operation also reads the data from the banks, it is reasonable to assign the same tCCD_L to read data and be stored in the register. Also, tRTP is still preserved since the sense amplifiers need to provide the data from the cells. Please note that the scaled read occupies only the local bank group I/O gating and thus does not interfere with the other scaled read commands in different bank groups.

The parallel arithmetic operations happen completely out of the conventional DRAM logic and is not governed by the existing timing parameters. To account for the parallel ALU being occupied, an extra timing parameter tPIM, which takes the worst case execution time for the arithmetic operations into account, is introduced. This timing parameter does not interfere with any other commands, but tPIM prohibits other arithmetic operations from taking place within the same bank group.

Writeback operation can be considered as the latter half of the existing write command. Instead of the data bus, the data is coming from one of the registers. Therefore the writeback operation is not affected by tCWL or tBURST, but the tCCD_L is kept as in the scaled read as the bank group I/O gating is occupied. tWR has to comply if the row is to be closed after the writeback since the data propagate into the row through the sense amplifiers.

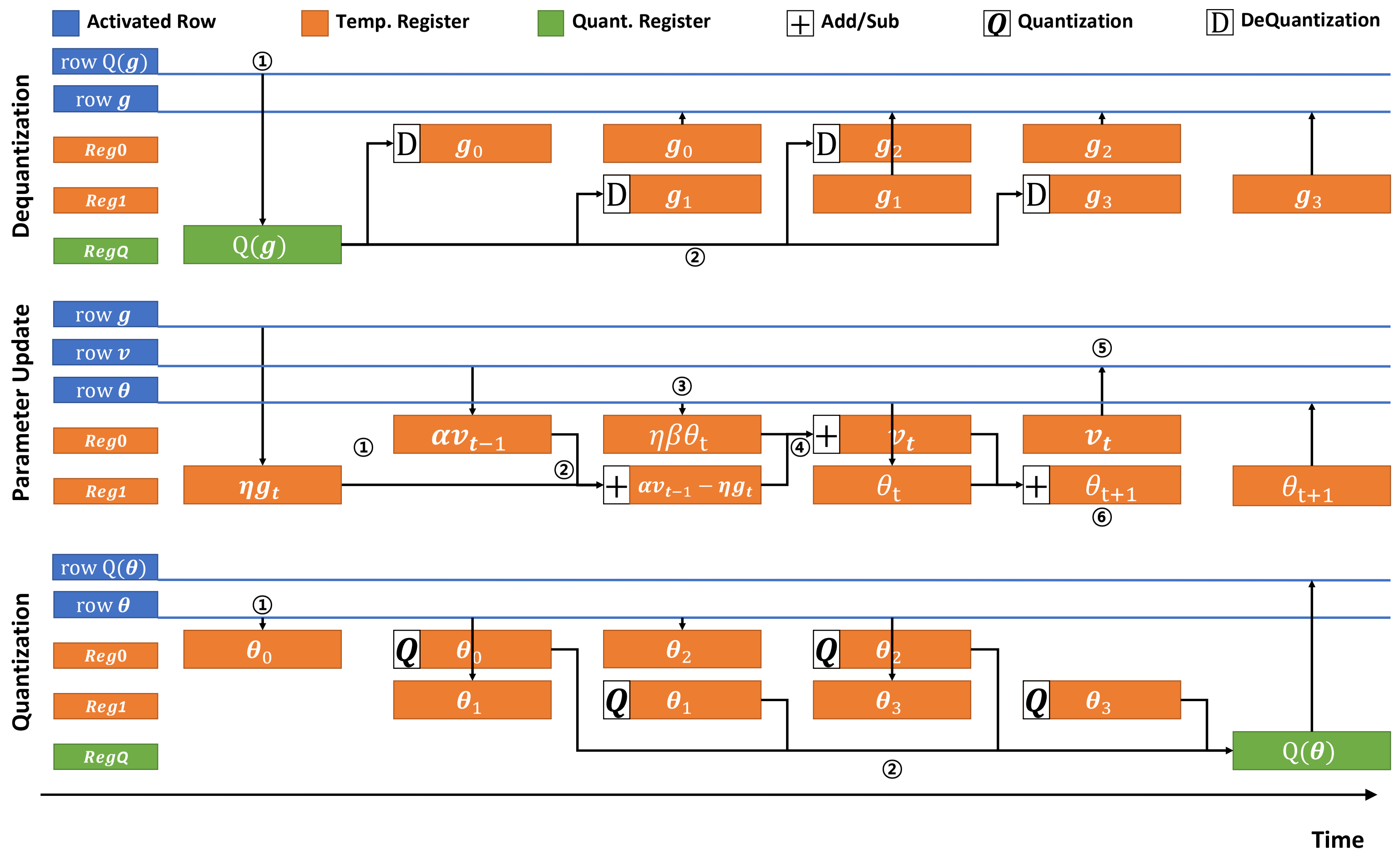


Figure 3.5: Example procedure for quantization/dequantization and momentum SGD algorithm with GradPIM.

3.3.4 Update Phase Procedure

To execute the update phase with GradPIM, the NPU first writes the low-precision gradients to the memory, and dequantization is performed to convert them to high-precision values to be used in the parameter update. Second, the parameter update algorithm is executed as in Eq. 3.3 and Eq. 3.4. Finally, quantization is performed on the updated master weights to convert them to a lower precision, allowing the NPU to read them in the next step. This section shows how these are done sequentially using the operations described in subsection 3.3.2.

Dequantization

Figure 3.5 (Top) shows the procedure for performing dequantization. It is assumed that the rows for the quantized gradients $Q(g)$ and the dequantized gradients g are already open on different banks within a bank group so that they can be open at the same time, and the procedure is as follows: ① A column of $Q(g)$ is loaded into the quantize register. ② A 1/4 column of the $Q(g)$ is dequantized and the resulting column is written to a temporary register. The dequantzation command specifies which 1/4 of the column should be read from the quantize register and which temporary register to write to (please see subsection 3.3.5). Then the gradient (g) is written back to the corresponding row, and ② is repeated four times until the entire column had been dequantized. The procedure is repeated for the consecutive columns of g.

One thing to note is that the entire procedure does not experience any row buffer miss except for when a new row is opened for next data accesses (like a cold miss in a cache) because a unit is placed in the local I/O of the bank group.

Parameter Update

Figure 3.5 (middle) shows the procedure for conducting the update phase with GradPIM using momentum SGD [73] algorithm as in Eq 3.3, 3.4 as a simple example.

It is also assumed that the rows for weight parameters θ, momentum v and gradients g are already open on different banks within a bank group so that they can be open at the same time. ① A column of g_t and v_{t-1} are loaded to the temporary registers, scaled by η and α using scaled_rd operation. ② The scaled values in the two registers above are processed with parallel_add. ③ A column of θ_t is loaded into a temporary register, scaled by $\eta\beta$. ④ Parallel_add is performed once again, creating v_t in EQ 3.4. ⑤ Writeback is performed from the register with v_t to the open row for v. ⑥ Similarly, a column of θ_{t+1} is generated by Equation 3.3 and written back to the row storing θ. Finally, ① - ⑥ is repeated for consecutive columns of g, v and θ until the entire row has been processed.

This procedure also requires no unnecessary row activations as in the dequantization case. In the case of the more complicated algorithm where more than one momentum is used per weight parameter, the required number of concurrent open rows might increase, but there are four banks per bank group in typical DDR4/5 SDRAMs and it is enough to cover all per-weight values in most of the SGD-based parameter update algorithms to my knowledge.

Table 3.1: Truth Table for GradPIM Commands

Func. \ Signal	Op0	Op1	Param0	Param1	Src/Dst
Scaled Read	L	L	Scale ID		Dst
DeQuant	H	L	Src Position		Dst
Quant	H	H	Dst Position		Src
Writeback	L	H	L	L	Src
Q. Reg	L	H	H	L	RD/WR
Add	L	H	H	H	Dst
Sub	L	H	L	H	Dst

Quantization

Figure 3.5 (Bottom) shows the procedure for performing dequantization. As the last step, the master weight parameters are quantized to a low precision, so that the NPU can read them during the forward and backward phase. The procedure is similar to dequantization, but the order is opposite. ① A column of master weight parameters are loaded to a temporary register, and quantization is performed. It fills a quarter of the quantization register, so this is repeated four times. ② Each time the quantization register becomes full, it is written back to the row with $Q(\theta)$. It is repeated for the consecutive columns of θ.

