ViT-slice: End-to-end Vision Transformer Accelerator with Bit-slice Algorithm

Dongjin Shin[1,*], Insu Choi[1,*] and Joon-Sung Yang[1,2,3]
[1]Department of Electrical and Electronic Engineering, [2]Department of Systems Semiconductor Engineering,
[3]BK21 Graduate Program in Intelligent Semiconductor Technology, Yonsei University, Seoul, South Korea
{dongjin_97,insuofficial,js.yang}@yonsei.ac.kr

ABSTRACT

Vision Transformers have demonstrated remarkable performance in various vision tasks. However, general-purpose processors, such as CPUs and GPUs, face challenges in efficiently handling the inference of Vision Transformers. To address the issue, prior works have focused on accelerating only attention due to its high computational cost in NLP Transformers. In contrast, Vision Transformers demonstrate a higher computational cost due to linear modules such as linear transformation, linear projection and Feed-Forward Network (FFN), compared to attention. In this paper, we present ViT-slice, an algorithm-architecture co-design that enhances end-to-end performance and energy efficiency by optimizing not only attention but also linear modules. At the algorithm level, we propose *bit-slice compression* that avoids storing the redundant most significant bits (MSBs). Additionally, we present *bit-slice dot product with early skip* to efficiently compute the dot product using bit-sliced data. To enable early skip during the dot product computation, we leverage a trainable threshold. On the hardware level, we introduce a specialized bit-slice dot product unit (BSDPU) to efficiently process the bit-slice dot product with early skip algorithm. Additionally, we present a bit-slice encoder and decoder for on-chip bit-slice compression. ViT-slice achieves 244×, 35.3×, 16.8×, 10.4×, 5.0× end-to-end speedup over Xeon CPU, EdgeGPU, TITAN Xp GPU, Sanger accelerator and ViTCoD accelerator, respectively.

1 INTRODUCTION

Vision Transformers have emerged as a powerful alternative to Convolutional Neural Networks (CNNs), demonstrating superior performance in diverse vision tasks. However, its adoption in resource-constrained edge environments remains a challenge due to its huge parameter size and computational cost. Addressing the challenges, it is essential to design an algorithm that compresses the parameters and efficiently accelerates the end-to-end operation of Vision Transformers, and the hardware that supports it.

Transformers have several key operations that significantly contribute to the overall computation and memory access cost. The operations include attention and linear modules, such as linear transformation, linear projection, and Feed-Forward Network (FFN). As the embedding dimension increases, linear modules incur a quadratic increase in computational cost, while attention shows a quadratic computational cost increase with respect to the sequence length. In NLP Transformers, the sequence length is larger than the embedding dimension, whereas in Vision Transformers, the sequence length is smaller than the embedding dimension. Consequently, Vision Transformers have a larger overhead for linear operations as opposed to attention operations.

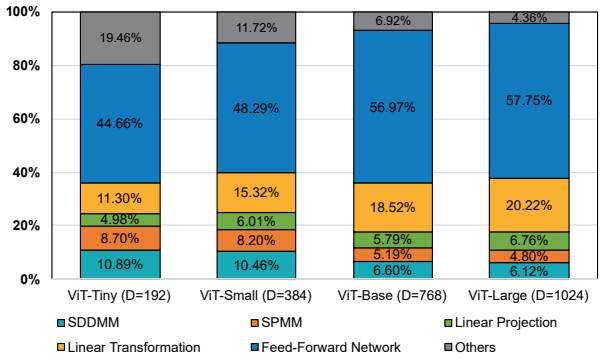

Figure 1: End-to-end runtime breakdown for various sizes of ViT models (D: Embedding Dimension)

To validate these observations, we profile a runtime breakdown of ViT-Tiny/Small/Base/Large [2] using the TITAN Xp GPU. The analysis, depicted in Figure 1, clearly indicates that the linear modules dominate the overall runtime of ViTs. As the model scales and the embedding dimension D increases, the runtime proportion occupied by linear modules increases significantly, reaching 84% in the ViT-Large model. Hence, to achieve more significant enhancements in the overall end-to-end latency, it becomes crucial to devise strategies that comprehensively optimize all major operations of the Vision Transformers, including attention and linear modules.

