S^{3}M: Static Semi-Segmented Multipliers for Energy-efficient DNN Inference Accelerators

Mingtao Zhang*‡, Quan Cheng*, Hiromitsu Awano*, Longyang Lin†, Masanori Hashimoto*‡
* Graduate School of Informatics, Kyoto University, Japan
† School of Microelectronics, Southern University of Science and Technology, China
‡ Email: zhang.mingtao.22h@st.kyoto-u.ac.jp, hashimoto@i.kyoto-u.ac.jp

2024 IEEE 42nd International Conference on Computer Design (ICCD) | 979-8-3503-8040-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCD63220.2024.00014

***Abstract*—Approximate multipliers offer an efficient approach to reduce power consumption in compute-intensive applications, such as Deep Neural Networks (DNNs). However, current 8-bit approximate multipliers struggle to maintain high accuracy across various DNN applications. In this paper, we highlight challenges in 8-bit multiplier designs with body approximation strategies and evaluate the effectiveness of input approximation methods. Recognizing that exact multipliers with quantization bit-widths below 8 bits have demonstrated superior performance, we aim to explore whether alternative input approximation methods can provide an even better trade-off between accuracy and energy consumption. To this end, by exploiting the fact that weight operand values are smaller than activations and prepared offline in DNNs, we simplify a static segmented multiplier (SSM) into a static semi-segmented multiplier (S^{3}M), achieving a 31.58% reduction in power-delay product (PDP) compared to the original SSM, with similar classification accuracy. Additionally, we propose Coded S^{3}M with optimized memory usage and implement various multipliers on a systolic array-based accelerator. Experimental results show that the proposed S^{3}M and Coded S^{3}M outperform existing 8-bit approximate multipliers in DNN applications, effectively bridging the PDP and inference accuracy trade-off observed across exact commercial IP multipliers of varied bit-widths without requiring time-consuming retraining. Consequently, the proposed multiplier designs provide enhanced computational solutions for energy-efficient DNN inference accelerators.**

***Index Terms*—Approximate computing, multiplier, low-power design, neural network**

I. Introduction

As DNNs increase in complexity, the heightened power consumption on edge devices necessitates a thorough exploration of low-power techniques. Given the inherent error resilience of DNN applications, reducing the bit-width of operands, as one of the techniques in approximate computing, is widely adopted for energy reduction as it decreases not only computational energy but also memory usage. With the success of 8-bit quantization in tensor processing units (TPU) [1], 8-bit precision has become the most popular choice for recent approximation works [2]. Given that multiply-accumulate (MAC) units account for approximately 90% of computations in CNNs [3], and considering the more complex structure and higher power consumption of integer multipliers compared to adders, numerous designs for 8-bit approximate multipliers have been proposed to further reduce power consumption [4].

Among various approximate multiplier design strategies [4], one popular approach for 8-bit width involves modifying the multiplier itself to reduce logic gates, named *body approximation* in this paper. Representative methods include replacing exact compressors with approximate compressors in the Dadda multiplier structure [5] [6] [7], introducing the approximate Booth encoding [8] [9] [10], and exploring an approximate multiplier with two-input gates such as AND and OR through a genetic programming algorithm [11] [12] [13].

Despite the existence of numerous 8-bit approximate multipliers with body approximation, none of them has proven to be practical and reliable across various DNN applications. Some works employ retraining to enhance the inference accuracy, but this can be prohibitively time-consuming, as GPUs cannot directly cope with approximate components [14]. Others combine approximate multipliers with an exact multiplier in a reconfigurable design to maintain application accuracy [15]. However, this approach results in an increase in both power and area due to the additional control logic and the exact multiplier, whereas such overhead is not emphasized.

On the other hand, an alternative approach is approximation at the input, represented by two types of input truncation schemes: dynamic segmented multiplier (DSM) [16] [17] and static segmented multiplier (SSM) [18] [19]. While DSM is more suitable for large bit-width multipliers, SSM could be a practical design strategy for 8-bit approximate multipliers as it does not require complex leading-one detectors. SSM extracts m contiguous bits from two n-bit operands by checking the $n-m$ most significant bits (MSB), where m is statically fixed in SSMs. Subsequently, the segmentation information of the two operands is incorporated into the shift information, expanding the inner $m \times m$ multiplication into the final $2n$-bit result. However, SSM has not been fully recognized in the DNN domain.

This paper investigates the challenges in designing effective 8-bit approximate multipliers in DNN inference accelerators and analyzes the strengths of the SSM. To adapt the SSM for DNN applications where one operand (weights) is prepared offline, we simplified its structure, enhancing hardware efficiency to rival the performance of exact DesignWare IP multipliers with various bit-widths. This demonstrates its practicality in real-world scenarios where many previous designs

2576-6996/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCD63220.2024.00014

failed to compete with commercial IPs. Moreover, unlike many prior designs that focus solely on the multiplier level, we implement multiple multipliers into a systolic array with memory bit-width aligned with the bit-width of activations and weights to comprehensively evaluate multiplier hardware performance at the accelerator level. Additionally, we encode the segmentation information into weight operands, further improving performance at the accelerator level. The major contributions of this work include:

- Analyzing the challenges and limitations of the 8-bit approximate multiplier in body approximation.
- Simplifying SSM to static semi-segmented multipliers (S^3M) that segment only weight inputs, proven to be more practical for DNN applications.
- Proposing a novel design named Coded S^3M, which reduces the memory footprint required for weight storage compared to S^3M, thereby reducing power consumption at the accelerator level.

