공학박사 학위논문

DNN Accelerator and Load Balancing Techniques Tailored for Accelerating Memory-Intensive Operations

메모리 집약적 연산 가속화를 위해 맞춤화된 DNN 가속기 및 로드 밸런싱 기술

2022년 8월

서울대학교 융합과학기술대학원
융합과학부 지능형융합시스템전공
이 선 정

DNN Accelerator and Load Balancing Techniques Tailored for Accelerating Memory-Intensive Operations

지도교수 안 정 호

이 논문을 공학박사 학위논문으로 제출함

2022년 7월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

이 선 정

이선정의 공학박사 학위 논문을 인준함

2022년 7월

Abstract
DNN Accelerator and Load Balancing Techniques Tailored for Accelerating Memory-Intensive Operations

Sunjung Lee
Intelligence Systems
Department of Transdisciplinary Studies
The Graduate School
Seoul National University

Deep neural networks (DNNs) are used in various fields, such as in image classification, natural language processing, and speech recognition based on high recognition accuracy that approximates that of humans. Due to the continuous development of DNNs, a large body of accelerators have been introduced to process convolution (CONV) and general matrix multiplication (GEMM) operations, which account for the greatest level of computational demand. However, in the line of accelerator research focused on accelerating compute-intensive operations, the execution time of memory-intensive operations has increased more than it did in the past.

In convolutional neural network (CNN) inference, recent CNN models adopt depth-wise CONV (DW-CONV) and Squeeze-and-Excitation (SE) to reduce the computational costs of CONV. However, existing area-efficient CNN accelerators are sub-optimal for these latest CNN models because they were mainly optimized for compute-intensive standard CONV layers with abundant data reuse that can be pipelined with activation and normalization operations. In contrast, DW-CONV and SE are memory-intensive with limited data reuse. The latter also strongly depends on the nearby CONV layers, making an effective pipelining a daunting task. Therefore, DW-CONV and SE only occupy 10% of entire operations but become memory bandwidth bound, spending more than 60% of the processing time in systolic-array-based accelerators.

During the transformer training process, the execution times of memory-intensive operations such as softmax, layer normalization, GeLU, context, and attention layer increased because conventional accelerators improved their computational performance capabilities dramatically. In addition, with the latest trend toward increasing the sequence length, the softmax, context, and attention layers have much more of an influence as their data sizes have increased quadratically. Thus, these layers take up to 80% of the execution time.

In this thesis, we propose a CNN acceleration architecture called MVP, which efficiently processes both compute- and memory-intensive operations with a small area overhead on top of the baseline systolic-array-based architecture. We suggest a specialized vector unit tailored for processing DW-CONV, including multipliers, adder trees, and multi-banked buffers to meet the high memory bandwidth requirement. We augment the unified buffer with tiny processing elements to smoothly pipeline SE with the subsequent CONV, enabling concurrent processing of DW-CONV with standard CONV, thereby achieving the maximum utilization of arithmetic units. Our evaluation shows that MVP improves performance by 2.6× and reduces energy consumption by 47% on average for EfficientNet-B0/B4/B7, MnasNet, and MobileNet-V1/V2 with only a 9% area overhead compared to the baseline.

Then, we propose load balancing techniques that partition multiple processing element tiles inside a DNN accelerator for transformer training acceleration. Traffic shaping alleviates temporal fluctuations in the DRAM bandwidth by handling multiple processing element tiles within a cluster in a synchronous manner but running different clusters asynchronously. Resource sharing reduces the execution time of compute-intensive operations by simultaneously executing the matrix units and vector units of all clusters. Our evaluation shows that traffic shaping and resource sharing improve the performance by up to 1.27× for BERT-Large training.

keywords: DNN accelerator, tailored vector unit, CNN, load balancing, partitioning processing units, transformer, traffic shaping, resource sharing
student number: 2018-38972

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning is one of the most popular applications today, and deep neural networks (DNNs) are the most influential aspect of machine learning. Although the popularity of DNNs has been growing, they remained out of the limelight initially. Early DNNs [57] operated successfully, but the performance of the computation units at the time could not cope with the vast amounts of computation. Thus, researchers mainly used statistical learning methods [46] such as those known as regression, tree, clustering, and support vector machine, as they are associated with smaller computational loads compared to the use of a DNN.

The advent of GPUs directly addressed the greatest problem facing DNNs. With the continuous development of semiconductor processes, modern data-parallel architectures have been able to provide massive computational performance advancements within a single chip. Early DNNs undertook significant

This chapter is based on [59].
"MVP: An Efficient CNN Accelerator with Matrix, Vector, and Processing-Near-Memory Units", ©2022 by Sunjung Lee and Jaewan Choi and Wonkyung Jung and Byeongho Kim and Jaehyun Park and Hweesoo Kim and Jung Ho Ahn, is licensed under CC BY 4.0. https://doi.org/10.1145/3497745.

numbers of computations, implying that it was difficult to use them in commercial services. However, the combination of the DNN and GPU [54] accelerated the development of related algorithms. In particular, vision and natural language processing evolved considerably, and various studies [21, 22, 38, 39, 41, 42, 44, 54, 64, 73, 77, 79, 83 – 87, 89, 103] using convolution (CONV) and transformers were introduced.

As DNNs have been established as non-fungible applications, specialized DNN accelerators have emerged. Google [50] independently developed a tensor processing unit that provided performance of 92 TOPS and used it for the first time in their service. NVIDIA [12, 19, 20, 68] provided up to 800 TFLOPs of the computational performance by adding a tensor core designed for processing DNNs on the GPU. In addition, many studies [1, 2, 7, 10, 13, 14, 24 – 26, 28, 33, 36, 37, 43, 47 – 50, 56, 62, 63, 72, 78, 80, 81, 92 – 94, 99, 100, 105] have introduced hardwares for DNN acceleration. Therefore, these hardwares significantly reduced the execution time of DNNs.

In a DNN, CONV and general matrix multiplication (GEMM) account for most of the computations. Conventional DNN accelerators are designed to utilize massive parallelism and abundant data reuse by considering the characteristics of these operations. However, the direction of accelerator research has focused on compute-intensive operations, leading to several problems. For the first time, variants of CONV are not processed efficiently because the structure is optimized for the standard CONV. Second, as the execution time of compute-intensive operations decreases, the impact of memory-intensive operations, which formerly required minor amounts, increases.

In this dissertation, we propose a novel DNN accelerator and load balancing techniques to address the aforementioned challenges. We propose a customized accelerator that efficiently processes the latest convolutional neural network (CNN) models for CNN inference. We also propose load balancing techniques and supporting hardware to accelerate both compute- and memory-intensive operations in the transformer training.

1.1 Accelerating Depth-wise Convolution on Edge Device

Convolutional neural networks (CNNs) are used in a variety of applications such as image recognition and object detection. Mobile and edge devices are becoming a primary platform for CNN inference due to superior response time, security, and privacy. Major smartphone companies such as Apple, Google, and Samsung deploy CNN inference engines in their devices [6,31,80]. By 2022, 7.3 billion mobile device users are projected to have on-device artificial intelligence (AI) capability [27,82], spurring the need for efficient CNN inference accelerators.