In this paper, we present ViT-slice, a novel algorithm-architecture co-design approach to effectively accelerate end-to-end computation in Vision Transformers. The contributions are listed as follows:

- **ViT-slice, an algorithm-architecture co-design**, significantly reduces the end-to-end latency of Vision Transformers by optimizing both attention and linear modules.
- **Bit-slice compression** efficiently utilizes memory bandwidth by avoiding the storage of the same four MSBs pattern.
- **Bit-slice dot product with early skip** enables efficient dot product computation by leveraging bit-sliced data and implementing early skip in the dot product process through the utilization of pre-trained thresholds.

*Both authors contributed equally to the paper (correspondence to Joon-Sung Yang <js.yang@yonsei.ac.kr>).

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3655955

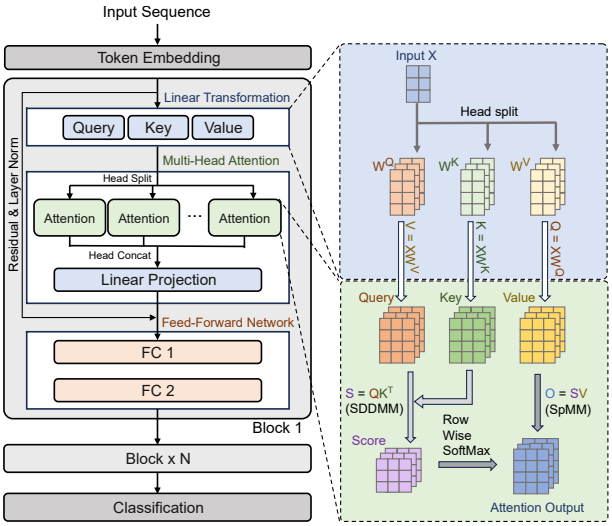

Figure 2: Structure of Vision Transformers and the computation process of linear transformation and attention.

- **Trainable threshold** used for early skip operation can be fine-tuned with novel differentiable pruning functions during the gradient-based training process.

2 BACKGROUND

2.1 Vision Transformer

The structure of Vision Transformers is illustrated in Figure 2. In the main stream of Vision Transformer, the first step involves creating queries, keys, and values, which serve as the inputs for multi-head attention. This process is called linear transformation. The queries and keys are then used in sample-based dense-dense matrix multiplication (SDDMM) operation to compute a score that captures the correlation information between patches. The score, together with the values, is used in a sparse-dense matrix multiplication (SpMM) operation to obtain attention output. Hence, attention operation involves both SDDMM and SpMM. After concatenating the attention outputs in each head, a linear projection is performed to derive the final result of the multi-head attention. By applying the FFN operation to the multi-head attention output, the Transformer block output is obtained. Vision Transformers is constructed by repeating the Transformer blocks. The repetition allows the model to capture hierarchical information within the input data.

2.2 Quantization

Quantization, a methodology employed in neural networks, serves to diminish bit precision, thereby mitigating computational demands and minimizing memory consumption. Nevertheless, it is imperative to acknowledge that the shift from higher to lower bit-widths introduces inherent noise, potentially compromising the accuracy of neural networks. Therefore, various quantization methods, including Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), have been studied [3, 7, 9, 10]. PTQ is a lightweight quantization method that does not involve a training process. On the other hand, QAT relies on fine-tuning neural networks using training data while applying simulated quantization operations [7]. In this paper, LSQ [3] is utilized to quantize Vision Transformer models, as it has demonstrated remarkable success as a QAT method primarily applied to quantize CNNs.

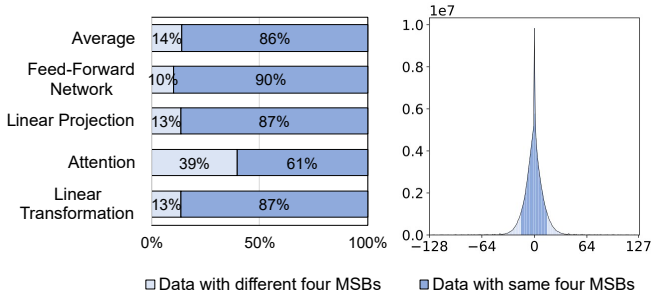

Figure 3: Data distribution within a Vision Transformer.