II. Challenges for 8-bit Multiplier Design with Body Approximation

A. Body Approximated Multipliers

In this paper, body approximation refers to making multiplication function more hardware-friendly through simplifications at the circuit architecture level, given that exact multiplication typically requires numerous logic gates. Conventionally, multiplier architecture designs mimic hand-written multiplication by generating an array of partial products, which are subsequently accumulated and compressed to yield the final result [4]. Body approximations are applied during this process either at the partial products generation phase (PPG), using techniques such as simplified Booth encoding, or at the partial products compression phase (PPC), where the number of logic gates in units like full adders or 4-2 compressors is reduced. Often, these approximation strategies are implemented on the least significant parts, and several designs also truncate the least significant columns of the partial products array (Trun.) to achieve greater area and power savings while reducing the number of output registers. Additionally, some studies use genetic algorithms (GA) to automatically generate approximate multipliers, employing basic logic functions like AND and OR to meet specific accuracy thresholds.

Table I summarizes representative designs of approximate multipliers from recent years, while earlier designs are listed in reference [4]. The Distribution column in Table I indicates the data distribution used for accuracy evaluation during the design phase. Notably, only a small portion of these designs take into account the properties of real-world applications when developing these approximations.

B. Computational Cost of Quantization

When implementing a neural network in 8-bit format, quantization is essential, as the original training data is typically in floating-point format. The quantization and dequantization equations, as derived from [20], are expressed as $x^Q = \mathrm{Int}(x/s) + z, \hat{x} = (x^Q - z) \times s$, where x, x^Q and $\hat{x}$ represent real, quantized and dequantized values, s and z denote the scale and zero point of quantization. The outcome of the real-valued matrix multiplication $\mathbf{Y} = \mathbf{A} \cdot \mathbf{W}$, with a matrix size of p, can be approximated with quantized values, as follows:

$$y_{ij} = \sum_{k=1}^{p} a_{ik} w_{kj} \simeq \sum_{k=1}^{p} (a_{ik}^Q - z_a)(w_{kj}^Q - z_w) s_a s_w$$
$$= \sum_{k=1}^{p} (a_{ik}^Q w_{kj}^Q - a_{ik}^Q z_w - w_{kj}^Q z_a + z_a z_w) s_a s_w. \quad (1)$$

In signed quantization, the zero point z is typically set to 0, known as symmetric quantization. On the other hand, asymmetric unsigned quantization involves a non-zero z, leading to more multiplications and additions in Eq. (1). This extra computation may diminish the advantage of quantized integer operations [27]. Thus, this paper focuses on the signed multiplier with symmetric quantization.

TABLE I
Summary of Representative Body Approximated Multipliers.

Work	PPG	PPC	Trun.	GA	Distribution
[5] [6] [21]		✓			Uniform
[7] [22]		✓			Gaussian
[8] [9] [10]	✓	✓			Uniform
[11] [12]				✓	Uniform
[13]				✓	Gaussian
[23] [24]		✓	✓		Uniform
[25] [26]	✓	✓	✓		Uniform

C. Hardware Challenge

Recent approximate multipliers typically incorporate their body approximation strategy into various types of exact digital multipliers from [28] [29], and conduct hardware performance comparisons with their hand-crafted exact multipliers to demonstrate the effectiveness of their approximation designs. However, these hand-crafted multipliers often show worse power, performance, and area (PPA) metrics compared to the DesignWare datapath IP [30]. This suggests that the performance of existing approximate multipliers is not fairly evaluated and compared with the exact multipliers that are commonly used in industrial designs. Our experiments reveal that current 8-bit exact multipliers are inferior to DesignWare IPs, which will be discussed in Section V-A. Therefore, body-approximate multipliers must eliminate additional gates to achieve considerable PPA reduction over commercial multiplier IPs.

D. Accuracy Challenges

Over the past decades, many works [31] [32] have employed systolic arrays to manage matrix multiplication in DNN applications, with the MAC unit being the most computation-intensive component. Fig. 1 exemplifies the structure of a popular MAC unit [33], comprising an 8-bit multiplier and a 24-bit adder with registers to accumulate the multiplication results. This accumulation, depicted by the blue line in Fig. 1,

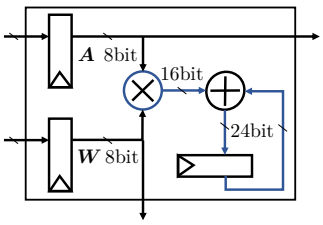

Fig. 1. MAC unit in systolic array-based DNN accelerators.

presents the first challenge for accuracy in approximate multipliers: errors that tend to skew in one direction (either positive or negative) accumulate over multiple iterations, leading to non-zero biased errors in MAC results.

Addressing this accuracy challenge for an 8-bit multiplier with body approximation includes two potential solutions: a high-accuracy multiplier with minimal absolute error, and an unbiased approximate multiplier with a balanced error direction. However, the former approach often results in suboptimal PPA outcomes due to the hardware challenges mentioned earlier, whereas the latter poses significant design challenges. Specifically, once a multiplier is designed with body approximation, its configuration becomes fixed. This is problematic since the diverse data distributions in different DNNs, or even across layers in a single DNN, are given to a configuration-fixed approximate multiplier. Some researchers have explored dynamic approximation strategies to overcome the error accumulation issue [16] [17] [34], but these approaches are often too complex for 8-bit widths, resulting in a significant PPA overhead.