3.3.5 Commanding GradPIM

This work utilizes the RFU (Reserved for Future Use) commands in existing DDR4 protocol [72] to realize the GradPIM commands. According to the standard, there are a number of configurable command signals used for RFU operations. Since all the commands require addresses for bank groups, banks, rows, and columns, it leaves five signals left for configuring GradPIM commands [2].

Table 3.1 shows the truth table for the commands added for GradPIM. For scaled read, 2 bits are assigned for the id of the scaler value and 1 bit for the destination register id. For quantization and dequantization, 2 bits are assigned to designate which quarter of the quantize register has to be accessed, and another bit for the src/dst temporary register id. For writeback and parallel ops, only 1 bit is assigned to denote the src or dst register id. Parallel ops do not require src register ids since there are only two temporary registers that are both used as operands. With quantization register control, since there is only one register, a single bit is assigned for wr/rd.

In case more fields are needed to support additional operations for future extensions, an extra command signal can be added or unused command combinations which are not explicitly stated as RFUs, but not claimed by the standard can be occupied. This would cause slightly more overhead to complicate the command decoder design, but would provide a plenty of command signals for enough flexibility.

3.4 NPU Co-design with GradPIM

3.4.1 NPU Architecture

A NPU used here is designed based on Diannao [53] as shown in Figure 3.6. Modern NPUs often utilize systolic array structures for MACs. However, systolic arrays necessitate performing additions in a sequential way that often suffers from numerical stability problem when using low-precision values due

[2]These are A12/BC_n, A17, A13, A11 and A10/AP. Please refer to [72] for more details. In the case of DDR5, the effective number of free bits increase to six as there are four RFU commands with four free bits each [126]

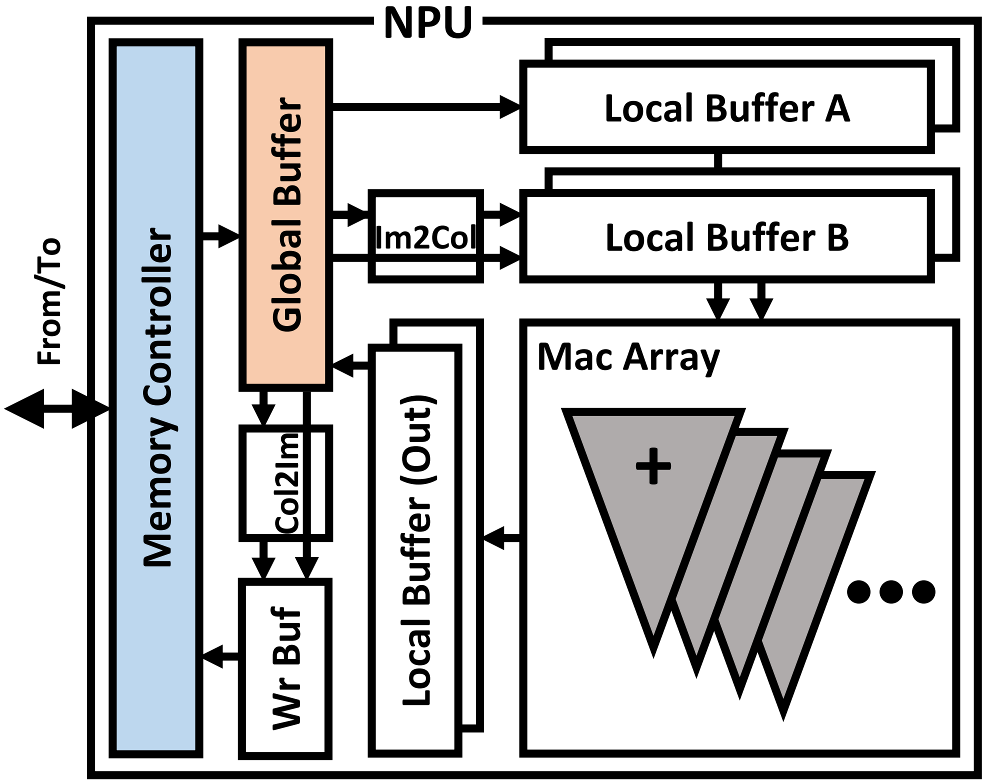


Figure 3.6: NPU architecture of GradPIM.

to swamping [70]. A popular way of solving this on low-precision training is chunk-based additions [62], which gradually adds up the elements in chunks so that there is less divergence between the exponents of the partial sums.

For such reason, the *MAC* array of the proposed NPU is composed of 128 adder trees where each tree receives 128 pairs of 8-bit values and calculates the sum of products to output one 8-bit activation.

To feed the MAC array with continuous stream of data, we adopt the widely used im2col-col2im data flow that converts the input activations into a Toeplitz matrix so that convolution operations can be handled with matrix multiplications as in CuDNN and a few NPUs [59, 127]. The input matrices are partitioned into 128x128 blocks that fit the local buffer to maximize data reuse.

To keep the MAC array utilization high, the widely used im2col-col2im data flow is adopted as in CUDNN and a few NPUs [59, 127]. The input matrices are partitioned into 128x128 blocks so that the local buffer can maximize the data reuse from the global buffer.

There are two sets of input local buffers (one for weight parameters and the other for activations in forward pass) and a set of output local buffers per adder tree. Each input buffer holds the 128x128 elements of the block, which is equal to the size of inputs of the MAC array. The output buffer also has a width of 128x128 bits, matching the size of the partial sum of the resulting matrix block. Each input and output local buffer is double-buffered to maintain the throughput while reading/writing from/to the global buffer. Each cycle, the local buffer of weight parameters provides a row of blocked matrix to each tree, and the activation local buffer provides a column of the blocked matrix to each adder tree. At the end of each cycle, the columns in the local buffer rotate, making a different match between the rows and the columns to be multiplied. This is analogous to the weight-stationary dataflow [57, 58] often used in systolic array architectures [52, 55, 50]. When the partial sum for the entire block is complete, the value is written back to the global buffer, while processing of the next block proceeds, using the values prepared in the other side of the double-buffers.

In order to avoid the memory traffic explosion due to the im2col scheme, a dedicated module is placed for performing im2col between the global buffer and the activation local buffer. The module creates a block of matrix from the image-format activations in the global buffer and writes to the activation local buffer. The global buffer aggregates a few matrix blocks to form a macroblock so that it can hold the right amount of data at a time. The calculated output blocks coming from the MAC arrays are accumulated in the global buffer. When processing is done for the macroblock, it is sent to the DRAM through the write buffer. In case of the backward phase, a dedicated col2im module is used between the global buffer and the write buffer, performing the inverse of an im2col required for the backward pass.

3.4.2 Data Placement

Conventional DRAM subsystems integrate multiple devices to form a data bus with a large width. Consequently, 64-bit data are split into 4, 8 or 16 bits and interleaved among each chip for x4, x8, x16 devices respectively. However, GradPIM requires the entire 32bit within a device in order to perform vector operations. And, the non-interleaving data arrangement scheme is used as in [128, 96] where consecutive bits are placed within each device. This puts an entire word into a device and allows for element-wise operations

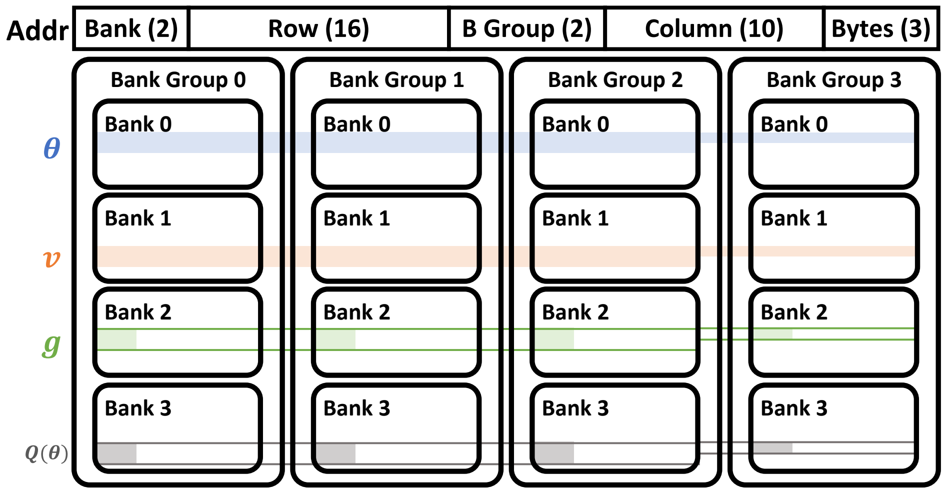


Figure 3.7: Address mapping and data placement scheme for GradPIM.