2.3 Accelerators for Transformers

In the field of Transformers, various accelerators have been introduced. Firstly, A^3 [5] reduces the computational cost of attention using a greedy iterative approximation. SpAtten [14] applies both cascade token pruning and head pruning to remove unnecessary tokens and heads, creating a coarse-grained structured sparsity pattern, resulting in a low sparsity ratio. To address the issue, Sanger [11] approaches the challenge differently, creating a fine-grained structured sparsity pattern by generating a bit mask for pruning. ViTCoD [16] leverages the fixed sequence length of Vision Transformers to facilitate pruning through a static sparsity pattern. Despite its innovation, ViTCoD optimizes attention alone, hence the end-to-end speedup is modest compared to the speedup achieved in attention. Therefore, in this paper, we propose an algorithm-architecture co-design that aimed to improve end-to-end speedup.

3 PROPOSED VIT-SLICE ALGORITHM

3.1 Motivation

We quantize the weights and activations of the models into 8-bit integers, with the aim of ensuring comprehensive support for a wide range of models with varying sizes and architectures. Figure 3 illustrates the data distribution within a Vision Transformer. Notably, 86% of the quantized values fall within the range of -16 to 15, with their four MSBs being either 0000_2 or 1111_2. Based on this observation, we introduce a novel technique called *bit-slice compression*, where only four LSBs are stored if four MSBs are identical (i.e., all 0*s* or all 1*s*), and to prevent information loss, Metadata such as MSB check bits (MCB) and sign bit are added (Figure 4 (a)). Furthermore, we present *bit-slice dot product with early skip*, which facilitates the efficient processing of bit-sliced data (Figure 4 (b)). The proposed algorithms reduce both memory access and computational cost, thereby contributing to enhanced energy efficiency and performance in the context of 8-bit quantized Vision Transformers.

3.2 Bit-slice Compression

Bit-slice compression slices the bits in half and applies compression based on the four MSBs. The original data is divided into three

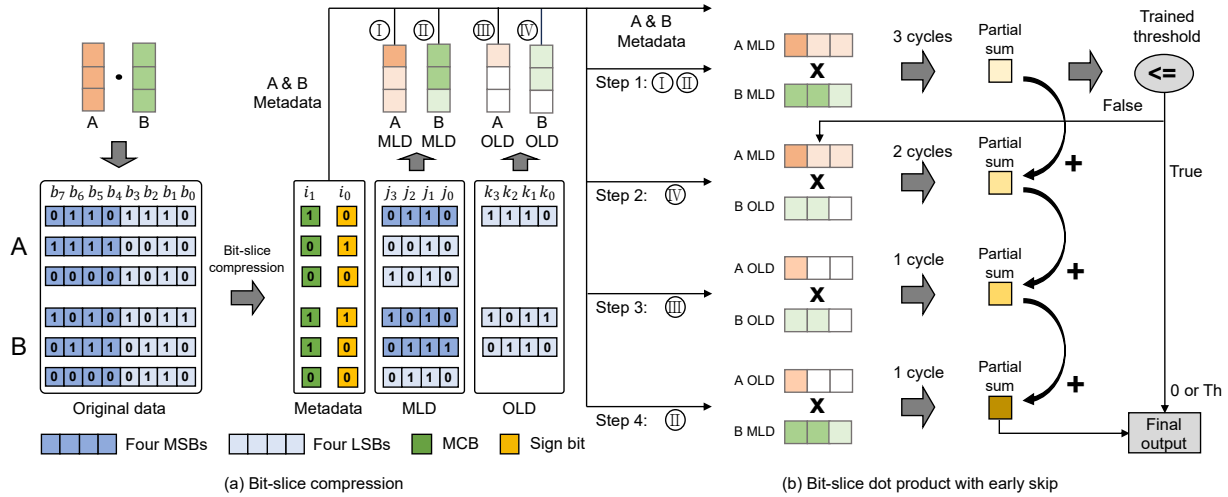

Figure 4: Dot product process of vector A and B using (a) bit-slice compression and (b) bit-slice dot product with early skip.