As standard convolution (ST-CONV) layers dominated the computation of conventional CNN models, a large body of accelerator research has focused on this layer type [10,13,14,24,26,50,63,92,94,99]. Among those, accelerators supporting systolic execution or its variants [50,65] are gaining popularity due to superior performance, energy, and area efficiency. These architectures efficiently exploit massive parallelism and abundant data reuse opportunities of input feature maps (IFmaps) in computing output feature maps (OFmaps) for CONV layers. Hence, major industry players such as Google have already adopted this systolic-array-based acceleration in their production [30,50].

The CNN models have evolved to reduce the arithmetic operations while retaining or improving recognition accuracy. In the early days, designers manually determined the model configurations, such as layer types, depth (the number of layers), and kernel sizes. ST-CONV with a kernel size of 3×3 or larger was dominant. As these ST-CONV layers are computationally heavy, various CONV types have been proposed to reduce the number of operations by changing the kernel shape or reducing the number of channels being referenced [18, 84, 85].

The advent of neural architecture search (NAS [106]) has led to several changes in configuring CNNs. Early NAS studies reduced the number of operations drastically by mostly using ST-CONV with 1×1 kernel size (called point-wise CONV, PW-CONV) and depth-wise CONV (DW-CONV). However, due to excessive use of skip connection and unstructured layer configurations, the actual execution time was longer than the time predicted by the number of arithmetic operations. Therefore, the latest CNN models constructed by NAS utilize a building block (BB) composed of PW-CONV, DW-CONV, and squeeze-and-excitation (SE) as a unit since [86], in which memory-intensive operations have become prevalent.

The systolic array (SysAr) architecture has structural limitations in handling these emerging, memory-intensive CNN operations. DW-CONV only uses a single IFmap to create a single OFmap. Therefore, SysAr that feeds a massive number of MACs in its matrix unit (MU) using a limited bandwidth from the unified buffer (UB) in a wavefront manner cannot fully utilize its MACs on DW-CONV. The other accelerator [15] fully utilizes the MACs it owns by arranging those with distributed register files and hierarchical on-chip networks, although

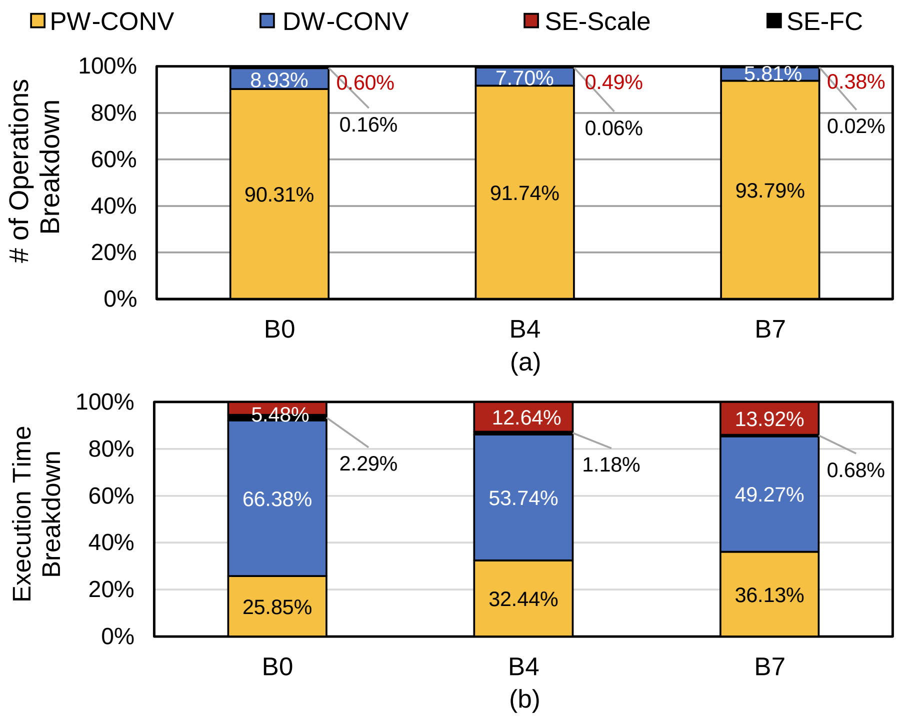


Figure 1.1: (a) The number of operations and (b) execution time breakdown of EfficientNet-B0/B4/B7 of the evaluated systolic-array-based accelerator (configurations are specified in Section 4.2.1). NORM, ACT, and POOL operations are not included as their execution time is hidden through pipelining.

at a significant cost in the area [56] and wire energy [33].

When normalization (NORM) and activation (ACT) operations are performed prior to CONV, SysAr becomes inefficient because its MU and vector unit (VU) cannot be pipelined. PW-CONV occupies more than 90% of the number of operations but takes under 40% in execution time (see Figure 1.1). Memory-intensive operations such as DW-CONV and SE-Scale (the operation of applying the squeezed weights to feature maps) take the remaining 60% of execution time, demonstrating the importance of accelerating memory-intensive operations for the latest CNNs.

In this paper, we propose an MVP architecture composed of Matrix, Vector, and Processing-near-memory units (more accurately, processing-near-on-chip-unified-memory units) to efficiently process both compute- and memory-intensive CNN operations with a small area overhead on the baseline SysAr architecture. VU of SysAr is augmented to process DW-CONV effectively by adding an array of arithmetic units, whose size is much smaller than that of MU. VU stores a tile of an OFmap produced by PW-CONV in small-but-high-throughput buffers and performs DW-CONV with the array of multipliers and adder trees, exploiting the key characteristic of DW-CONV that the reuse distance of its IFmaps is short. Processing-near-memory unit (PNMU) performs element-wise operations (NORM, ACT, and SE-Scale) that are processed immediately prior to CONV, which are performed at MU, to eliminate unnecessary on-/off-chip memory access. VU and PNMU significantly reduce the execution time of memory-intensive operations as these operations can mostly be overlapped with computation for the nearby PW-CONV layers.