Some considerations are needed for the address mapping to avoid bank-group and bank conflicts. Except for the simple SGD optimizer, update phase requires more than one arrays per parameter (e.g., θ and v in Eq 3.3) for computation. Reading them are sequential, but incurs a cumbersome problem for GradPIM. When the two arrays of values are placed in two different bank groups, it requires an inter-bank communication, which is unsupported by GradPIM because it would occupy the global shared data bus within the DRAM chips. On the other hand, if the two arrays are placed within a single bank, it is a trivial case of a bank conflict. It would require alternating row activations for accessing each array.

Therefore, it is necessary to ensure that the corresponding elements of the arrays are placed within the same bank group, but different banks. It can be solved by carefully designing the address mapping. Figure 3.7 shows the address mapping for GradPIM. To enable maximum bank-group level parallelism, bank-group interleaving is adopted, so that multiple bank groups can operate concurrently. The bank ids within the bank groups are assigned to the MSB of the addresses. This makes sure that multiple different arrays can always be placed in distinct banks. When allocating the arrays such as θ or v, they are aligned to the bank boundary, so that the items at the matching positions always stay within the same bank group. In the case of the quantized weight parameters, it is impossible to perfectly align them with the non-quantized weight parameters, because their elements differ in width while the number of elements are the same. If they are aligned to the beginning of the array, they will not be present in the same bank group anymore, violating the GradPIM operation requirements. To solve the problem, it is chosen to utilize only the first quarter of the row for the quantized weights. By doing this, even though it wastes the DRAM capacity, it do not waste the off-chip bandwidth. The GradPIM operation will run without a problem because the column id of the elements do not have to match as long as they are placed within the same bank group and different bank. Multiple channels or multiple ranks are not considered in this section, but the channel or rank bits can be placed between the bank group bits and the bank bits as long as it does not violate the same bank group, different bank criteria. Additionally, bank-group bits are placed at the lowest position right after bytes so as to activate each GradPIM in bank-groups as soon as possible.

To exploit the effect of data mapping to the programming model, we assume that the API and device driver provide a device-side allocation function supporting separation between data structures, similar to multi-stream features in SSDs [129]. Therefore the user only specifies that each data structure should be stored to different banks, without being exposed to the indices of the physical banks or bank groups. In the current setting, we assume that the NPU has its dedicated memory attached with GradPIM. However, GradPIM can also be used when the host and the NPU share the memory by assigning certain memory region to the NPU, similar to pinned memory in CUDA [130].

3.5 Evaluation

3.5.1 Evaluation Methodology

The proposed NPU is implemented by an in-house simulator written in SystemC [131]. GradPIM has been modeled by extending DRAMSim 3.0 [132], and it is faithfully modeled that the timing of GradPIM operations as explained in subsection 3.3.3 while keeping the existing timings posed by the existing DDR protocol.

To verify the NPU, it has been synthesized in Verilog HDL at 1GHz using Nangate 45nm open cell library [133]. I have ensured the timing closure and the functionality then drew the power consumption of the NPU. The synthesis results for NPU components are presented in Table 3.2. I have used a DRAM based on DDR4-2133 with 4 ranks having 4 bank groups and 4 banks per bank group. and taken the timing parameters from [134]. The energy and timing parameters used in the paper are displayed in Table 3.3. To model the overhead of the GradPIM logic within the memory, it is conducted that a layout of the GradPIM unit over the DRAM constraint of 3 metal layers and core utilization of 70% using the 45nm process and scaled it to 32nm.

Table 3.2: NPU Synthesis Results

Module	Area (mm^2)	Power (W)
Datapath	19.89	14.33
Global Buffer	18.30	4.05
Local Buffers	3.08	0.71
Total	41.27	19.09

The area and energy measured is shown in Table 3.4. Compared to an x8 8Gb DDR4-SDRAM device, it consumes only 0.01% area overhead to the DRAM, corresponding to about 1Mb DRAM cells. I followed [135] to model the partial energy needed for read/write within the bank group (IDDpre). For the off-chip links, the DRAM power calculator from Micron [136] have been used.

The proposed design has been evaluated with two versions of ResNet (18- and 50- layers), Mobilenet [137], MLP [82] and AlphaGo Zero [81] to represent various types of DNN workloads. The results are shown in Figure 3.8 - 3.10. In the figure, the filled parts of the bar represent parameter update phase, and the empty parts of the bars represent the forward/backward phase. For the baseline, the proposed NPU, with 8 bit datapaths with dedicated 32 bit update units within the NPUs is used.

3.5.2 Experimental Results

Performance. Figure 3.8 (Top) shows the execution time of the chosen networks with minibatch size 32 (128 for MLP). The layers are grouped into a few blocks with similar characteristics for brevity. The leftmost bar shows the

Table 3.3: DRAM parameters

Spec.		DDR4-2133	
Timing (Cycles)	Value	Current (mA)	Value
tCK	0.94ns	Vdd	1.2V
tCL	16	IDD0	75
tRCD	16	IDD2P	25
tRP	16	IDD2N	33
tRAS	36	IDD3P	39
tCCD_L	6	IDD3N	44
tCCD_S	4	IDD4W	225
tPIM	5	IDD4R	225
tFAW	23	IDDpre	98

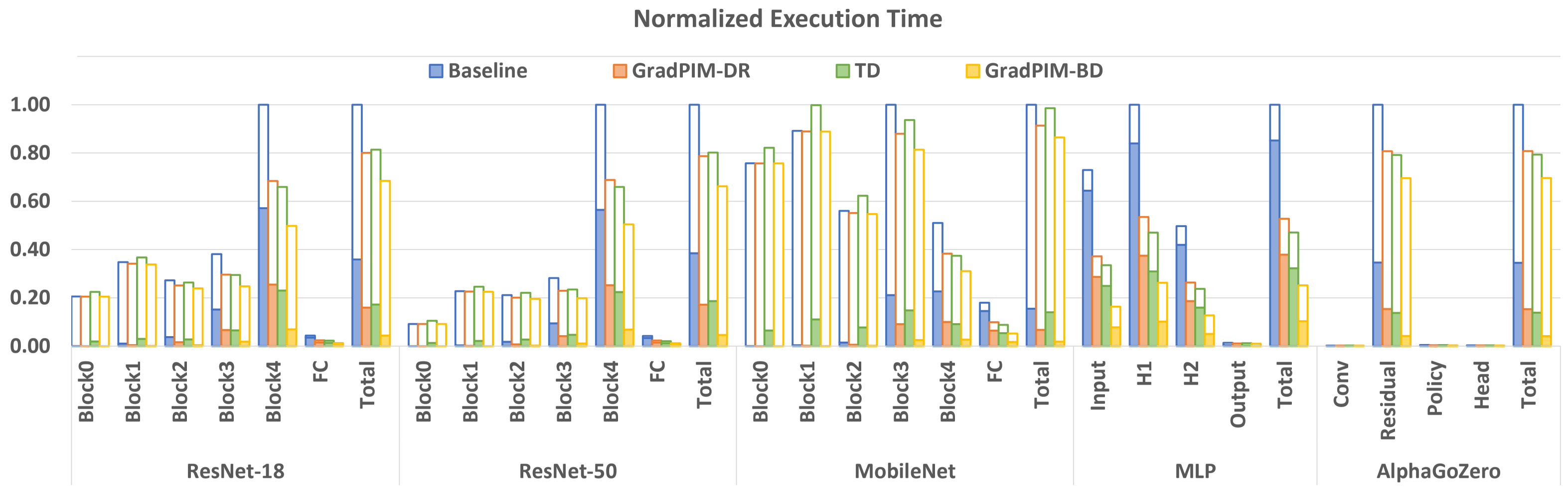


Figure 3.8: Normalized execution time of each layer on various networks using GradPIM. The filled parts of the bars represent parameter update phase, and the empty parts of the bars represent the forward/backward phase.

baseline, where the NPU has dedicated 32bit modules to execute the update phase including adders and quantize/dequantize units to convert between the precisions.