fields: 1) Metadata, 2) MSBs or LSBs data (MLD), and 3) Only LSBs data (OLD), as illustrated in Figure 4 (a). The Metadata includes MCB and a sign bit. If the four MSBs differ, the MCB is set to 1, and the MSB of the original data is stored as the sign bit, and the four MSBs are stored as the MLD, while the LSBs 4 bits are stored as the OLD. If the four MSBs are the same, the MCB is set to 0, and the MSB of the original data is stored as the sign bit, and the LSBs 4 bits are stored as MLD. In the example, the first row of original data A in Figure 4 (a) is 0110_1110 ($b_7b_6b_5b_4\_b_3b_2b_1b_0$). Because four MSBs ($b_7$-$b_4$) are not identical, MCB in bit-slice compressed data is set 1 (i_1 in Metadata) and the sign bit is 0 (b_7 in data A is written to i_0 in Metadata). Then, 0110 ($b_7b_6b_5b_4$) is written to MLD ($j_3j_2j_1j_0$), and 1110 ($b_3b_2b_1b_0$) is copied to OLD ($k_3k_2k_1k_0$). The second row in the original data A, 1111_0010, is compressed as 0 (MCB) in i_1 in Metadata, 1 (sign bit) in i_0 in Metadata, and 0010 ($b_3b_2b_1b_0$) is written to $j_3j_2j_1j_0$ in MLD. The approach effectively compresses the data by omitting the storage of the four MSBs when they are identical.

3.3 Bit-slice Dot Product

We propose bit-slice dot product with early skip to efficiently perform dot product operations using data compressed through bit-slice compression. In Figure 4 (b), we utilize Metadata, MLD, and OLD, which are obtained from Figure 4 (a), to calculate the dot product output. For I - IV in Figure 4 (a), dark colors represent data containing four MSBs, whereas light colors indicate data containing four LSBs. Finally, white colors signify data that is zero-skipped due to the compressed sparse column (CSC) format [6].

Initially, A & B Metadata is loaded and used throughout step 1-4. Subsequently, A & B MLD is loaded to perform a dot product operation in step 1. Assuming the use of one multiplier, the step takes 3 cycles to generate a partial sum. The partial sum is then compared to a threshold. If the partial sum is less than or equal to the threshold, it is masked either to 0 or a threshold value (Th), and the calculation ends which is early skip (i.e., it skips downstream dot product operations). When an early skip occurs during only SDDMM, the masked value is assigned as Th. For other operations, including SpMM and linear modules, the masked value is set to 0. However, if the partial sum exceeds the threshold value, the process then progresses to step 2. In step 2, B OLD is loaded and a dot product operation is performed with A MLD. Since there are two overlapping parts of non-zero data, it takes a total of 2 cycles to generate a partial sum, which is then accumulated with the previous partial sum. In the next step, A OLD is loaded and a dot product operation is executed with B OLD. This step involves a single overlapping part of non-zero data and requires 1 cycle to complete. The partial sum is accumulated with the previous partial sum. In the final step, a dot product operation is conducted with A OLD and B MLD for 1 cycle. The partial sum is accumulated to calculate the final output. The order of calculations in bit-slice dot product with early skip is determined to minimize the SRAM access. Bit-slice dot product with early skip significantly enhances the performance and energy efficiency of both the attention and linear modules in Vision Transformers.

3.4 Trainable Threshold

The threshold value used for the early skip operation is fine-tuned within the QAT process to achieve the optimal trade-off between computational efficiency and model accuracy. To facilitate it, we introduce a novel approximate differentiable pruning function called soft threshold pruning (STP), incorporated with a soft binary mask (SBM) function that generates a softened binary mask. The STP and SBM functions comprise two distinct versions: STP_S and SBM_S designed for SDDMM, and STP_L and SBM_L intended for linear modules and SpMM. The functions are defined as follows:

$$\text{SBM}_S(x) = \frac{1}{1 + e^{-(\alpha(x-Th))}} \tag{1}$$

$$\text{STP}_S(x) = (x - Th) \cdot \text{SBM}_S(x) + Th \tag{2}$$

$$\text{SBM}_L(x) = \frac{1}{1 + e^{-(\alpha(x-Th))}} + \frac{1}{1 + e^{-(\alpha(-x-Th))}} \tag{3}$$

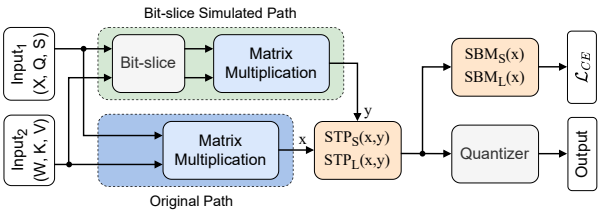

Figure 5: Overview of ViT-slice QAT process with the simulated bit-slice dot product operation and algorithms for training Th.

$$\mathrm{STP}_L(x) = x \cdot \mathrm{SBM}_L(x) \tag{4}$$

where Th denotes a trainable threshold and α is a coefficient that determines the stiffness of the function near $x = Th$.