1.2 Accelerating Transformer Models in Training

The transformer [91] is one of the most popular deep neural networks (DNNs) for language modeling, machine translation, question & answering, and text generation. BERT [22] and GPT [73], which are the modified versions of the vanilla transformer model, accelerate its popularity by improving recognition accuracy levels such that they are close to those of a human. However, due to continuous changes of these models to increase their accuracy, the transformer model is now associated with certain side effects such as a massive increase in

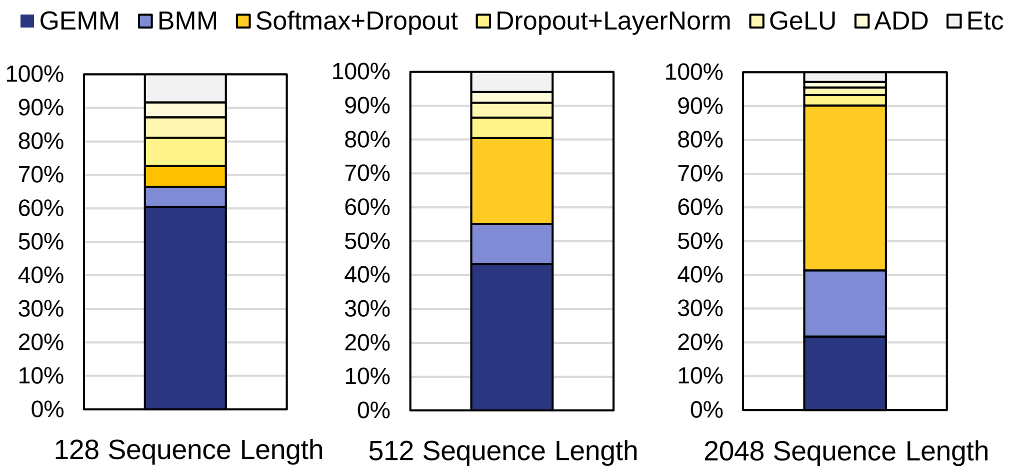


Figure 1.2: Execution time breakdown of BERT-Large training using A100. Here, 128, 512, and 2048 sequence lengths are used by BERT-Large, and Megatron-LM [64]. We obtain the results using the latest NVIDIA code [67].

the number of computations and a large model size.

To process DNN models efficiently, many DNN accelerator studies [1, 2, 7, 10, 12 – 14, 19, 20, 24 – 26, 28, 33, 36, 37, 43, 47 – 50, 56, 59, 62, 63, 68, 72, 78, 80, 81, 92 – 94, 99, 100, 105] have been introduced. As compute-intensive operations such as GEMM and convolution account for most of the computations in DNN models, conventional DNN accelerators are designed to utilize massive parallelism and abundant data reuse by considering the characteristics of these operations. However, because the direction of accelerator research is mainly focused on the compute-intensive operations, the execution time of memory-intensive operations becomes a significant factor. Figure 1.2 shows the execution time breakdown of BERT-Large training. Depending on sequence length (SL), memory-intensive operations such as softmax, layer normalization, and GeLU take up to 60% of the execution time. Also, batched matrix multiplication (BMM) with 10 times fewer operations per byte (OP/B) than GEMM occupies up to 20% of the execution time. Therefore, it is important to reduce the impact of memory-intensive operations in the transformer.

To reduce the execution time of memory-intensive operations, there have been load balancing studies [8, 51, 52] by partitioning multiple computing units within a single chip, and processing tasks in each partition asynchronously. [51] alleviates temporal fluctuations in the memory bandwidth demands due to the various computational characteristics of the layers through DRAM traffic shaping. AI-MT [8] and MAGMA [52] propose task scheduling methods for multi-tenant inference. They categorize DNN models into several dependency-free groups according to the compute- or memory-intensive characteristics. Then, they map dependency-free DNN groups on multiple accelerator cores. Lastly, they execute multiple dependency-free groups concurrently. In this way, they can reduce the number of stalls caused by the DRAM bandwidth bottleneck.

However, conventional DNN training does not process multiple tasks simultaneously in an accelerator due to several limitations. In cloud data centers, the inference uses a small number of batch sizes to comply with the quality of service (QoS) requirements. When inference task uses relatively few batch sizes, it cannot utilize all the computing units included in the single accelerator. Also, multi-tenant inference can be processed in parallel because there are no dependencies between multiple inference tasks. Therefore, previous load balancing studies could apply multi-tenant inference by allocating multiple inference tasks in a single accelerator. In contrast, the training processes only a single DNN model in the single accelerator because it uses a large batch size. Also, because the training should update weight parameters every time for every mini-batch, multiple mini-batches cannot be processed in parallel.

In this paper, we propose load balancing techniques that utilize the latest DNN training methods, in this case gradient accumulation [29], and sequence binning [67]. Gradient accumulation performs weight updates at once after collecting the gradients of several mini-batches. This relieves the dependency between mini-batches so that during training multiple tasks can be processed in parallel in an accelerator. Sequence binning allocates SL variably according to the padded tokens in the input.

We propose traffic shaping techniques using gradient accumulation. Traffic shaping partitions a single accelerator into multiple clusters and processes multiple clusters asynchronously during the gradient accumulation step. Traffic shaping reduces the execution time of memory-intensive operations by alleviating DRAM bandwidth fluctuations during the weight update. Traffic shaping realizes an additional performance improvement through sequence binning. Traffic shaping is capable of high performance improvements when tasks with different computational characteristics are assigned to each cluster. Because sequence binning provides various SLs, it contributes to a further performance improvement.

We also propose a resource sharing technique that executes the matrix units and vector units of all clusters simultaneously when the compute- and memory-intensive operations are performed concurrently on different clusters. Resource sharing reduces the execution time of the compute-intensive operations. To the best of our knowledge this paper is the first work to address training acceleration with load balancing techniques.

1.3 Research Contributions

In this dissertation, we make the following contributions:

- We categorize the layer configurations of various CNN models and identify that the latest CNN models are composed of BBs that utilize PW-CONV, DW-CONV, and SE.
- We identify that the arithmetic resource of systolic-array-based CNN accelerators is severely underutilized when processing the memory-intensive operations prevalent in the BBs of the latest CNN models.
- We propose an MVP architecture that combines Matrix, Vector, and Processing-near-memory units to efficiently process both compute- and memory-intensive operations of the latest CNN models with a small area overhead.
- MVP improves performance by 2.6× and reduces energy consumption by 47% on average over EfficientNet-B0/B4/B7, MnasNet, and MobileNet-V1/V2 with a 9% area overhead over the baseline SysAr.
- We apply load balancing techniques in transformer training acceleration using gradient accumulation which is the latest DNN training methods.
- We use sequence binning to realize additional performance improvements in the load balancing techniques.
- We reduce the execution time of compute-intensive operations through resource sharing with the simultaneous execution of matrix and vector units.
- Our load balancing techniques improve performance by up to 1.27× compared to the baseline when we train a BERT-Large model.

1.4 Outline

The organization of this dissertation is as follows. Chapter 2 describes the trends of the latest DNN models and the challenges of existing accelerator when they process the latest DNN models. Chapter 3 summarize two DNN accelerators targeting inference and training. In chapter 4, We propose a CNN accelerator tailored for processing DW-CONV, including multipliers, adder trees, and multi-banked buffered to meet the high memory bandwidth requirement. In Chapter 5, we propose tiled architecture and load balancing techniques to alleviate temporal fluctuations in DRAM bandwidth and to efficiently utilize computation units. Chapter 6 discusses the use cases of both techniques and we present the conclusions in Chapter 7.