The second bar, noted ‘GradPIM-DR’ shows the GradPIM architecture directly controlled by the memory controller on the NPU. The third bar, ‘TD’ represents a design similar to TensorDIMM [102] where a buffer chip is added between the NPU and the memory, and the GradPIM units are placed within the buffer chips. The last bar ‘GradPIM-BD’ represents a design where GradPIM is placed on a buffered DIMM, and the commands to control the GradPIM units are sent from the buffer chip. This alleviates the command bus contention, and allows a fair comparison of GradPIM with TensorDIMM. The execution times are normalized to the baseline execution time of the most time-consuming block within each network, while the ‘total’ is separately normalized to the baseline execution time on the entire network.

Compared to the baseline, GradPIM-DR achieves around 2.25× higher performance for the parameter update phase from being able to utilize the bank group internal bandwidth of the DRAM. For the entire training, the overall speedup is about 24.8% in geometric mean as it is governed by the Amdahl’s law. with GradPIM-DR, the bottleneck is at the limited command rate

Table 3.4: GradPIM Layout Results

Module	Area (μm^2)	Power (mW)
Adder	320.1	0.058
Quantize	275.4	0.056
Dequantize	244.8	0.041
Scaler	606.1	0.159
Registers (×3)	206.7	0.04
Total	8267.8	1.74

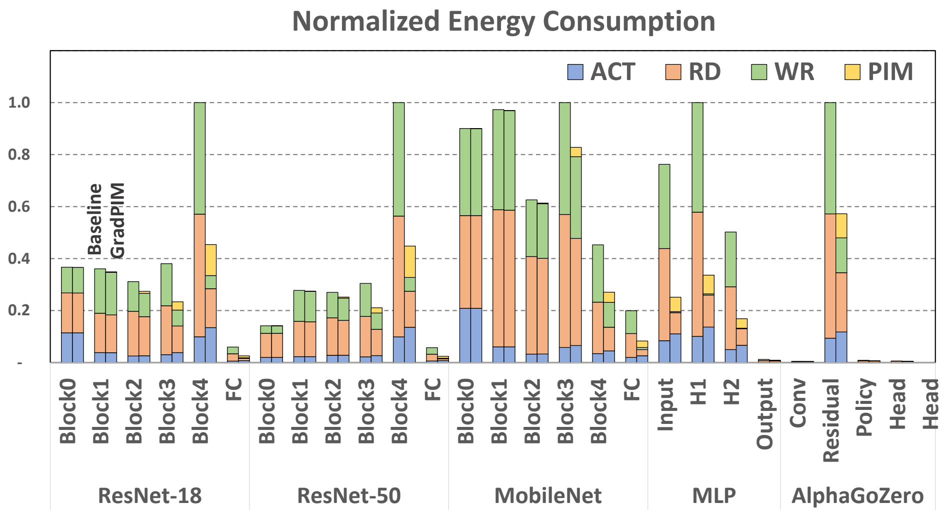


Figure 3.9: Energy consumption of various networks using GradPIM.

(Section 3.3.5). TD and GradPIM-BD alleviate the problem by using buffered DIMMs and sending the commands from the buffer devices. TD achieves 24.9% speedup in geometric mean. However, it's speedup is limited by the amount of rank-level parallelism. GradPIM-BD achieves 39.2% speedup overall, by being able to utilize all the bank group level parallelism.

Energy Consumption. The energy consumption from the memory is shown in Figure 3.9. The energy saving is almost proportional to the speedup as the saving is mostly from reduced external bandwidth usage, which translates to less data bus switching. As can be seen from the breakdown, most of the energy reduction is coming from the reduced amount of read/write. For the parameter update phase, the baseline NPU has to read the high precision master weight, perform the update, and write the master weight back to the memory, all occupying the off-chip bus. However, the data occupying the data bus with GradPIM is the low-precision gradient, one way only. Therefore, the proposed model benefits from two sources of savings: the lower bit width, and

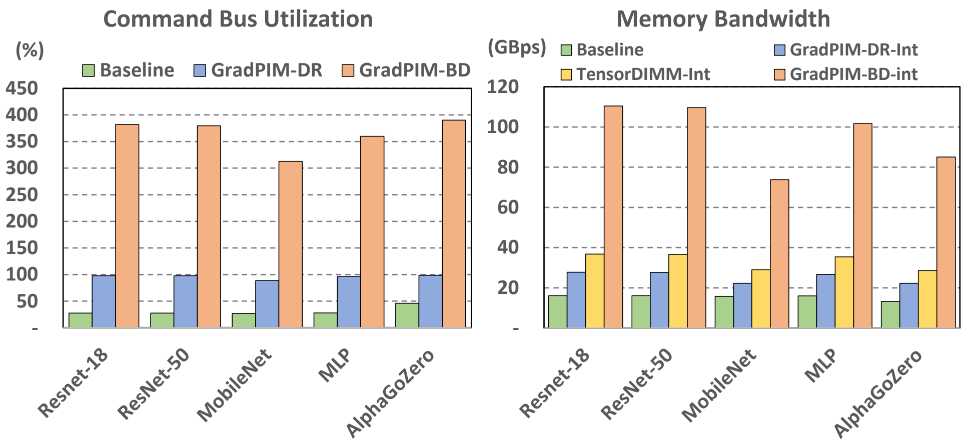


Figure 3.10: Command bus utilization (left) and the external/internal memory bandwidth consumption using GradPIM (right).

not having to read but only having to write the data. The activation consumes almost the same amount of energy between the baseline and GradPIM since GradPIM does not change the amount of data read from the DRAM array. In fact, the memory footprint is exactly the same, and the slight differences comes from change in access order. Please note that using GradPIM does not cause the number of activations to violate the tFAW constraint. Despite the increase in the internal reads, most of them result in row hits. Although not included in the paper, the experiments have been conducted to verify that relaxing tFAW to infinity incurs negligible difference in performance.

Bottleneck Analysis. Figure 3.10 (right) shows the external and internal bandwidth consumption of the DRAM during the update phase to get a deeper look. The baseline NPU's external bandwidth consumption during the update phase is around 15GBps, reaching near the theoretical maximum of 17.1GBps. The internal bandwidth of the baseline is trivially the same and omitted from the figure. On the other hand, the internal bandwidth consumption of GradPIM on average is 25GBps, far higher than that of the baseline. This reasons

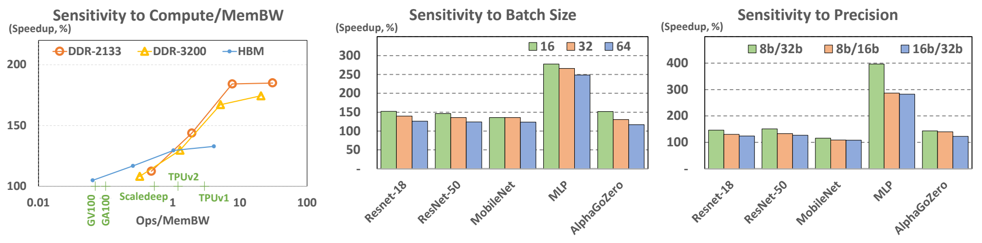


Figure 3.11: Sensitivity to compute-bandwidth ratio (left), minibatch size (mid), and minibatch size (right).

from each GradPIM unit working independently on each bank group, and this is the source of speedup and the energy consumption. We found that the bottleneck is at the command bus reaching near 100% as shown in Figure 3.10 (left), which shows that the command bus utilization is near 100% with GradPIM-DR for all networks, blocking any further internal bandwidth increase. With GradPIM-BD, the commands are generated from the buffer chip and thus it alleviates command bus bottleneck. It consumes around 95GBps, more than 3.5× the internal bandwidth compared to GradPIM-DR. However, Figure 3.10 (left) shows that its command bus utilization is still almost at the maximum, blocking any further internal bandwidth usage.