The hardware and accuracy challenges above have directed our focus toward approximation at the input stage rather than the body approximation, as these challenges are better managed with input approximation strategies, which is an area yet to be fully explored.

III. Static Input Segmentation

A. Approximation at Input

Input approximation reduces the bit-width of operands, allowing for the use of a smaller exact multiplier, which simplifies the design and reduces memory usage. Two widely adopted methods for this are quantization and truncation of the least significant bits (LSBs). In truncation, compensation is often applied, typically using a two-input OR gate [19]. Specifically, the LSB of the truncated operands is replaced by the logical OR result of its value with the MSB of the discarded LSBs.

To assess the effectiveness of various input approximation strategies in DNN-like applications, we performed MAC operations on two 2048 randomly generated Gaussian operands using various input approximation strategies. This process was repeated 20,000 times, recording the strategy that yielded the minimum accumulated error compared to the original floating-point result each time. Fig. 2 summarizes the distribution of different strategies achieving the minimum accumulated error.

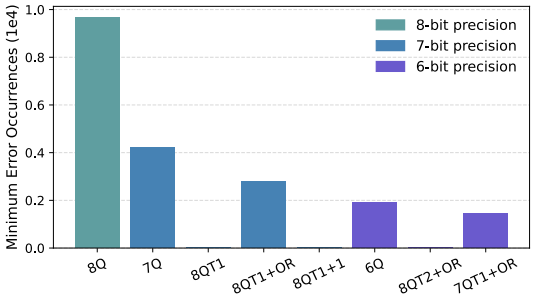

Fig. 2. Minimum error occurrences for different strategies. 8Q: 8-bit quantization. T1: truncating 1 LSB, +OR: OR gate compensation, +1: setting 1 to the LSB for compensation.

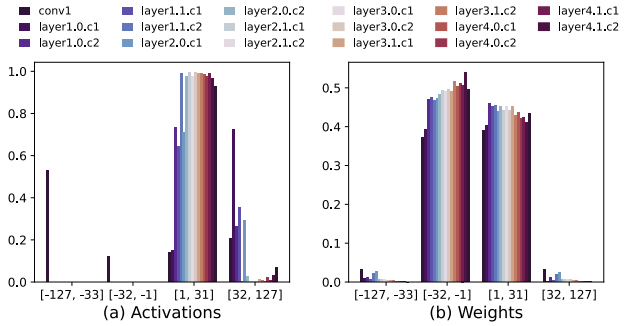

Fig. 3. Distributions of activation and weight of 8-bit quantized ResNet18 on ImageNet dataset. X-axis: Quantized Values. Y-axis: Occurrence Probability.

Note that even 6Q attains the minimum accumulated error in 1,928 (9.64%) out of the total 20,000 cases. We can see that the simple quantization strategy achieves the highest accuracy in the same bit-width precision. For example, among 7-bit precision cases, 7-bit quantization 7Q is the best. Furthermore, 1-bit truncation with OR gate compensation maintains higher accuracy than lower bit precision and 2-bit truncation, e.g., 8QT1+OR $>$ 6Q, 7QT1+OR $>$ 8QT2+OR. On the other hand, the other input approximation strategies fail due to their biased error tendencies.

The superior accuracy of quantization arises from its inherent unbiased property, as each floating-point value is mapped to the nearest integer. On the other hand, a significant precision gap still exists between different bit-width quantizations. An effective design for an input approximate multiplier in DNN applications should, therefore, aim to achieve hardware efficiency comparable to lower bit-widths while maintaining accuracy close to higher bit-widths. Achieving this requires careful analysis of the data distribution in real applications.

B. Static Input Segmentation in DNN

To further explore input approximation strategies more suitable for DNN, we investigate the DNN data distribution. Fig. 3 shows the data distribution of 8-bit quantized ResNet18 on ImageNet dataset. The majority of data, especially weights, is concentrated within a deviation of 32 (2^5) from 0. This concentration suggests the potential for achieving high accuracy

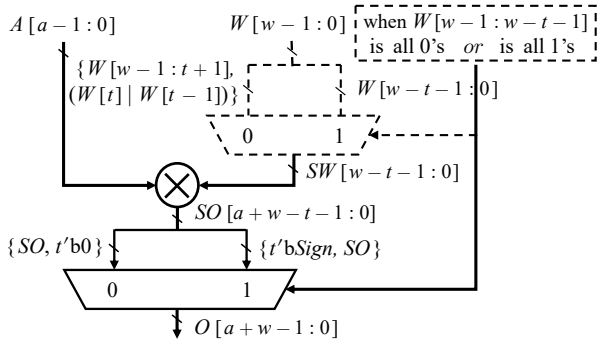

Fig. 4. Structure of S^{3}M.

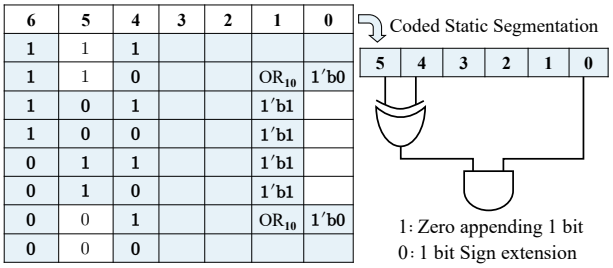

Fig. 5. Coded S^{3}M example for 6-bit segmentation from 7-bit word. Blue background represents the segmented operand; Precise for values within [-16, 15] corresponding to the top and bottom rows; slight error for values outside this range with coded shift information for zero-padding at LSB or sign-extension at MSB.

by maintaining exact multiplication within the range of [-32, 31], where only 6 bits contain information.