Chapter 2

Background and Motivation

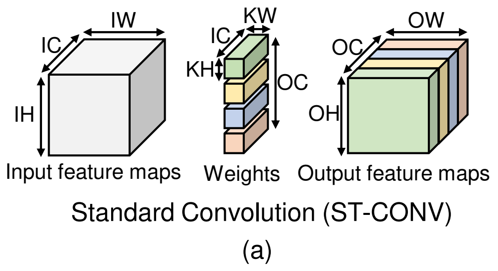
2.1 CNN background and trends

2.1.1 Various types of convolution (CONV) operations

ST-CONV is a basic type of CONV where the size of a kernel, $KH \times KW$, is typically larger than 1×1. It convolutes IFmaps and weights to produce OFmaps, requiring $KH \times KW \times IC \times OH \times OW \times OC$ operations in total (see Figure 2.1(a)). Recent CNN models adopt new types of CONV layers to reduce the number of operations, either by limiting the number of input/output channels or using smaller kernels.

Grouped CONV (GR-CONV) and depth-wise CONV (DW-CONV) reduce the number of required operations compared to ST-CONV by dividing the input channels into multiple groups of channels. We call the number of groups nGR. GR-CONV requires $KH \times KW \times (\frac{IC}{nGR}) \times OH \times OW \times (\frac{OC}{nGR}) \times nGR$ op-

This chapter is based on [59].
"MVP: An Efficient CNN Accelerator with Matrix, Vector, and Processing-Near-Memory Units", ©2022 by Sunjung Lee and Jaewan Choi and Wonkyung Jung and Byeongho Kim and Jaehyun Park and Hweesoo Kim and Jung Ho Ahn, is licensed under CC BY 4.0. https://doi.org/10.1145/3497745.



IH	IFmap Height	OH	OFmap Height
IW	IFmap Width	OW	OFmap Width
IC	# of IFmap Channels	OC	# of OFmap Channels
KH	Kernel Height	KW	Kernel Width
nGR	Number of Groups		

(b)

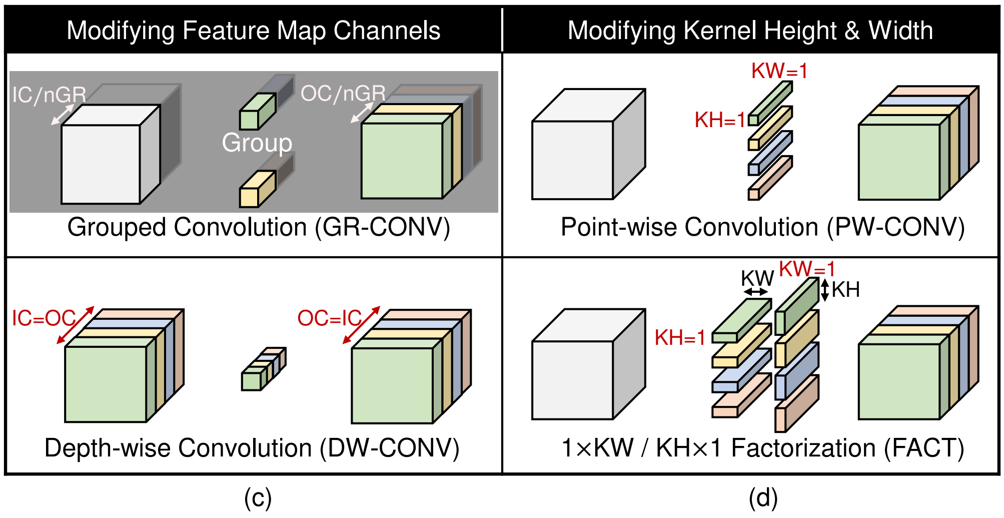

Modifying Feature Map Channels
Modifying Kernel Height & Width
IC/nGR
OC/nGR
Group
Grouped Convolution (GR-CONV)
KW=1
KH=1
Point-wise Convolution (PW-CONV)
IC=OC
OC=IC
Depth-wise Convolution (DW-CONV)
KW
KW=1
KH
KH=1
1×KW / KH×1 Factorization (FACT)
(c)
(d)

Figure 2.1: (a) ST-CONV (in general, ST-CONV includes PW-CONV, but we distinguish the two terms to emphasize whether the kernel size is 1×1 or not), (b) abbreviations for IFmaps, OFmaps and weights, (c) CONVs with different ways to refer channels, and (d) CONVs with different kernel configurations.

erations, which are reduced by a factor of nGR times compared to ST-CONV (see Figure 2.1(c)). DW-CONV is an extreme form of GR-CONV where $nGR = IC = OC$.

Point-wise CONV (PW-CONV) and Factorization (FACT) reduce the number of operations by using smaller kernels (see Figure 2.1(d)). PW-CONV exploits 1×1 kernels. FACT factorizes a $KH \times KW$ kernel into two: one with a $1 \times KH$ size and the other with a $KW \times 1$ size. Then it performs convolution for each of the two kernels, thereby reducing the total number of operations by a factor of $\frac{(KH \times KW)}{(KH+KW)}$ times.

2.1.2 Trends in CNN model architecture

Various CNN models have been proposed to increase the accuracy of image recognition and, at the same time, reduce the burden of computation (see Figure 2.2).

Handcrafted models: In the early days, many researchers tried empirical studies, tuning myriads of parameters such as layer types, number of layers, channel size, and resolution by hand to improve the accuracy of CNN models. As the number of layers increases, the accuracy tends to increase, albeit with the following issues. First, CNN models are computationally hungry. Early CNN models populated just a few layers, but later models use dozens of layers or more, requiring lots of arithmetic operations (e.g., VGGNet [79] requires about 20G operations). Therefore, GoogleNet [84], Inception-V3 [85], Xception [18], and ResNext [97] adopted PW-CONV, FACT, DW-CONV, and GR-CONV, respectively, to reduce the number of arithmetic operations. Second, there was a time when training was nearly impossible if the number of layers exceeds a certain threshold due to a vanishing gradient problem during backpropagation. ResNet [38] addressed this problem using skip connection, paving the way for stacking hundreds of layers. Batch normalization (BN) [45] and SE [42] were proposed to further increase the accuracy of CNN models by changing the entire Fmap elements using element-wise operations.