3.5.3 Sensitivity Analysis

Compute/bandwidth ratio. Figure 3.11 (left) shows the speedup sensitivity with respect to the ratio between the ops provided by the MAC array and the DRAM bandwidth, measured from multiple MAC array size (64x64-512x512) and multiple DDR4 data rate (DDR4 2133-3200 and HBM). The speedups are measured from AlphaGoZero. The X axis shows the ratio between the ops and the memory bandwidth. The ratios of a few NPUs [55, 138, 139, 140, 141] have been marked. For a fair comparison, I have used the metric of operations per second (compute) divided by activation item per second (memory bandwidth) to take different target precisions into account.

As Ops/BW gets higher, GradPIM achieves more speedup due to the increased dependency on the effective memory bandwidth. The figure shows that GradPIM achieves meaningful speedups (20-80%) for a range that covers that of the chosen NPUs, but the speedup diminishes as the ratio approaches that of the GPUs (<10%).

Minibatch Size. While the minibatch size does not affect the speedup of GradPIM over the update phase, it directly affects the portion of the update phase in the entire training. Figure 3.11 (mid) shows the change in the overall speedup coming from the batch size. While the overall speedup relative to the baseline at the same minibatch size does not change by a large amount, it shows a continuous trend where the smaller batch size leads to more speedup. Thus GradPIM has better potential for speedup with smaller batchsizes. As seen in Section 3.5.5, this will help improve the scaling efficiency for distributed deep learning.

Sensitivity to mixed precision levels. Figure 3.11 (right) shows the different speedups when the 8/16 or 16/32 mixed precision is used instead of 8/32 as in the default setup. While the speedup of GradPIM highly depends on the ratio between the low-precision and the high-precision representations, the setting of 8/16 bit and 16/32 bit system still provides a meaningful amount of speedup of 30.0% and 33.3%, respectively. 8bit training is still not at the mature status, and 16/32 bit mixed precision training is still what's dominant on the field. However, I envision that as the technology advances, lower precision would be utilized more often in favor of GradPIM.

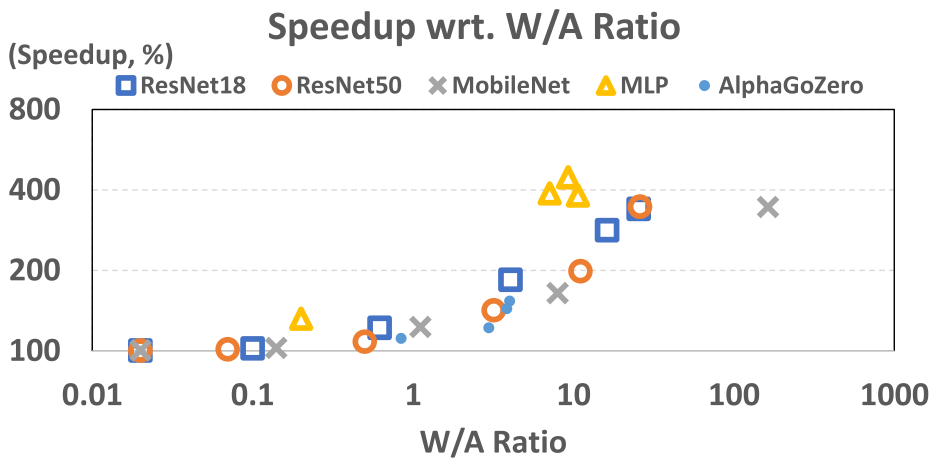


Figure 3.12: Layer characterizations.

3.5.4 Layer Characterizations.

To study the relation between the layer characterizations and the speedups obtained by GradPIM, I have plotted the speedups according to the weight/activation ratio on Figure 3.12 with both the X/Y axis in log scale. The plot shows a clear correlation between the weight/activation ratio and the speedup. Usually, for the convolutional layers from the earlier stages of the networks, there are large activation fields, with relatively small filters and the speedup for them are generally small. For the layers from the later stages of the networks, the activation maps get smaller due to the result of repetitive pooling and the striding convolutions. These layers and FC layers often exhibit a very high weight/activation ratio, and the speedup gets larger.

3.5.5 Distributed Data Parallelism

One popular way to improve the training performance is through decentralized distributed data parallelism [142, 143], which allows using multiple nodes to run the exact same model which different portions of the minibatch. This

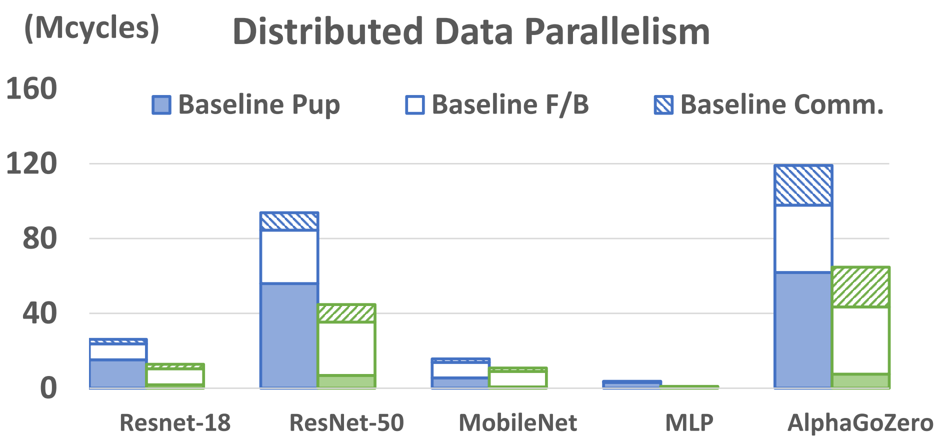


Figure 3.13: Projections to distributed training.

provides another opportunity for GradPIM. By applying distributed data parallelism it parallelizes the forward and backward pass of the training, but the parameter update phase is performed independently at each NPU, which is almost equivalent to the sequential portion of the application. Also, the all-reduce [144] communication pattern required in the distributed data parallelism for the gradient sharing includes another set of element-wise operations that can be mapped to GradPIM.

To find out the potential for GradPIM to work for distributed data parallel distributed learning, I projected the experimental results to see the performance gain over the same network, with a modest number of 4 nodes, each taking 8 inputs from 32 inputs in the minibatch. I have assumed the nodes are connected in a torus-like network with 100Gbps connections [69] with HW supporting NI as in [145]. In Figure 3.13, the communication time is depicted as Comm. for the baseline and GradPIM in the legend. Due to the smaller effective batch size per node, GradPIM shows much better scalability compared to the baseline, and the performance is ALMOST 2× better than the baseline with distributed training.

3.6 Summary

This work have proposed GradPIM, a processing-in-memory design based on the extension of the DDR4 protocol along with an NPU designed to work with GradPIM for mixed-precision training. This work have demonstrated that GradPIM can speed up the update phase of the DNN training up to around 40%, leading to overall 20% performance gain according to the setup. GradPIM poses only a negligible overhead to the DRAM and can be fully controlled by extending the DDR memory controllers. Even though the proposed design is based on DDR4 SDRAM, it is believed that the same design can be adapted to other memories such as HBM, HMC or GDDR. It is expected to show similar speedups or improvement if more bankgroup numbers is exploited in advanced memory technologies toward high-performance NPUs.