As the both STP_S and STP_L are differentiable, the functions can be integrated with training process. However, the functions alone do not inherently drive Th to converge to a higher value, resulting in higher sparsity. Therefore, we propose a novel regularization term $\mathcal{L}_{SBM}$, which encourages the Th to be a large value, as follows:

$$\mathcal{L}_{SBM} = \lambda \sum_{j=0}^{B} \mathbb{E}\left[\sum_{i=0}^{N} \mathrm{SBM}_L(X_j^{(i)}) + \mathrm{SBM}_S(S_j)\right] \tag{5}$$

where $X^{(n)}$ represents the outputs of the n^{th} module and N determines the number of module types. S represents the output of SDDMM, and B determines the number of blocks in the model. λ is a coefficient controlling the magnitude of the regularization effects. During the optimization of the objective loss, $\mathcal{L}_{SBM}$ is added to the loss term with a cross-entropy loss.

Finally, we introduce extended QAT process to simulate bit-slice dot product operation during the trained process. Figure 5 shows the overall flow of the process with the modified STP function as follows:

$$\mathrm{STP}_S(x, y) = (x - Th) \cdot \mathrm{SBM}_S(y) + Th \tag{6}$$

$$\mathrm{STP}_L(x, y) = x \cdot \mathrm{SBM}_L(y) \tag{7}$$

The process features two branched paths: the original path and the bit-slice dot product simulated path. The bit-slice block emulates the first cycle of the bit-slice dot product operation. Since the bit-slice operation includes the round function which is not differentiable, we employ the straight-through estimator [1] to approximate the gradient. When all the techniques introduced in this section are harmonized during the QAT process, the model is trained to achieve high sparsity while experiencing a negligible accuracy drop.

4 PROPOSED VIT-SLICE ARCHITECTURE

4.1 Overview

An overview of ViT-slice is shown in Figure 6. To support bit-slice compression, a bit-slice encoder is designed (Figure 7 (a)). In the post-processing phase, the bit-slice encoder compresses the original data into MCB, a sign bit, and encoded data. At the pre-processing stage, a bit-slice decoder separates the encoded data into MLD and OLD, using MCB (Figure 7 (b)). The MLD and OLD can then be utilized for bit-slice dot product. Taking into account the sparsity of

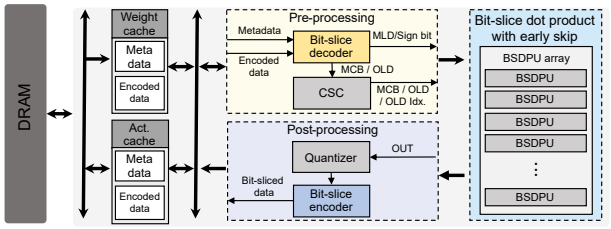

Figure 6: ViT-slice architecture overview.

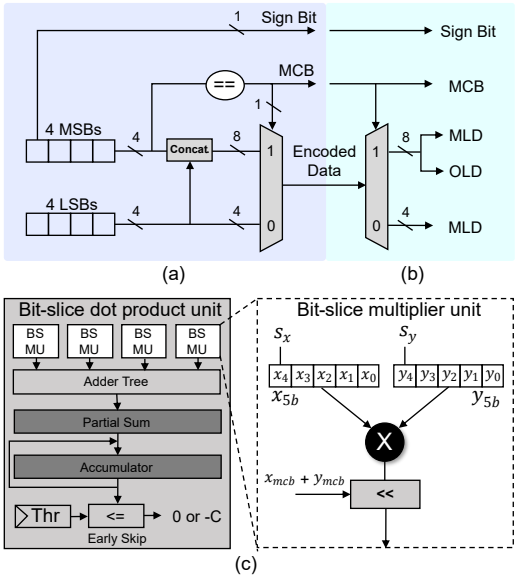

Figure 7: (a) Bit-slice encoder, (b) bit-slice decoder, and (c) bit-slice dot product unit (BSDPU).