NAS models: Because manually developing CNN models requires significant engineering efforts, NAS has been proposed to design the models automatically. NAS uses machine learning to explore efficient models in performance, computation complexity, and memory capacity by using various layer types developed in the handcrafted models as candidates. In the early NAS works,

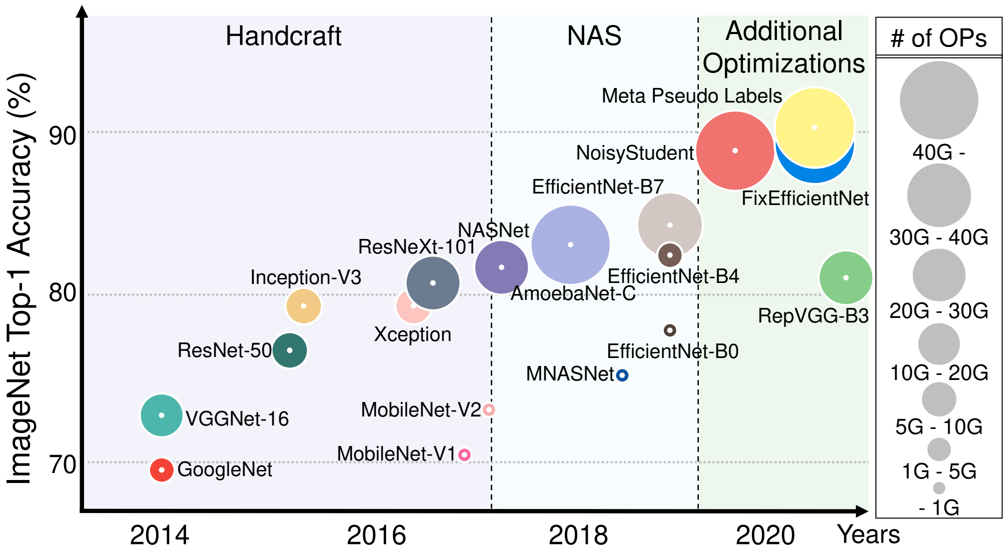


Figure 2.2: Top-1 accuracy and the number of OPs of CNN models by year. Handcraft refers to the CNN models in which various configurations are manually tuned by researchers until mid-2017. The radius of a circle is proportional to the number of MAC operations required by the models.

NASNet [106] and AmoebaNet [74] obtained high accuracy, but they required a huge amount of operations and suggested complex structures. Thus, the actual hardware execution time was much longer than the execution time predicted by the number of operations. To solve this problem, MnasNet [86] simplified the model structure using a BB, which was first used in ResNet, as a candidate for design search. EfficientNet [87] expanded MobileNet-V2 [77] and MnasNet to obtain high accuracy with fewer operations than the early NAS models.

Additional optimizations: Because EfficientNet demonstrated strength in computational complexity and accuracy to the other CNN models, subsequent studies tend to use it as a baseline model. NoisyStudent [96] increased the number of layers, channel size, and resolution of EfficientNet and changed the training method to perform pre-processing using ImageNet data and then proceed

Table 2.1: Categorization of operation types used in various CNN models. (∗ means that it is only used at the initial layer.)

	ST	PW	FACT	DW	GR	BB	SE
VGGNet [79]	✓						
GoogleNet [84]	✓	✓					
ResNet [38]	✓	✓				✓	
Inception-V3 [85]	✓	✓	✓				
Xception [18]	∗✓	✓		✓			
ResNeXt [97]	∗✓	✓			✓	✓	
MobileNet-V1 [41]	∗✓	✓		✓			
MobileNet-V2 [77]	∗✓	✓		✓		✓	
NASNet [106]	∗✓	✓		✓			
AmoebaNet [74]	∗✓	✓	✓	✓			
MnasNet [86]	∗✓	✓		✓		✓	✓
EfficientNet [87]	∗✓	✓		✓		✓	✓
NoisyStudent [96]	∗✓	✓		✓		✓	✓
FixEfficientNet [89]	∗✓	✓		✓		✓	✓
Meta Pseudo Labels [71]	∗✓	✓		✓		✓	✓
RepVGG [23]	✓						

with self-training using Instagram data. FixEfficientNet [89] improved accuracy by applying image augmentation to both training and test time using EfficientNet. Meta Pseudo Labels [71] achieved a top-1 accuracy of over 90% for the first time using EfficientNet as a baseline model. Meta Pseudo Labels used knowledge distillation [40] which trains a student network with a pre-trained teacher network. For the training, it used the ImageNet dataset as labeled data and the JFT-300M [40,83] dataset as unlabeled data. RepVGG [23] is designed to achieve high utilization on conventional accelerators or GPUs. RepVGG uses skip connection for training and excludes it for inference to simplify the model and to reduce inference time. Also, RepVGG does not use DW-CONV and

PW-CONV which undergo low utilization; instead it only uses ST-CONV like VGGNet. Although RepVGG is suitable for conventional hardware, it has lower top-1 accuracy compared to the EfficientNet variants.

Even though the components of the CNN models have been changed over time, there is little change in the fundamental structure and the types of CONV in CNN models since MnasNet. CNN models mainly use fixed compositions consisting of repeated BBs, including PW-CONV, DW-CONV, and SE (Table 2.1). PW-CONV and DW-CONV are popular as they are the most efficient to reduce the number of operations. BB is used to simplify the structure of the models. SE, like BN, increases accuracy by considering the degree of influence between Fmaps with a small computational cost. In contrast, ST-CONV is used only as the initial layer in the latest CNN models due to its high computational cost. GR-CONV does not appear in the latest CNN models, and FACT is rarely used even if they are candidates of NAS.

2.1.3 EfficientNet: A state-of-the-art CNN model

EfficientNet [87] is a representative of more recent CNN models that use multiple BBs to achieve high accuracy with a relatively low computational cost. EfficientNet is an expanded structure of MobileNet-V2 and MnasNet that scales all three parameters (resolution, the number of channels, and the number of layers) concurrently as opposed to previous works [38, 41, 84] which scale only subsets of the three dimensions. EfficientNet introduces eight exemplar models (EfficientNet-B0∼B7), which require 4∼20× lower computational costs compared to the previous CNN models with similar recognition accuracy. For example, both EfficientNet-B0 and ResNet-50 achieve nearly 77% Top-1 accu-

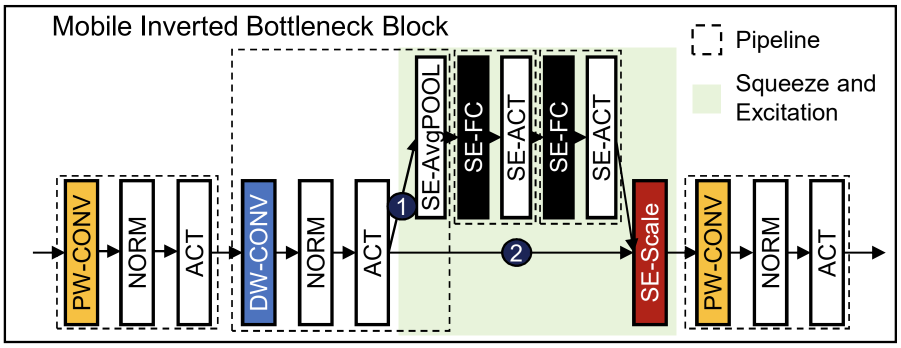


Figure 2.3: Mobile inverted bottleneck block (MBConv) of EfficientNet.

racy, but they require 390M and 4.1G operations, respectively.