3.6.1 Discussion

Supporting Other Kinds of Parameter Update Algorithms: This work have demonstrated that GradPIM works on SGD with momentum and a weight decay term. Some algorithms such as NAG [76] can be supported with GradPIM naturally in the same way demonstrated in this work. However, there are other kinds of parameter update algorithms used in practice with more complexity. For example, Adagrad [75] and RMSProp [77] use a decaying average of the square gradients, and Adam [74] takes second order momentum into account. These extra values require access to adjacent rows in a bank group concurrently. Since momentum SGD utilizes only 3 banks, there is room for another bank to join the computation for that extra row. In an unlikely rare case where the number of rows to be opened exceeds the number of banks per bank group (four in our setting), the computation can be split into multiple passes, so that three rows are accessed and the intermediate values are stored in another row for the next pass. It would require activating and reading the data multiple times, while causing only small overhead on the overall performance.

Using per-bank unit instead of per bank group: As it is described in Section 3.3.1, a GradPIM unit is placed per each bank group to allow parallel operations with multiple rows. Another option that can be considered is to place a unit per bank. Since it is not allowed to have multiple rows open at the same time, the multiple series of data has to be placed in an array-of-structures, so that they can be accessed in a single row. While it would give PIM to process them with high bandwidth, the NPU will have trouble writing the gradient into the memory. Consequently, the overhead from writing the gradient with certain stride would outweigh the benefits of using PIM.

Learning Rate Scheduling: One could raise question about how to apply learning rate scheduling, as a fixed learning rate is assumed in this work. GradPIM can be extended to support varying learning rate with slightly more logic added. Scaling the values each time by 2 can be easily implemented using a shifter. For more complicated scheduling such as cosine [146] or polynomial decay [147], it may be chosen to approximate the decaying function as computing the exact value of them is expensive. Another way would be to utilize the mode register and let the NPU provide the new learning rate value, at the expense of some performance overhead.

Bibliography

[1] S Novak, C Parker, D Becher, M Liu, M Agostinelli, M Chahal, P Packan, P Nayak, S Ramey, and S Natarajan. Transistor aging and reliability in 14nm tri-gate technology. In *2015 IEEE International Reliability Physics Symposium*, pages 2F–2. IEEE, 2015.

[2] John Keane and Chris H Kim. Transistor aging. *IEEE Spectrum*, 48(5):28–33, 2011.

[3] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. Impact: imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design (ISLPED)*, pages 409–414. IEEE Press, 2011.

[4] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2013.

[5] Hussam Amrouch, Behnam Khaleghi, Andreas Gerstlauer, and Jörg Henkel. Reliability-aware design to suppress aging. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[6] Behzad Boroujerdian, Hussam Amrouch, Jörg Henkel, and Andreas Gerstlauer. Trading off temperature guardbands via adaptive approximations. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 202–209. IEEE, 2018.

[7] Jongho Kim, Kiyoung Choi, Yonghwan Kim, Wook Kim, Kyungtae Do, and Jungyun Choi. Delay monitoring system with multiple generic monitors for wide voltage range operation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(1):37–49, 2018.

[8] Hussam Amrouch, Victor M van Santen, Thomas Ebi, Volker Wenzel, and Jörg Henkel. Towards interdependencies of aging mechanisms. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pages 478–485. IEEE Press, 2014.

[9] C Prasad, KW Park, M Chahal, I Meric, SR Novak, S Ramey, P Bai, H-Y Chang, NL Dias, WM Hafez, et al. Transistor reliability characterization and comparisons for a 14 nm tri-gate technology optimized for system-on-chip and foundry platforms. In *2016 IEEE International Reliability Physics Symposium (IRPS)*, pages 4B–5. IEEE, 2016.

[10] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel. Towards aging-induced approximations. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[11] Swaroop Ghosh and Kaushik Roy. Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proceedings of the IEEE*, 98(10):1718–1751, 2010.

[12] V Huard, F Cacho, A Benhassain, and C Parthasarathy. Aging-aware

adaptive voltage scaling of product blocks in 28nm nodes. In *2016 IEEE International Reliability Physics Symposium (IRPS)*, pages 7C–2. IEEE, 2016.

[13] Hassan Mostafa, Mohab Anis, and Mohamed Elmasry. Nbti and process variations compensation circuits using adaptive body bias. *IEEE transactions on semiconductor manufacturing*, 25(3):460–467, 2012.

[14] Minki Cho, Stephen T Kim, Carlos Tokunaga, Charles Augustine, Jaydeep P Kulkarni, Krishnan Ravichandran, James W Tschanz, Muhammad M Khellah, and Vivek De. Postsilicon voltage guard-band reduction in a 22 nm graphics execution core using adaptive voltage scaling and dynamic power gating. *IEEE Journal of Solid-State Circuits*, 52(1):50–63, 2016.

[15] Jiangyi Li and Mingoo Seok. Robust and in-situ self-testing technique for monitoring device aging effects in pipeline circuits. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. ACM, 2014.

[16] Ning Zhu, Wang Ling Goh, Weija Zhang, Kiat Seng Yeo, and Zhi Hui Kong. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. *IEEE transactions on very large scale integration (VLSI) systems*, 18(8):1225–1229, 2009.

[17] Hamid Reza Mahdiani, Ali Ahmadi, Sied Mehdi Fakhraie, and Caro Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 57(4):850–862, 2009.

[18] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. A low latency generic accuracy configurable adder. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[19] Jin Miao, Ku He, Andreas Gerstlauer, and Michael Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *Proceedings of the International Conference on Computer-Aided Design*, pages 728–735. ACM, 2012.

[20] Honglan Jiang, Cong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(4):60, 2017.

[21] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 820–825. ACM, 2012.

[22] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. On reconfiguration-oriented approximate adder design and its application. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–54. IEEE, 2013.

[23] Bert Moons and Marian Verhelst. Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 237–242. IEEE, 2015.

[24] Daniele Jahier Pagliari and Massimo Poncino. Application-driven synthesis of energy-efficient reconfigurable-precision operators. In *2018*

IEEE International Symposium on Circuits and Systems (ISCAS), pages 1–5. IEEE, 2018.

[25] J. Wang, S. Kuang, and Y. Chuang. Design of reconfigurable low-power pipelined array multiplier. In *2006 International Conference on Communications, Circuits and Systems*, volume 4, pages 2277–2281, June 2006.

[26] M. de la Guia Solaz, W. Han, and R. Conway. A flexible low power dsp with a programmable truncated multiplier. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 59(11):2555–2568, Nov 2012.

[27] Charles R Baugh and Bruce A Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on computers*, 100(12):1045–1047, 1973.

[28] I. S. Abu-Khater, A. Bellaouar, and M. I. Elmasry. Circuit techniques for cmos low-power high-performance multipliers. *IEEE Journal of Solid-State Circuits*, 31(10):1535–1546, Oct 1996.

[29] K. Bhardwaj, P. S. Mane, and J. Henkel. Power- and area-efficient approximate wallace tree multiplier for error-resilient systems. In *Fifteenth International Symposium on Quality Electronic Design*, pages 263–269, March 2014.

[30] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[31] Christopher S Wallace. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, 0(1):14–17, 1964.

[32] Lionel Vincent, Philippe Maurine, Edith Beigné, Suzanne Lesecq, and Julien Mottin. Temperature and fast voltage on-chip monitoring using low-cost digital sensors. In *VARI: Workshop on CMOS Variability*, 2013.

[33] Hussam Amrouch, Seyed Borna Ehsani, Andreas Gerstlauer, and Jorg Henkel. On the efficiency of voltage overscaling under temperature and aging effects. *IEEE Transactions on Computers*, 2019.

[34] I-Ming Pao and Ming-Ting Sun. Modeling dct coefficients for fast video encoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(4):608–616, 1999.

[35] Patrick Seeling and Martin Reisslein. Video transport evaluation with h. 264 video traces. *IEEE Communications Surveys & Tutorials*, 14(4):1142–1165, 2011.