OLD, it is stored in CSC format, enabling zero skipping. Bit-sliced data, such as MCB, a sign bit, MLD, and OLD, are not amenable to processing by general processing elements (PEs). As such, a bit-slice dot product unit (BSDPU), depicted in Figure 7 (c), is introduced. BSDPU handles bit-slice dot product with early skip, using MCB, the sign bit, MLD, OLD, index of OLD, and trained binary threshold.

4.2 Bit-slice Encoder and Decoder

The bit-slice encoder depicted in Figure 7 (a) is specifically designed to support on-chip bit-slice compression. Firstly, it operates by dividing the 8-bit original data into four MSBs and four LSBs. MCB is determined by performing an equivalent check operation on the four MSBs. The bit-slice encoder employs a 2-to-1 multiplexer (MUX) to produce the encoded data. When the MCB is 1, the MUX outputs the concatenated value of the four MSBs and four LSBs as the encoded data. On the contrary, when the MCB value is 0, the MUX outputs only the four LSBs as the encoded data. The sign bit utilizes the MSB of the original data. As a consequence, the bit-slice encoder transforms the original data into the sign bit, MCB, and encoded data.

However, the encoded data cannot be directly used for calculations because the bit-width of one data element varies depending

on whether the four MSBs are identical or different. To address the issue, a bit-slice decoder has been introduced, as shown in Figure 7 (b). When the MCB is 1, the bit-slice decoder reads 8 bits from the encoded data and stores four MSBs as MLD and four LSBs as OLD. In contrast, when the MCB value is 0, only 4 bits are read from the encoded data, and the four bits are stored as MLD. The MLD and OLD generated in the bit-slice decoder are then utilized in bit-slice dot product with early skip, in conjunction with the sign bit and MCB.

4.3 Bit-slice Dot Product Unit

We propose BSDPU, as shown in Figure 7 (c), to support bit-slice dot product with early skip. BSDPU comprises bit-slice multiplier units (BSMUs), an adder tree, an accumulator, and an early skip unit. Bit-slice multiplier unit (BSMU) takes either MLD or OLD as inputs and performs multiplication using a 5-bit multiplier. While OLD is always a positive number, the sign of MLD cannot be determined. For example, when the original data is 1111_0110_2, only the four LSBs 0110_2 are stored, because the four MSBs are the same. When the four LSBs are used as inputs to a general multiplier, the MSB of the four LSBs is used as a sign bit. Consequently, the original data 1111_0110_2, being a negative number, is used for multiplication as a positive number. To solve the problem, we separately store the MSB of the original data as a sign bit and use it for sign extension. By introducing this method, the original data 1111_0110_2 can be correctly compressed to 10110_2 and used for multiplication.

To align the digits of the four MSBs and four LSBs, we use a barrel shifter. When the four MSBs are multiplied by another four MSBs, the result requires an 8-bit left shift. On the other hand, when the four MSBs are multiplied by the four LSBs, the result necessitates a 4-bit left shift. No left shift is performed in the multiplication of the four LSBs by another four LSBs. The barrel shifter can determine the number of bits to shift at a time through an additional signal, which is the sum of the MCB of X and the MCB of Y. Finally, we implement early skip using a trained binary threshold. If the partial sum of all multiplication results between MLD exceeds the trained binary threshold, it continues to bit-slice dot product and if it does not, it outputs the result as 0 or Th.

5 EVALUATION

5.1 Experiment Settings

We evaluate our method on DeiT [13] and LeViT [4] with ImageNet [8] dataset. The weights of the models are initialized with the corresponding pre-trained weights from [15]. Subsequently, the models are fine-tuned using the proposed QAT method, along with the 8-bit LSQ quantizer [3]. For the trainable threshold Th, a learning rate of 2e-2 is applied to accelerate the convergence speed. The value of λ for the regularization term is set to 0.3 for DeiT [13] and 0.8 for LeViT [4]. Additionally, the stiffness coefficient α for the SBM functions is set to 50 for both models.