SSM is suited for handling this data distribution property when segmenting 6 bits from 8-bit input operands. Input operands residing within the range of [-32, 31] can be identified by the presence of MSBs as either 000 or 111. In the signed SSM from [19], a check is performed on the MSBs of the input operands. If these checked bits are either all 0s or 1s, they are truncated, except the sign bit, without accuracy loss from 8-bit quantization. For checked bits not all-0s or all-1s, indicating a larger absolute magnitude of operand, the LSBs are truncated with OR gate compensation.

However, SSM faces two challenges: (1) the online segmenting significantly increases the overall multiplication delay, and (2) activations are not strictly confined to small values as illustrated in Fig. 3. While the original SSM proposes offline weight segmentation [18], the multiplier must wait for activation segmentation. Furthermore, the segmentation information from both activation and weight must be combined to determine the shift amount for obtaining the final multiplication result, which increases the delay further.

IV. Proposed Multipliers

A. Static Semi-Segmented Multiplier (S^{3}M)

To address the two challenges pointed out in the previous section, we have simplified the SSM design to create the S^{3}M. This new S^{3}M design segments only one operand offline in relation to weights, which are typically more concentrated in smaller values than activations and are pre-trained and stored in memory for inference.

Fig. 4 illustrates the structure of S^{3}M, where a and w are the bit-width of the activations A and weights W, respectively, and t represents the number of bits to be truncated. SW denotes the segmented weight, and SO is the result of the inner multiplication, with O indicating the final multiplication output. The segmentation, represented by the dotted line, is completed offline. In this design, S^{3}M features only an inner multiplier with a size of $a \times (w-t)$ and a two-input multiplexer serving as the final shifter, meaning that the overhead due to segmentation is significantly reduced. Consequently, S^{3}M effectively decreases both delay and power consumption compared to the original SSM model. Meanwhile, the activations are processed using simple quantization, a method well-suited to their typically broad data distribution.

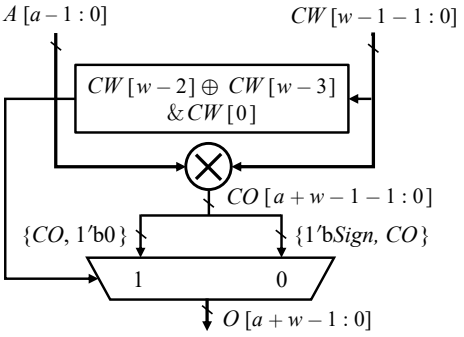

Fig. 6. Structure of Coded S^{3}M.

B. Coded S^{3}M

For S^{3}M in Fig. 4, when $a = w = 8$ and $t = 2$, the design requires storing 8-bit activations and 7-bit weights (including 1-bit shift information, which corresponds to the selection signal of the final multiplexer). This results in a 1-bit reduction in memory usage compared to an 8-bit multiplier with body approximation or online segmentation. Moreover, we have found that the shift bit in S^{3}M can bring about further improvements in PPA at the DNN accelerator level.

To eliminate the extra 1-bit memory usage per weight, we encode the shift information into the weight operands offline. Here, considering that 7QT1 offers better accuracy than 8QT2, as shown in Fig. 2, we opt for truncating only one LSB in each case. In situations where the two MSBs of the segmented weight are opposite, the LSB is set to 0 in non-shifting cases and to 1 in shifting cases. In non-shifting cases, the second least significant bit (second LSB) is adjusted by an OR operation between the LSB and the second LSB to mitigate the coding error. In shifting cases, setting the LSB to 1 helps distinguish them from non-shifting cases while also partially mitigating segmentation errors. Conversely, when the

two MSBs of the segmented weight are the same, all cases are non-shifting. Fig. 5 illustrates an example of the coded method for $w = 7$ with 1-bit truncation, where the blue-boxed area indicates the weight operand after segmentation. In this example, when the two MSBs of the segmented operand are opposite and the LSB is 1 simultaneously, the inner multiplier output undergoes a left shift by one bit.

Fig. 6 shows the structure of the proposed Coded S^{3}M, where CW denotes the coded weights and CO is the result of the inner multiplier. The addition of a simple decoder, consisting of an XOR and an AND gate, incurs minimal overhead compared to S^{3}M while efficiently saving one bit storage for each weight. Note that the delay of (CW[w-2] $\oplus$ CW[w-3]) & CW[0] is shorter than that of the inner multiplier, and hence the entire delay is expected to be the same as S^{3}M.

V. Experimental Results

We quantized a pre-trained VGG11-BN model on CIFAR-10 and CIFAR-100 datasets downloaded from [35], and applied similar quantization to ResNet-18, ResNet-50, and ResNet-101 on the ImageNet dataset from [36]. The post-training quantization method described in [20] was employed, which involved replacing 8-bit exact multipliers with different bit-width quantization and various approximate multipliers for accuracy evaluation. We also developed a DNN accelerator, featuring an output-stationary systolic array with integrated SRAMs, where the bit-width was adjusted based on the activations and weights. All circuits were synthesized using Design Compiler with a commercial 22nm 0.81V library. For the multipliers, power consumption and maximum delay were estimated from the synthesis data, targeting a challenging 0.3ns clock period. At the accelerator level, power assessments were performed using PrimeTime, with the test data conforming to the distribution observed in ResNet-18 on ImageNet, and the netlist was synthesized under a stringent 0.5ns clock period.