EfficientNet is a sequence of mobile inverted BBs (MBConvs). MBConv consists of the first PW-CONV followed by DW-CONV, SE, and the second PW-CONV (see Figure 2.3), where each CONV has trailing NORM and ACT layers. SE includes global average pooling (SE-AvgPOOL), fully connected layer (SE-FC), activation (SE-ACT), and element-wise multiplication (SE-Scale). SE first calculates scale factors for each channel using OFmap created by DW-CONV and then operates SE-Scale using the scale factors and the DW-CONV. There is a data dependency between DW-CONV and SE because each step uses the whole OFmaps directly. In step ①, SE-AvgPOOL needs all of the Fmap elements to compute the average, and SE-FC needs all of the channels to get scale factors per channel. In step ②, SE-Scale multiplies the scale factors with OFmaps of DW-CONV which is stored at UB or DRAM because all the Fmaps were used in the previous step. Last, the second PW-CONV follows SE-Scale.

The compositional characteristics of MBConv make the arithmetic intensity of EfficientNet fluctuate a lot. Figure 2.4 shows operations per byte (OP/B) over

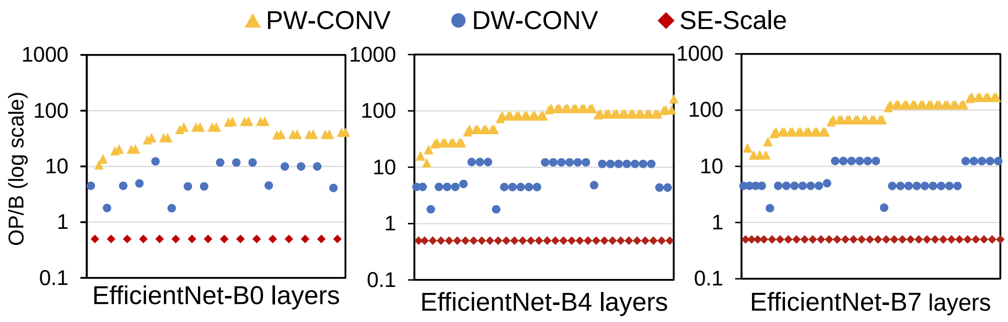


Figure 2.4: Operations per byte (OP/B) of EfficientNet-B0/B4/B7 layers. The leftmost column represents the layer processing an input image. We only show PW-CONV, DW-CONV, and SE-Scale because they occupy most operations of the models. It is assumed that all three operations use 8-bit data type.

the layers in EfficientNet-B0/B4/B7. The leftmost column in each graph represents the layer processing an input image. PW-CONV reuses an IFmap by OC times. Because EfficientNet populates more channels as it comes closer to the classification layer, OP/B increases from left to right on the graphs. DW-CONV produces an OFmap by using a single IFmap; therefore, EfficientNet's kernel sizes determine the degree of data reuse. EfficientNet uses 3×3 and 5×5 kernels. Similar to NORM and ACT (not shown in Figure 2.4), SE-Scale achieves a low OP/B due to the lack of Fmap data reuse in conducting element-wise operations. Even if the number of layers and functions differs by the network sizes (B0 being the smallest and B7 the largest), because OP/B fluctuates a lot and DW-CONV and element-wise operations appear frequently, CNN accelerators must be designed to perform both compute- and memory-intensive layers/functions equally well.

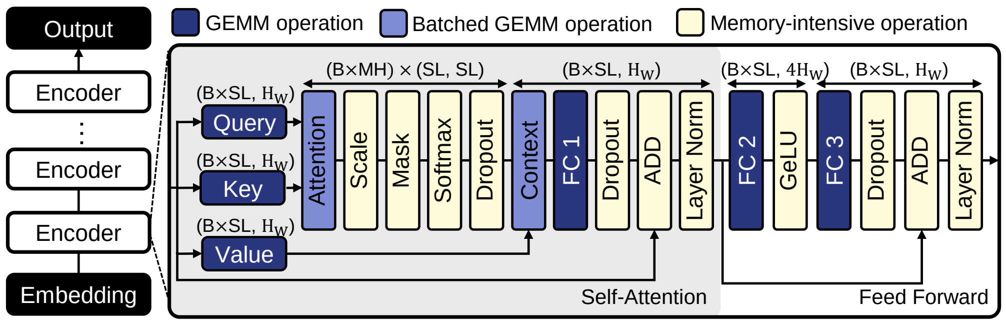


Figure 2.5: BERT model structure. Self-attention and feed forward represent gray and white ranges within an encoder. B, SL, H, and MH refer to batch, sequence length, hidden, and multi-head, respectively. Equations such as (B×SL, H_W), (B×MH)×(SL, SL) indicate the output size.

2.2 Transformer background and trends

2.2.1 Bidirectional encoder representations from transformers (BERT)

BERT [22] consists of an embedding layer, 24 encoder layers, and an output layer (see Figure 2.5). The encoder includes a self-attention layer and a feed-forward layer. The self-attention layer includes GEMM (query, key, value, and FC 1 layers), batched GEMM (BMM) (attention, and context layers), and memory-intensive (scale, mask, softmax, dropout, add, and layer normalization layers) operations. The GEMM operation has an OP/B higher than 500 due to a massive data reusability. The BMM operation is less than 100 OP/B because (B x multi-head (MH)) GEMM operations are performed independently. BMM operations can degrade the performance due to a DRAM bandwidth bottleneck when using an accelerator with high computation performance. Because memory-intensive operations have less than 10 OP/B, a bandwidth bottleneck always occurs in all memory hierarchies.

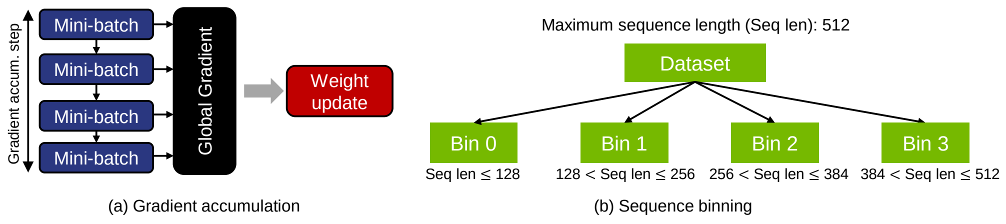


Figure 2.6: (a) Gradient accumulation. (b) Sequence binning.

The proportion of the execution time for GEMM, BMM, and memory-intensive operations varies depending on the SL size. The layers between the attention and context layers change the output size quadratically according to the SL size. In contrast, the remaining layers increase linearly. Therefore, as shown in Figure 1.1, when the SL is small, the execution time for GEMM operations takes up a large portion. When the SL is not small, the execution time for BMM and memory-intensive operations takes up a large portion.