[36] Hussam Amrouch, Subrat Mishra, Victor van Santen, Souvik Mahapatra, and Jörg Henkel. Impact of bti on dynamic and static power: From the physical to circuit level. In *2017 IEEE International Reliability Physics Symposium (IRPS)*, pages CR–3. IEEE, 2017.

[37] Nvidia Corp. Whitepaper gpu-based deep learning inference : A performance and power analysis. Technical report, Nvidia Corp., 2015.

[38] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the In-*

ternational Conference on Architectural Support for Programming Languages and Operating Systems, pages 269–284, 2014.

[39] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[40] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. Training deep neural networks with low precision multiplications. In *Workshop Contribution at International Conference on Learning Representations*, 2015.

[41] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *Proceedings of the Design Automation Conference*, pages 124:1–124:6, 2016.

[42] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded Computing Systems*, 12:92, 2013.

[43] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2015.

[44] Peter U Diehl, Bruno U Pedroni, Andrew Cassidy, Paul Merolla, Emre Neftci, and Guido Zarrella. Truehappiness: Neuromorphic emotion recognition on TrueNorth. *arXiv preprint arXiv:1601.04183*, 2016.

[45] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[46] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015.

[47] Rangharajan Venkatesan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. Spintastic: Spin-based stochastic logic for energy-efficient computing. In *Proceedings of the Design, Automation & Test in Europe Conference*, pages 1575–1578, 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[49] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[50] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[51] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, 2016.

[52] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ISCA*, 2015.

[53] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ASPLOS*, 2014.

[54] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *MICRO*, 2014.

[55] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[56] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ISCA*, 2017.

[57] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *ISCA*, 2010.

[58] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *CVPR*, 2014.

[59] Sangkug Lym, Armand Behroozi, Wei Wen, Ge Li, Yongkee Kwon, and Mattan Erez. Mini-batch serialization: CNN training with inter-layer data reuse. *SysML*, 2019.

[60] Wonkyung Jung, Daejin Jung, Sunjung Lee, Wonjong Rhee, Jung Ho Ahn, et al. Restructuring batch normalization to accelerate CNN training. *SysML*, 2019.

[61] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer CNN accelerators. In *MICRO*, 2016.

[62] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *NeurIPS*, 2018.

[63] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. In *NeurIPS*, 2019.

[64] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. Mixed precision training with 8-bit floating point. *arXiv preprint*, 2019.

[65] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, et al. Mixed precision training of convolutional neural networks using integer operations. In *ICLR*, 2018.

[66] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *ICLR*, 2018.

[67] Tensor Cores in NVIDIA Volta Architecture | NVIDIA.

[68] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *NeurIPS*, 2018.

[69] Hiroaki Mikami, Hisahiro Suganuma, et al. Imagenet/resnet-50 training in 224 seconds. *arXiv preprint*, 2018.

[70] Nicholas J Higham. The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14(4):783–799, 1993.

[71] Xi Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. Fxpnet: Training a deep convolutional neural network in fixed-point representation. In *IJCNN*, 2017.

[72] DDR4 SDRAM specification.

[73] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[74] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[75] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[76] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). In *Doklady an ussr*, volume 269, pages 543–547, 1983.

[77] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning.

[78] DDR3 SDRAM specification.

[79] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, et al. Google workloads for consumer devices: Mitigating data movement bottlenecks. In *ASPLOS*, 2018.

[80] Tensorflow: Mobile.

[81] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[82] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[83] Peter M Kogge. EXECUBE-A new architecture for scaleable MPPs. In *ICPP*, 1994.

[84] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, 1997.

[85] Yi Kang, Wei Huang, Seung-Moon Yoo, D Keen, Zhenzhou Ge, V Lam,

P Pattnaik, and J Torrellas. FlexRAM: toward an advanced intelligent memory system. In *ICCD*, 1999.

[86] Ken Mai, T Paaske, N Jayasena, R Ho, WJ Dally, and M Horowitz. Smart Memories: a modular reconfigurable architecture. In *ISCA*, 2000.

[87] Graham Kirsch. Active Memory: Micron' s Yukon. In *IPDPS*, 2003.

[88] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA*, 2015.

[89] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In *ISCA*, 2015.

[90] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. GraphP: Reducing communication for pim-based graph processing with efficient data partition. In *HPCA*, 2018.

[91] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. GraphQ: Scalable pim-based graph processing. In *MICRO*, 2019.

[92] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. In *HPDC*, 2014.

[93] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, A Buyuktosunoglu, A Davis, and F Li. NDC:

Analyzing the Impact of 3D-Stacked Memory+ Logic Devices on MapReduce Workloads. In *ISPASS*, 2014.

[94] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al. Rowclone: fast and energy-efficient in-dram bulk data copy and initialization. In *MICRO*, 2013.

[95] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *MICRO*, 2017.

[96] Jinho Lee, Jung Ho Ahn, and Kiyoung Choi. Buffered compares: Excavating the hidden parallelism inside dram architectures with lightweight logic. In *DATE*, 2016.

[97] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. ComputeDRAM: In-memory compute using off-the-shelf drams. In *MICRO*, 2019.

[98] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Gather-scatter DRAM: In-dram address translation to improve the spatial locality of non-unit strided accesses. In *MICRO*, 2015.

[99] Shaahin Angizi and Deliang Fan. ReDRAM: A reconfigurable processing-in-DRAM platform for accelerating bulk bit-wise operations. In *ICCAD*, 2019.

[100] Fabrice Devaux. The true processing in memory accelerator. In *HCS*, 2019.

[101] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. *ISCA*, 2016.

[102] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *MICRO*, 2019.

[103] Lei Jiang, Minje Kim, Wujie Wen, and Danghui Wang. XNOR-POP: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 DRAMs. In *ISLPED*, 2017.

[104] Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, and Jun Yang. DrAcc: a dram based accelerator for accurate cnn inference. In *DAC*, 2018.

[105] Chirag Sudarshan, Jan Lappas, Muhammad Mohsin Ghaffar, Vladimir Rybalkin, Christian Weis, Matthias Jung, and Norbert Wehn. An in-DRAM neural network processing engine. In *ISCAS*, 2019.

[106] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *MICRO*, 2017.

[107] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. McDRAM: Low latency and energy-efficient matrix computations in DRAM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2613–2622, 2018.

[108] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish

Narayanasamy, David Blaauw, and Reetuparna Das. Compute caches. In *HPCA*, pages 481–492, 2017.

[109] Mingu Kang, Min-Sun Keel, Naresh R Shanbhag, Sean Eilert, and Ken Curewitz. An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM. In *ICASSP*, pages 8326–8330, 2014.

[110] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*, 2020.

[111] Amogh Agrawal, Akhilesh Jaiswal, Deboleena Roy, Bing Han, Gopalakrishnan Srinivasan, Aayush Ankit, and Kaushik Roy. Xcel-RAM: Accelerating binary neural networks in high-throughput SRAM compute arrays. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(8):3064–3076, 2019.

[112] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G Friedman. AC-DIMM: associative computing with stt-mram. In *ISCA*, 2013.

[113] Shaahin Angizi, Zhezhi He, and Deliang Fan. ParaPIM: a parallel processing-in-memory accelerator for binary-weight deep neural networks. In *ASPDAC*, 2019.

[114] Shaahin Angizi, Zhezhi He, and Deliang Fan. Dima: a depthwise CNN in-memory accelerator. In *ICCAD*, 2018.

[115] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *DAC*, 2016.

[116] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ISCA*, 2016.

[117] Janusz A Starzyk et al. Memristor crossbar architecture for synchronous neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(8):2390–2401, 2014.

[118] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ISCA*, 2016.

[119] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. Floatpim: In-memory acceleration of deep neural network training with high precision. In *ISCA*, pages 802–815, 2019.

[120] Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, and Lee-Sup Kim. Bank-group level parallelism. *IEEE Transactions on Computers*, 66(8):1428–1434, 2017.

[121] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*, 2018.

[122] Peng Guo, Hong Ma, Ruizhi Chen, Pin Li, Shaolin Xie, and Donglin Wang. FBNA: A fully binarized neural network accelerator. In *FPL*, 2018.

[123] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In *FPT*, 2016.