A. Multiplier Level Evaluation

Table II presents the hardware results of signed 8-bit exact multipliers. The Dadda and Booth multipliers were self-implemented, while Mul8s-1KV6 is the signed 8-bit exact multiplier taken from the multiplier library [11]. The increased power-delay product (PDP) relative (Rel.) to the DesignWare IP exposes a gap between hand-crafted multipliers and commercial multiplier IP, which is the baseline performance loss for body approximation multipliers.

Many prior studies on approximate multipliers evaluated their designs using low-challenge datasets such as MNIST or CIFAR, which typically exhibit greater error resilience compared to ImageNet. Fig. 7 presents the PDP and classification accuracy on the CIFAR dataset for various signed multipliers. DesignWare IP serves as the baseline with varying bit-width multiplications ranging from 8×8 to 4×4 with simple quantization. The experiment includes the proposed S^{3}M and

TABLE II
Exact Multiplier Synthesis Results (Speed-Oriented).

Signed 8-bit Exact Multiplier	Area (μm^2)	Power (μW)	Delay (ns)	Rel.PDP (%)
DesignWare IP	166	311	0.34	-
Dadda	150	274	0.41	+ 6.24
Booth	175	324	0.35	+ 6.60
Mul8s-1KV6 [11]	137	238	0.50	+ 12.26

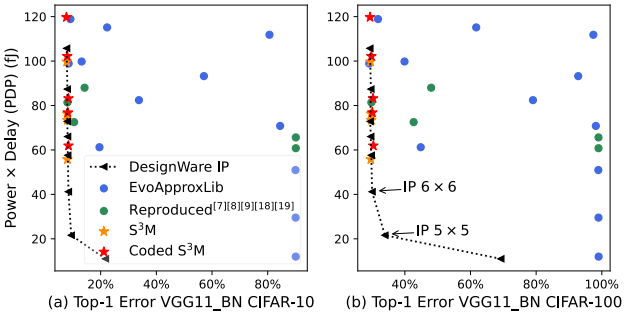

Fig. 7. PDP vs Top1 error for several multipliers on VGG11_BN model pre-trained on CIFAR dataset.

Coded S^{3}M with different bit-widths, along with signed 8-bit approximate multipliers from EvoApproxLib [11], and several other reproduced signed 8-bit approximate multipliers [7] [8] [9]. It also encompasses the signed 8-bit SSM (6-bit segmentation) with weights pre-segmented [18] and OR gate compensation strategies [19], providing a comprehensive comparison.

In Fig. 7, most 8-bit multipliers with body approximation exhibit significant errors, even when evaluated on the CIFAR-10 dataset. The errors are more pronounced on the CIFAR-100 dataset, highlighting the disparities between these approximate multipliers and the DesignWare IP baseline. The SSM related designs are all in the same line with DesignWare IP, demonstrating their suitability for DNN applications by performing 8-bit exact multiplication for predominantly small-value input operands with a smaller inner multiplier. However, given the high error resilience of the CIFAR datasets, utilizing 6-bit quantization or even 5-bit quantization with the exact IP multiplier may offer the optimal energy-efficient solution, as the increase in error is minimal.

For the high-challenge dataset evaluation, we deployed multiplier designs from Fig. 7 on ResNet with the ImageNet dataset. The PDP and classification error are depicted in Fig. 8, where the designs with less accuracy than 6-bit quantization (6×6) are excluded. The $\times$ symbol in Fig. 8(a) indicates the inner multiplier precision of static segmented multipliers. It is evident that only one multiplier with body approximation, Mul8s-1KVM [11], appears in Fig. 8, yet it struggles to maintain acceptable accuracy for ResNet101. In contrast, all proposed S^{3}Ms designs and several Coded S^{3}Ms models demonstrate a better trade-off, as indicated by their lower left

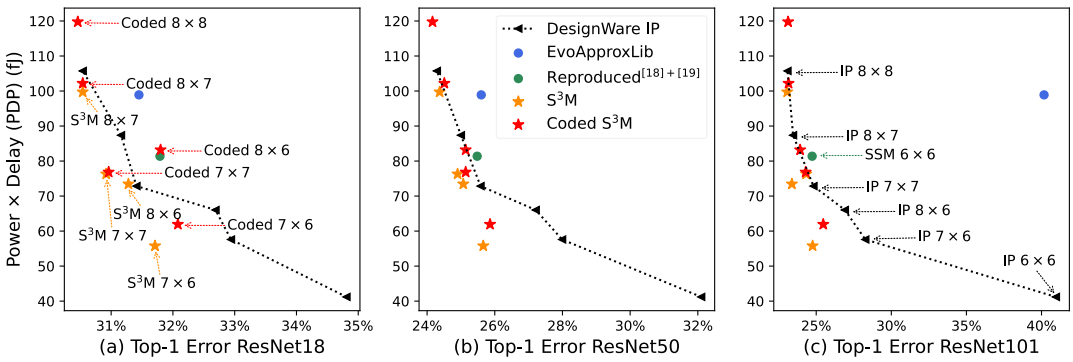

Fig. 8. PDP vs Top1 error for several multipliers on ResNet model pre-trained on ImageNet dataset. The figure displays only one body approximation design; multipliers not presented here have Top-1 errors exceeding that of IP 6×6.