To estimate the chip area and total power consumption, the proposed design is synthesized with 32nm CMOS technology using Synopsys Design Compiler. In addition, CACTI [12] is used to estimate the energy and area of SRAMs. The BSDPU array incorporates 786 BSDPUs, with each BSDPU further composed of 4 BSMUs. ViT-slice occupies an area of 7.46mm^2 and consumes a power of 1.18W.

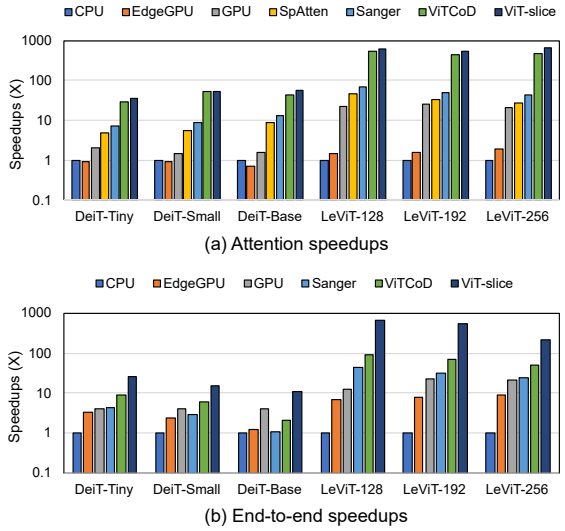

Figure 8: Speedup improvement over CPU, EdgeGPU, GPU and 3 SOTA accelerators on 6 benchmarks.

The synthesized frequency is 500MHz. For performance evaluation of the proposed design, a cycle-accurate simulator is implemented with the assumption of a HBM bandwidth of 64 GB/s.

5.2 Experimental Results

Performance Comparison. Figure 8 shows the attention and end-to-end speedup comparisons of ViT-slice and six baselines. We evaluate the speedup by adjusting the sparsity to maintain an accuracy drop within 1.5%. On average, ViT-slice achieves 328×, 215×, 28.3× attention speedup, and 244×, 35.3×, 16.8× end-to-end speedup over CPU (Intel Xeon Gold 6152), EdgeGPU (Nvidia Jetson Xavier NX), GPU (TITAN Xp). Additionally, we compare the attention and end-to-end speedup of ViT-slice with SOTA Transformer accelerators, such as SpAtten [14], Sanger [11], and ViTCoD [16]. To ensure a fair comparison, the number of processing units for SOTA accelerators is determined based on similar area constraints at the 32nm technology node. As a result, ViT-slice achieves an average attention speedup of 12.8×, 8.5×, and 1.2× over SpAtten, Sanger, and ViTCoD, respectively. Furthermore, ViT-slice outperforms Sanger and ViTCoD with 10.4× and 5.0× an average end-to-end speedup, respectively. While Sanger and ViTCoD concentrate solely on optimizing attention, leading to a limited improvement in end-to-end speedup compared to the attention speedup, ViT-slice adopts a broader approach. By optimizing not just attention but also linear modules with bit-slice compression and early skip algorithm, ViT-slice achieves a higher end-to-end speedup, outperforming both Sanger and ViTCoD.

Performance Analysis. Figure 9 provides a performance breakdown of ViT-slice in the DeiT-Tiny/Small/Base models. The baseline in the analysis is a conventional systolic array. Systolic array + Bit-slice is a framework where the bit-slice algorithms are executed in a conventional systolic array. Considering that one multiplication operation demands anywhere from one to four cycles, the increase in operations results in a reduction of the speedup to 0.78×, relative to the baseline. ViT-slice (without early skip) is a framework in

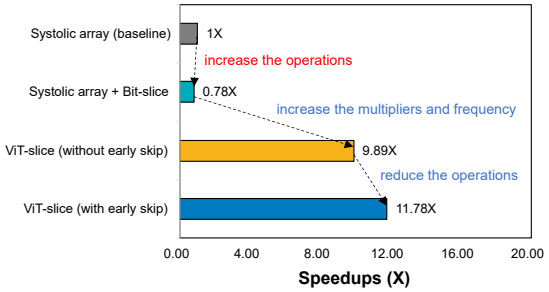

Figure 9: Breakdown of average speedup achieved by ViT-slice on DeiT-Tiny/Small/Base models.