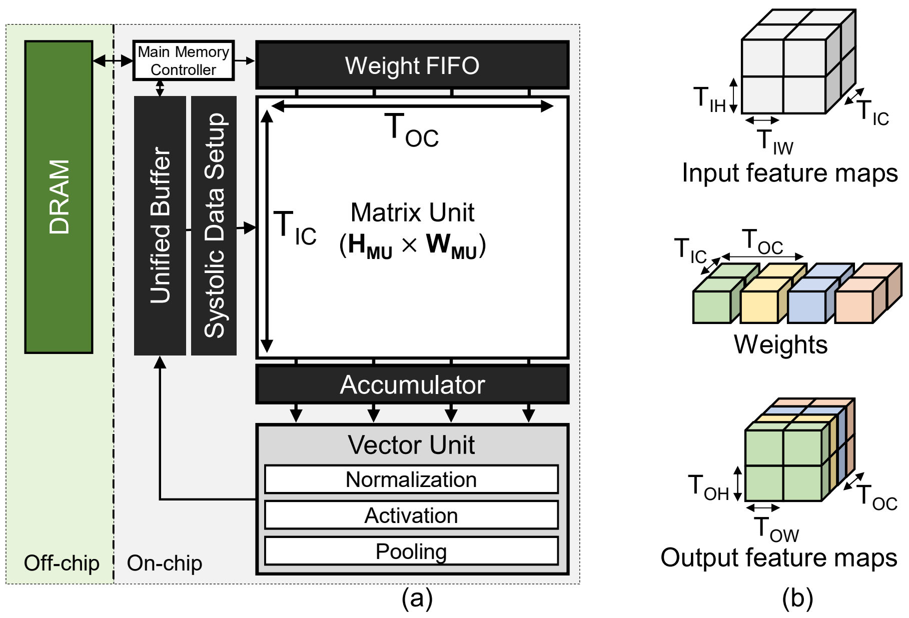
2.2.2 Trends in training transformer models

Training refers to the process of learning optimal weight parameters using a large dataset. The training consists of multiple pairs of a forward and a backward pass. A forward pass infers results by executing multiple layers with input data. During the forward pass, intermediate values are stored in DRAM, as intermediate values are used during the backward pass. After the forward pass, the classification layer calculates the loss. The backward pass is the step in which the weight parameters are redesigned using the intermediate values and loss. Training is conducted in mini-batch unit, referring to a pair consisting of one forward pass and one backward pass. The weight parameter is updated for each mini-batch.

Due to continuous changes such as increases in the model size and SL [11, 64, 73, 103] in the transformer, the transformer adopts new training methods.

Gradient accumulation: gradient accumulation [29] is proposed to solve the DRAM capacity problem of DNN accelerators. Due to increases in the model size and SL size, the DRAM capacity for intermediate values and weight parameters increases. Due to the limited DRAM capacity, the accelerators use small batch sizes for weight updates. Gradient accumulation increases the total batch size by performing multiple forward and backward passes during the gradient accumulation step and updating the weights at once as shown in Figure 2.6(a). The total batch size is the unit focused on during a single weight update. The mini-batch size is a unit that can be allocated within the DRAM capacity. The gradient accumulation step is calculated by dividing the total batch size by the mini-batch size. Gradient accumulation has the same effect found when multiple GPUs undertake data parallelism in a distributed learning [32].

Sequence binning: sequence binning [67] is proposed to eliminate redundant computations on padded tokens. In conventional training, if the input is less than the maximum SL, padded tokens are added to the input up to maximum SL. Training is then conducted using the maximum SL. However, sequence binning does not require the addition of padded tokens to the input, instead distinguishing multiple inputs into multiple bins during data pre-processing step. For example, if the maximum SL is 512 and there are four bins, the total dataset is distinguished into a unit of 128 (see Figure 2.6(b)). Bin 0 stores less than 128 SL, and bin 1 stores SLs between 128 and 256. The other SLs are stored in the same way. After data pre-processing is finished, training is performed using various SLs.



T_{IH}	Tiled IH	T_{IW}	Tiled IW	T_{IC}	Tiled IC
T_{OH}	Tiled OH	T_{OW}	Tiled OW	T_{OC}	Tiled OC

(c)

Figure 2.7: (a) Baseline systolic-array architecture (SysAr). When SysAr computes a CONV layer, IC and OC are mapped to each row and column of MU [75, 76] without using im2col [16]. (b) Tiling features of ST/PW-CONV in SysAr. Tiling notations are listed in (c).

2.3 Baseline DNN acceleration architecture

TPU [50] and NVDLA [65] adopt systolic execution or its variants, achieving superior performance and energy efficiency than other architecture types [70] in performing CNN inference. SysAr fully utilizes the data reuse characteristics of CONV by operating $\mathcal{O}(N^2)$ MACs with $\mathcal{O}(N)$ on-chip memory bandwidth from UB in a systolic data flow manner (we assume the height (H_{MU}) and width (W_{MU}) of MU are both N). Because NVDLA similarly achieves high area and energy efficiency by multicasting IFmap elements to MACs, our proposed architecture can also be applied to NVDLA. In this paper, we use SysAr as a baseline.

SysAr consists of MU, UB, systolic data setup, weight FIFO, accumulator (ACC), main-memory controller, and VU that includes NORM/ACT/POOL units (see Figure 2.7(a)). MU is a systolic array, having MACs arranged in a two-dimensional form. We assume the weights and feature maps are 8-bit data. UB is an on-chip buffer that stores IFmaps and OFmaps. Through the systolic data setup, UB sends data to MU in a diagonal wavefront manner and saves the intermediate results. Weight FIFO temporarily stores a part of weights transferred from DRAM and sends them to the registers inside MU. ACC accumulates the partial sums calculated from MU and stores them inside the buffer until partial sums become complete OFmap elements. NORM/ACT/POOL units perform element-wise operations on the 32-bit OFmap elements in a pipelined manner; the OFmap elements are converted into 8-bit values and transferred to UB.

Although MU targets GEMM operations, it also provides specialized hardware [75], algorithm [76], and instruction set architecture [50] for CONV. Therefore, SysAr can perform CONV without converting a 3D Fmap into a 2D one (im2col). Figure 2.7(b) shows the tiles of Fmaps and weights and their dimensions. Each row of MU takes one of T_{IC} IFmap elements, while each column takes one of T_{OC} weights. Each IFmap element propagates sequentially from left to right, and the output, which is a partial sum, propagates from top to bottom. Here we call each column a SysAr lane.

The Fmaps are divided into tiles if they do not fit in the on-chip storage (UB or ACC). ACC limits the size of an output tile ($T_{OW} \times T_{OH}$) to the capacity of

ACC divided by T_{OC}. Data traversal order is determined to maximize the data reuse of Fmaps and weights. Because the datapath from ACC to UB is uni-directional, each partial sum inside ACC must be a complete OFmap element. Thus, SysAr traverses IFmap and weights in all IC-, KW-, and KH-directions first.

2.4 Motivation

2.4.1 Challenges of computing memory-intensive CNN layers

Conventional CNN accelerators focus on compute-intensive ST-CONV and PW-CONV operations. They consider only limited types of memory-intensive operations that immediately follow CONV, such as NORM, ACT, and POOL. As opposed to ST-CONV and PW-CONV, DW-CONV does not reuse IFmap between OCs. Therefore, DW-CONV hardly exploits the broadcasting and systolic execution techniques mainly for IFmap reuse in conventional CNN accelerators. This characteristic of DW-CONV makes MACs in MU underutilized or requires more on-chip memory bandwidth to fully utilize MACs.