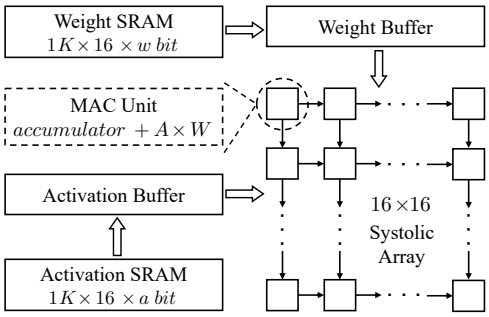

Fig. 9. Accelerator structure with a 16 × 16 systolic array.

positioning compared to exact IP multipliers. For instance, S^{3}M 8×7 achieves a 5.69% PDP reduction from IP 8×8 with an accuracy loss ranging from -0.02% to 0.08%. On the other hand, the prior SSM 6 × 6 design is positioned in the upper right compared to exact IP multipliers, indicating its inferiority in the trade-off. Specifically, IP 7 × 7 offers a larger PDP reduction with a smaller Top-1 classification error. Meanwhile, S^{3}M 7 × 6 reduces PDP by 31.58% from SSM 6 × 6 while maintaining similar classification accuracy, as shown in Fig. 8 (a). In this scenario, the proposed S^{3}M emerges as an energy-efficient alternative to exact IP multipliers at the multiplier level and the accelerator level when the memory bit-width is fixed.

B. Accelerator Level Evaluation

We integrated the same multipliers from Fig. 8 into the output-stationary systolic array in the DNN accelerator depicted in Fig. 9. SRAM bit-widths a and w adapt to the bit-width of activation and weight inputs in the MAC unit within the array. Additionally, the accumulator bit-width decreases to align with the reduction in SRAM bit-widths.

Fig. 10 shows the results, where 8 × 8 in (a) indicates the inner multiplier bit-width of S^{3}M, while a8_w8 in (c) represents the required activation and weight SRAM bit-widths. Compared with Fig. 8, the PDP of the proposed Coded S^{3}M shows improvement and surpasses that of S^{3}M and exact IP multipliers in several cases due to the reduced memory and accumulator bit-widths.

The performance of the representative designs is detailed in Table III, which includes a floating-point 8-bit multiplier with 2-bit exponent and 5-bit mantissa (FP8-E2M5) [37]. This FP8 multiplier was selected for its 5-bit mantissa + shift structure having a similarity with SSM, enabling a comprehensive comparison. However, while the FP8-E2M5 maintains acceptable accuracy with a 50% PDP reduction compared to the INT8 case at the multiplier level, it incurs about a 63% PDP overhead at the accelerator level due to the complex FP16 accumulation. Additionally, the body approximation multiplier Mul8s-1KVM also exhibits a higher PDP than IP 8 × 8 at the accelerator due to its larger maximum delay.

Notably, regarding the proposed multipliers, the area and PDP improvements from the IP baseline are more significant at the multiplier level than at the accelerator level. For instance, at the multiplier level, Coded S^{3}M 7×6 achieves a substantial 41.33% reduction in PDP compared to IP 8 × 8, whereas at the accelerator level, the reduction is 22.39%. Additionally, the S^{3}M 8 × 7 demonstrates a 5.69% PDP reduction at the multiplier level and a 1.77% PDP reduction at the accelerator level. The relatively smaller PDP reduction at the accelerator level can be attributed to the composition of the accelerator in this experiment, which consists of approximately 22.5% power consumption from the multiplier, while 19.7% from SRAM, and the remaining 57.8% from components like the accumulator and register. This indicates that evaluating at the multiplier level alone is insufficient to fully assess the effectiveness of approximate multipliers.

On the other hand, Coded S^{3}M 8 × 7 achieves no accuracy loss for ResNet18 and it offers a higher PDP reduction of 4.25% at the accelerator level compared to 3.35% at the multiplier level, benefiting from a reduced memory bit-width. Furthermore, Coded S^{3}M 8×8 demonstrates enhanced classification accuracy for ResNet18 and ResNet50 by 0.08% and 0.13%, respectively, thanks to 8-bit weights derived from 9-bit quantization. Despite a 13.37% increase in PDP at the multiplier level, this rise is mitigated at the accelerator level, where the increase is limited to just 3.80%.

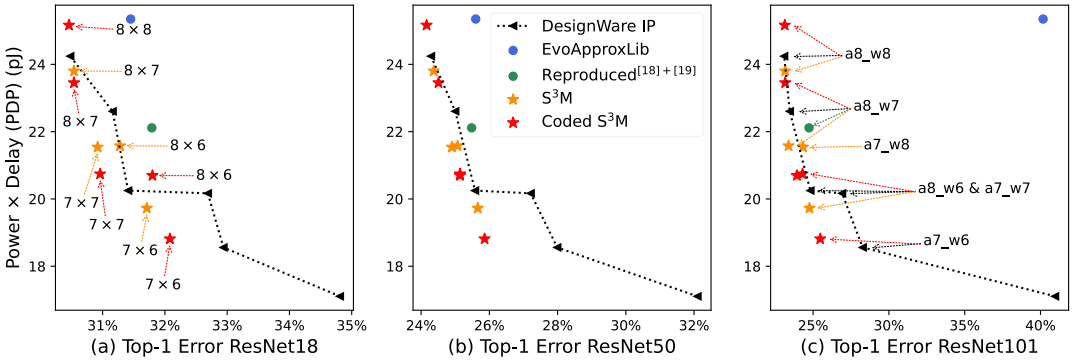

Fig. 10. PDP vs Top1 error for systolic arrays with several multipliers on ResNet model pretrained on ImageNet dataset.