[124] Li Jiao, Cheng Luo, Wei Cao, Xuegong Zhou, and Lingli Wang. Accelerating low bit-width convolutional neural networks with embedded fpga. In *FPL*, 2017.

[125] Intel® AVX-512 instructions.

[126] DDR5 full spec draft rev0.1.

[127] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint*, 2014.

[128] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *HPCA*, 2015.

[129] Changho Choi et al. Multi-stream write SSD. *Flash Memory Summit*, 2016.

[130] CUDA Nvidia. Nvidia cuda c programming guide. *Nvidia Corporation*, 120(18):8, 2011.

[131] Preeti Ranjan Panda. SystemC: a modeling platform supporting multiple design abstractions. In *ISSS*, 2001.

[132] Shang Li, Zhiyuan Yang, Dhriaj Reddy, Ankur Srivastava, and Bruce

Jacob. DRAMsim3: a cycle-accurate, thermal-capable DRAM simulator. *Computer Architecture Letters*, 2020.

[133] Jesper Knudsen. Nangate 45nm open cell library. *CDNLive, EMEA*, 2008.

[134] 8Gb: x4, x8, x16 DDR4 SDRAM.

[135] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems. In *MICRO*. IEEE, 2017.

[136] DDR4 SDRAM system-power calculator.

[137] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[138] Cloud tensor processing units (tpus).

[139] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *ISCA*, pages 13–26, 2017.

[140] Data sheet: Quadro gv100.

[141] Nvidia ampere architecture in-depth.

[142] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[143] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*, 2016.

[144] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[145] Youjie Li, Jongse Park, Mohammad Alian, Yifan Yuan, Zheng Qu, Peitian Pan, Ren Wang, Alexander Schwing, Hadi Esmaeilzadeh, and Nam Sung Kim. A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In *MICRO*, 2018.

[146] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint*, 2016.

[147] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

요약

근사 컴퓨팅은 연산의 정확도의 손실을 어플리케이션 별 적절한 수준까지 허용함으로써 연산에 필요한 비용 (에너지나 지연시간)을 줄인다. 게다가, 근사 컴퓨팅은 컴퓨팅 시스템 설계의 회로 계층부터 어플리케이션 계층까지 다양한 계층에 적용될 수 있다. 본 논문에서는 근사 컴퓨팅 방법론을 다양한 시스템 설계의 계층에 적용하여 전력과 에너지 측면에서 이득을 얻을 수 있는 방법들을 제안하였다. 이는, 연산 근사화 (computation Approximation)를 통해 회로의 노화로 인해 증가된 지연시간을 추가적인 전력소모 없이 보상하는 방법과 (챕터 1), 근사 뉴런모델 (approximate neuron model)을 이용해 에너지 효율이 높은 신경망을 구성하는 방법 (챕터 2), 그리고 메모리 대역폭으로 인한 병목현상 문제를 높은 정확도 데이터를 활용한 연산을 메모리 내에서 수행함으로써 완화시키는 방법을 (챕터 3) 제안하였다.

첫 번째 챕터는 회로의 노화로 인한 지연시간위반을 (timing violation) 설계 마진이나 (reliability guardband) 공급전력의 증가 없이 연산오차 (computation approximation error)를 통해 보상하는 설계방법론 (design methodology)를 제안하였다. 이를 위해 주요경로의 (critical path) 지연시간을 동작시간에 정확하게 측정할 필요가 있다. 여기서 제안하는 방법론은 RTL component와 system 단계에서 평가되었다. RTL component 단계의 실험결과를 통해 제안한 방식이 표준화된 평균제곱오차를 (normalized mean squared error) 상당히 줄였음을 볼 수 있다. 그리고 system 단계에서는 이미지처리 시스템에서 이미지의 품질이 인지적으로 충분히 회복되는 것을 보임으로써 회로노화로 인해 발생한 지연시간위반 오차가 에러의 크기가 작은 연산오차로 변경되는 것을 확인 할 수 있었다. 결론적으로, 제안된 방법론을 따랐을 때 0.8%의 공간을 (area) 더 사용하는 비용을 지불하고 21.45%d의 동적전력소모와 (dynamic power consumption) 10.78%의 정적전력소모의 (static power consumption) 감소를 달성할 수 있었다.

두 번째 챕터는 근사 뉴런모델을 활용하는 고-에너지효율의 신경망을 (neural

network) 제안하였다. 본 논문에서 사용한 두 가지의 근사 뉴런모델은 확률컴퓨팅과 (stochastic computing) 스파이킹뉴런 (spiking neuron) 이론들을 기반으로 모델링되었다. 확률컴퓨팅은 산술연산들을 확률적으로 수행함으로써 이진연산을 낮은 전력소모로 수행한다. 최근에 확률컴퓨팅 뉴런모델을 이용하여 심층 신경망 (deep neural network)를 구현할 수 있다는 연구가 진행되었다. 그러나, 확률컴퓨팅을 뉴런모델링에 활용할 경우 심층신경망이 매 클락사이클마다 (clock cycle) 하나의 비트만을 (bit) 처리하므로, 지연시간 측면에서 매우 나쁠 수 밖에 없는 문제가 있다. 따라서 본 논문에서는 이러한 문제를 해결하기 위하여 스파이킹 뉴런모델로 구성된 스파이킹 심층신경망을 확률컴퓨팅을 활용한 심층신경망 구조와 결합하였다. 스파이킹 뉴런모델의 경우 매 클락사이클마다 여러 비트를 처리할 수 있으므로 심층신경망의 입력 인터페이스로 사용될 경우 지연시간을 줄일 수 있다. 하지만, 확률컴퓨팅 뉴런모델과 스파이킹 뉴런모델의 경우 부호화 (encoding) 방식이 다른 문제가 있다. 따라서 본 논문에서는 해당 부호화 불일치 문제를 모델의 파라미터를 학습할 때 고려함으로써, 파라미터들의 값이 부호화 불일치를 고려하여 조절 (calibration) 될 수 있도록 하여 문제를 해결하였다. 이러한 분석의 결과로, 앞 쪽에는 스파이킹 심층신경망을 배치하고 뒷 쪽애는 확률컴퓨팅 심층신경망을 배치하는 혼성신경망을 제안하였다. 혼성신경망은 스파이킹 심층신경망을 통해 매 클락사이클마다 처리되는 비트 양의 증가로 인한 지연시간 감소 효과와 확률컴퓨팅 심층신경망의 저전력 소모 특성을 모두 활용함으로써 각 심층신경망을 따로 사용하는 경우 대비 우수한 에너지 효율성을 비슷하거나 더 나은 정확도 결과를 내면서 달성한다.

세 번째 챕터는 심층신경망을 8비트 부동소숫점 연산으로 학습하는 신경망 처리유닛의 (neural processing unit) 파라미터 갱신을 (parameter update) 메모리-내-연산으로 (in-memory processing) 가속하는 GradPIM 아키텍쳐를 제안하였다. GradPIM은 8비트의 낮은 정확도 연산은 신경망처리유닛에 남기고, 높은 정확도를 가지는 데이터를 활용하는 연산은 (파라미터 갱신) 메모리 내부에 둠

으로써 신경망처리유닛과 메모리간의 데이터통신의 양을 줄여, 높은 연산효율과 전력효율을 달성하였다. 또한, GradPIM은 bank-group 수준의 병렬화를 이루어내 높은 내부 대역폭을 활용함으로써 메모리 대역폭을 크게 확장시킬 수 있게 되었다. 또한 이러한 메모리 구조의 변경이 최소화되었기 때문에 추가적인 하드웨어 비용도 최소화되었다. 실험 결과를 통해 GradPIM이 최소한의 DRAM 프로토콜 변화와 DRAM칩 내의 공간사용을 통해 심층신경망 학습과정 중 파라미터 갱신에 필요한 시간을 40%만큼 향상시켰음을 보였다.

주요어: 근사컴퓨팅, 회로노화, 확률컴퓨팅, 스파이킹-신경망, 메모리-내-연산, 혼성-정확도 학습, 신경망처리유닛

학번: 2015-20917