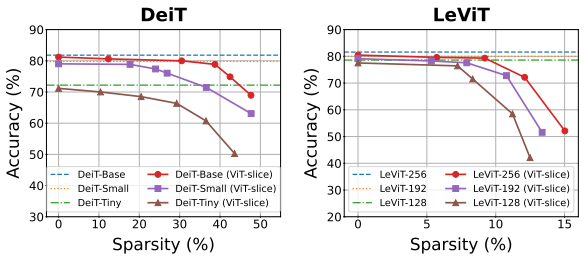

Figure 10: Trade-off curves between average sparsity and accuracy loss for DeiT-Base/Small/Tiny and LeViT-256/192/128.

which the bit-slice algorithms are executed in ViT-slice. ViT-slice employs BSDPUs facilitating efficient processing of bit-sliced data. In addition, by reducing the number of accumulators, ViT-slice achieves a 3.07× increase in the number of multipliers compared to the baseline. It also has a 1.59× higher frequency due to the usage of 5-bit multipliers. As a result, ViT-slice (without early skip) achieves a 9.89× speedup over baseline. ViT-slice (with early skip) reduces the total operation count by integrating an early skip within the bit-slice dot product process, according to the trained threshold. The result of this enhancement is a substantial 11.76× speedup when compared to the baseline.

Algorithm Analysis. Figure 10 exhibits the trade-off between the sparsity and accuracy of DeiT-Base/Small/Tiny and LeViT-256/192/128. It also includes the accuracy of the baseline models, which remain in full-precision format. For the analysis, we average the results across all modules where the proposed training algorithm is applied. To obtain models with different sparsity levels, we utilize different values of the regularization coefficient λ in the range of 0.1 to 10 during the training process. When the sparsity levels are set to 10.39% to 30.51% for DeiT models and 7.23% to 9.21% for LeViT models, we observe only a negligible accuracy drop of approximately 1.5% from the baseline accuracy. The results demonstrate that the novel training algorithm successfully allows for the reduction of model size through quantization and computational requirements through sparsity, while maintaining competitive accuracy.

6 CONCLUSION

We propose ViT-slice, a novel algorithm-architecture co-design to accelerate Vision Transformers. At the algorithm level, we cut down on memory access by eliminating unnecessary bits with bit-slice compression. Additionally, we reduce computational cost required by processing bit-sliced data efficiently with bit-slice dot product early skip. We apply the bit-slice algorithms not just to the attention module, but also to linear modules, including linear projection, linear transformation, and FFN, significantly boosting the end-to-end speedup of Vision Transformers. In hardware aspects, ViT-slice employs a dedicated BSDPU to efficiently handle bit-sliced data. BSDPU leverages 5-bit multipliers and shifter for bit-sliced data multiplication, while using an adder tree to reduce the accumulator count and increase the number of multipliers, enhancing overall throughput. ViT-slice achieves end-to-end speedup of 10.4× and 5.01× over the Sanger and ViTCoD accelerators.

ACKNOWLEDGMENTS

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT) under Grant 2022-0-00971 and 2021-0-00106; in part by the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE) of Korea and National Research Foundation (NRF) of Korea; and in part by Samsung Electronics. The EDA Tools used in this work were supported by IDEC, Daejeon, South Korea.

REFERENCES

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).
[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR* (2021).
[3] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. LEARNED STEP SIZE QUANTIZATION. In *International Conference on Learning Representations*.
[4] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, et al. 2021. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*. 12259–12269.
[5] Tae Jun Ham, Sung Jun Jung, et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 328–341.
[6] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, et al. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.
[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, et al. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.
[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).
[9] Yanjing Li, Sheng Xu, Baochang Zhang, Xianbin Cao, Peng Gao, and Guodong Guo. 2022. Q-vit: Accurate and fully quantized low-bit vision transformer. *Advances in Neural Information Processing Systems* 35 (2022), 34451–34463.
[10] Zhenhua Liu, Yunhe Wang, et al. 2021. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems* 34 (2021).
[11] Liqiang Lu, Yicheng Jin, Hangrui Bi, et al. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 977–991.
[12] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.
[13] Hugo Touvron, Matthieu Cord, Matthijs Douze, et al. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.
[14] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 97–110.
[15] Thomas Wolf, Lysandre Debut, Victor Sanh, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 38–45.
[16] Haoran You, Zhanyi Sun, et al. 2023. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 273–286.