TABLE III
EVALUATION RESULTS OF SEVERAL REPRESENTATIVE MULTIPLIERS (SPEED-ORIENTED).

Design $a \times w$	Multiplier Level				Accelerator Level				Top-1 Error on ImageNet		
	Area (μm^2)	Power (μW)	Delay (ns)	Rel.PDP (%)	Area (μm^2)	Power (mW)	Delay (ns)	Rel.PDP (%)	ResN18 (%)	ResN50 (%)	ResN101 (%)
IP 8×8	166	311	0.34	-	162883	45.74	0.53	-	30.54	24.29	23.09
IP 8×7	135	259	0.34	−16.72	150962	42.63	0.53	−6.80	31.16	25.00	23.46
IP 7×7	142	232	0.31	−31.98	140442	38.94	0.52	−16.47	31.41	25.58	24.82
IP 8×6	112	214	0.31	−37.26	129464	38.05	0.53	−16.80	32.69	27.21	26.91
IP 7×6	102	180	0.32	−45.53	119447	35.69	0.52	−23.44	32.94	27.99	28.27
IP 6×6	95	142	0.30	−59.71	108125	32.89	0.52	−29.44	34.81	32.10	41.96
FP8-E2M5	117	165	0.32	−50.07	246066	41.95	0.94	+62.67	30.95	24.74	23.62
Mul8s-1KVM [11]	122	215	0.46	−6.47	166203	42.96	0.59	+4.58	31.45	25.60	41.17
SSM 6×6 [18] [19]	140	209	0.39	−22.91	152711	40.95	0.54	−8.79	31.79	25.48	24.73
S^3M 8×7	157	277	0.36	−5.69	175246	46.67	0.51	−1.77	30.54	24.37	**23.07**
S^3M 7×7	140	220	0.34	−29.26	153602	42.23	0.51	−11.15	30.92	24.90	24.33
S^3M 8×6	138	218	0.34	−29.90	150452	42.31	0.51	−10.98	31.28	25.07	23.38
S^3M 7×6	110	169	0.33	−47.26	136602	38.68	0.51	−18.62	31.71	25.66	24.76
Coded S^3M 8×8	164	324	0.37	+13.37	180894	47.48	0.53	+3.80	**30.46**	**24.16**	23.13
Coded S^3M 8×7	156	292	0.35	−3.35	166051	44.25	0.53	−4.25	30.54	24.51	23.16
Coded S^3M 7×7	131	228	0.34	−26.69	149350	39.89	0.52	−14.43	30.96	24.14	24.30
Coded S^3M 8×6	127	240	0.34	−22.83	138346	39.05	0.53	−14.62	31.80	25.14	23.93
Coded S^3M 7×6	100	188	0.33	−41.33	126453	36.18	0.52	−22.39	32.08	25.86	25.47

The previously favored multiplier design for DNN applications, SSM 6×6 [18] [19], appears more inferior to commercial multiplier IPs in Fig. 10 than in Fig. 8. Furthermore, other existing approximate multipliers with body approximation fail to achieve acceptable accuracy and comparable PDP against commercial IPs, highlighting the obstacle of improving DNN inference accelerator performance through approximate multiplier design alone. In contrast, although the absolute PDP gains from the proposed S^3M and Coded S^3M are modest relative to commercial IPs, it is clear that most of these models are positioned closer to the coordinate origin in both Figs. 8 and 10. This positioning demonstrates a superior trade-off between PDP and classification accuracy for computation-demanding inference tasks, like ResNet on ImageNet, which indicates that that the proposed designs are approaching the practical limits of multiplier design for real-world DNN applications.

VI. CONCLUSION

In this paper, we highlight the challenges in traditional 8-bit approximate multiplier designs with body approximation and showcase the advantages of simple quantization and SSM for DNN applications. By exploiting the properties of weights being fixed offline and having biased distributions, we simplified the structure of SSM and propose S^3M, which shows better energy-accuracy trade-offs compared to other approximate multipliers in this experiment. Additionally, to decrease the memory bit-width in the accelerator, we introduce Coded S^3M. Both the proposed S^3M and Coded S^3M models achieve over 40% PDP reduction at the multiplier level and approximately 20% at the accelerator level in the 7×6 configuration, compared to the 8-bit exact IP multiplier, while

maintaining similar accuracy in the 8×7 case, with respective PDP reductions of 1.77% and 4.25% in the DNN inference accelerator. More importantly, the proposed S^3M and Coded S^3M models bridge the gap in PDP and inference accuracy trade-offs among commercial multiplier IPs of varying precisions, offering more energy-efficient options for multiplication and inspiring future approximate multiplier designers about the design direction.

ACKNOWLEDGMENTS

This work was supported in part by JST SPRING, Grant Number JPMJSP2110.

REFERENCES

[1] Norman P Jouppi, Cliff Young, Nishant Patil, *et al.* "In-datacenter performance analysis of a tensor processing unit." *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[2] Giorgos Armeniakos, *et al.* "Hardware approximate techniques for deep neural network accelerators: A survey." *ACM Computing Surveys*, 55(4):1–36, 2022.

[3] Bert Moons and Marian Verhelst. "An energy-efficient precision-scalable convnet processor in 40-nm cmos." *IEEE Journal of solid-state Circuits*, 52(4):903–914, 2016.

[4] Ying Wu, Chuangtao Chen, *et al.* "A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits." *ACM Transactions on Design Automation of Electronic Systems*, 29(1):1–37, 2024.

[5] Antonio Giuseppe Maria Strollo, *et al.* "Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(9):3021–3034, 2020.

[6] Gunho Park, *et al.* "Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(7):2950–2961, 2021.

[7] Hang Xiao, *et al.* "Fast and high-accuracy approximate mac unit design for cnn computing." *IEEE Embedded Systems Letters*, 14(3):155–158, 2021.

[8] Suganthi Venkatachalam, Hyuk Jae Lee, and Seok-Bum Ko. "Power efficient approximate booth multiplier." *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–4, 2018.

[9] Gunho Park, Jaeha Kung, and Youngjoo Lee. "Simplified compressor and encoder designs for low-cost approximate radix-4 booth multiplier." *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(3):1154–1158, 2022.

[10] Weiqiang Liu, *et al.* "Design of approximate radix-4 booth multipliers for error-tolerant computing." *IEEE Transactions on computers*, 14(3):66(8):1435–1441, 2017.

[11] Vojtech Mrazek, *et al.* "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods." *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 406–418, 2017.

[12] Vojtech Mrazek, Lukas Sekanina, *et al.* "Libraries of approximate circuits: Automated design and application in cnn accelerators." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):406–418, 2020.

[13] Zhen Li, Su Zheng, *et al.* "Adaptable approximate multiplier design based on input distribution and polarity." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(12):1813–1826, 2022.

[14] Vojtech Mrazek, *et al.* "Alwann: Automatic layer-wise approximation of deep neural network accelerators without retraining." *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[15] Zois-Gerasimos Tasoulas, *et al.* "Weight-oriented approximation for energy-efficient neural network inference accelerators." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(12):4670–4683, 2020.

[16] Soheil Hashemi, R Iris Bahar, and Sherief Reda. "Drum: A dynamic range unbiased multiplier for approximate applications." *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 418–425, 2015.

[17] Shaghayegh Vahdat, *et al.* "Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(5):1161–1173, 2019.

[18] Srinivasan Narayanamoorthy, *et al.* "Energy-efficient approximate multiplication for digital signal processing and classification applications." *IEEE transactions on very large scale integration (VLSI) systems*, 23(6):1180–1184, 2014.

[19] Antonio Giuseppe Maria Strollo, *et al.* "Approximate multipliers using static segmentation: Error analysis and improvements." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(6):2449–2462, 2022.

[20] Pytorch Docs: Quantized dtypes and quantization schemes. https://pytorch.org/docs/stable/quantization-support.html

[21] M. Zhang, *et al.* "Area efficient approximate 4–2 compressor and probability-based error adjustment for approximate multiplier." *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1714-1718, 2023.

[22] L. Du, *et al.* "A Low-Power DNN Accelerator With Mean-Error-Minimized Approximate Signed Multiplier," in IEEE Open Journal of Circuits and Systems." *IEEE Open Journal of Circuits and Systems*, vol. 5, pp. 57-68, 2024.

[23] R. Duan, *et al.* "A Hardware-Efficient Approximate Multiplier Combining Inexact Same-weight N:2 Compressors and Remapping Logic with Error Recovery." *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, pp. 1-6.

[24] F. Sabetzadeh, *et al.* "An Ultra-Efficient Approximate Multiplier With Error Compensation for Error-Resilient Applications." *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 2, pp. 776-780, Feb. 2023.

[25] Y. He, *et al.* "A Probabilistic Prediction-Based Fixed-Width Booth Multiplier for Approximate Computing." *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4794-4803, Dec. 2020.

[26] Z. Aizaz, *et al.* "Area and Power Efficient Truncated Booth Multipliers Using Approximate Carry-Based Error Compensation." *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 2, pp. 579-583, Feb. 2022.

[27] Hao Wu, *et al.* "Integer quantization for deep learning inference: Principles and empirical evaluation." *arXiv preprint arXiv:2004.09602*, 2020.

[28] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*, Isevier, 2011.

[29] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[30] Synopsys datapath ip. https://www.synopsys.com/dw/buildingblock.php

[31] Mohammed Elbtity, *et al.* "Aptpu: Approximate computing based tensor processing unit." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(12), 2022.

[32] Ourania Spantidi, *et al.* "Targeting dnn inference via efficient utilization of heterogeneous precision dnn accelerators." *IEEE Transactions on Emerging Topics in Computing*, 11(1):112–125, 2022.

[33] Zuodong Zhang, *et al.* "Read: Reliability-enhanced accelerator dataflow optimization using critical input pattern reduction." *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[34] Mohsen Imani, *et al.* "Rmac: Runtime configurable floating point multiplier for approximate computing." *Proceedings of the international symposium on low power electronics and design (ISLPED)*, pages 1–6, 2018.

[35] chenyaofo. pytorch-cifar-models, Pretrained models on CIFAR10/100, 2021. https://github.com/chenyaofo/pytorch-cifar-models

[36] Pytorch hub for researhers: Deep residual networks pre-trained on imagenet. https://pytorch.org/hub/pytorch_vision_resnet/

[37] Mart van Baalen, *et al.* "Fp8 versus int8 for efficient deep learning inference." *arXiv preprint arXiv:2303.17951)*, 2023.