Figure 2.8(a) shows MACs in use and on-chip memory bandwidth usage when performing ST-CONV or PW-CONV in an $N \times N$ SysAr. At each cycle, the MACs in the leftmost column of MU takes N IFmap elements transferred from the on-chip memory. The IFmap elements shift from left to right at each cycle, so each element is reused by N weights. In a steady state where IFmap elements are distributed to all the MACs, N B/cycle on-chip bandwidth is enough to utilize all the N^2 MACs. In contrast, DW-CONV requires each column of MACs to take IFmap elements from different OCs (see Figure 2.8(b)), not

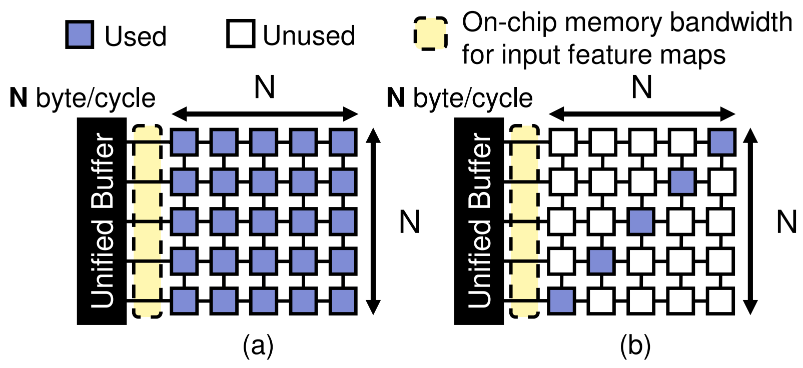


Figure 2.8: MAC and on-chip memory bandwidth used inside of MU when N×N SysAr computes CONV (assuming that IC and OC are bigger than N): the cases of (a) executing ST-CONV or PW-CONV and (b) executing DW-CONV.

taking advantage of multicasting in systolic execution. Therefore, even if DW-CONV has a significantly smaller computational load compared to ST-CONV and PW-CONV, it takes a large portion of execution time due to low MAC utilization. SysAr can only supply the on-chip bandwidth of N B/cycle even when processing DW-CONV. To fully utilize the $N \times N$ MACs, DW-CONV requires $\mathcal{O}(N^2)$ of on-chip memory bandwidth, one of the most expensive resources in conventional CNN accelerators.[1]

By contrast to SysAr, Eyeriss-v2 [15] adopts a hierarchical mesh structure in its memory system, having its on-chip memory bandwidth high enough to fulfill the bandwidth requirement of DW-CONV. This hierarchical structure exploits distributed SRAM for global buffer where each bank of the global buffer can transfer data simultaneously and connects several global buffer banks to local

[1]The structural limitation of Edge TPU also makes it suffer from this problem, resulting in a much larger execution time than predicted by the number of operations [34]. To resolve this issue, Edge TPU replaces DW-CONV with more hardware-friendly ST-CONV at a significantly increased computational cost.

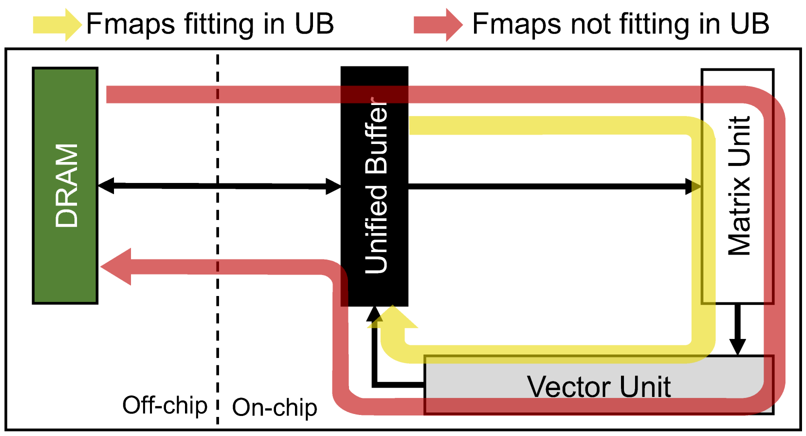


Figure 2.9: Data movement patterns inside SysAr when performing SE-Scale. Fmaps can be read from either UB or DRAM, depending on their size. We focus on only Fmaps because they are much larger than the scale factors.

SRAM inside PEs in an all-to-all manner. However, a major drawback of this design is the large area overhead as it needs to provide local SRAM for all the three data types: weights, IFmaps, and partial sum. As mentioned in [56], assuming an equal number of PEs, Eyeriss-v2 is more than twice as large as SysAr.

Conventional accelerators hide the execution time of NORM, ACT, and POOL by executing them within VU in a pipelined manner when they follow CONV. However, as opposed to these memory-intensive operations, such a pipelined execution is hardly applicable to SE-Scale that follows CONV because of the data dependency between SE-Scale and CONV (see Figure 2.3). SE-Scale requires a complete IFmap from the previous CONV to produce one single OFmap element.

Figure 2.9 shows the data movement patterns inside SysAr when performing SE-Scale. Depending on the size of Fmaps and UB, data should be read from either UB (yellow arrow) or DRAM (red arrow). We only depict the movement of Fmaps because Fmaps are much larger than the scale factors, whose number is the same as the number of channels.

When Fmaps fits in UB, SE-Scale performs multiplication after reading the Fmaps stored in UB. Because the datapath from UB to VU is uni-directional, Fmaps must go through MU to reach VU, and MU does not operate in the meantime, leading to severe underutilization of the MACs. Also, although SE-Scale requires much smaller computation than that of PW-CONV (150$\sim$250$\times$ in [87]), the execution time of SE-Scale is high because VU has $N\times$ fewer MACs than MU (see Figure 1.1). Even if MU handles SE-Scale to utilize its MACs, it takes the same time as in VU because the bandwidth of UB is limited to $\mathcal{O}(N)$. When Fmaps do not fit into UB, off-chip memory bandwidth becomes a performance bottleneck because of the extra access to more bandwidth-hungry DRAM.

2.4.2 Opportunity for load balancing in BERT training

The load balancing technique minimizes resource under-utilization by processing multiple DNN models in parallel within an accelerator. When the accelerator executes all PE tiles synchronously, compute-intensive operations achieve a high MU utilization rate, but utilize little DRAM bandwidth. In contrast, memory-intensive operations only utilize the VU and DRAM bandwidth. However, the load balancing technique obtains a benefit in terms of the execution time by properly distributing MU, VU, and DRAM bandwidth resources.

Figure 2.10 explains the impact of the load balancing technique with an example in which the memory bandwidth demands from two clusters vary

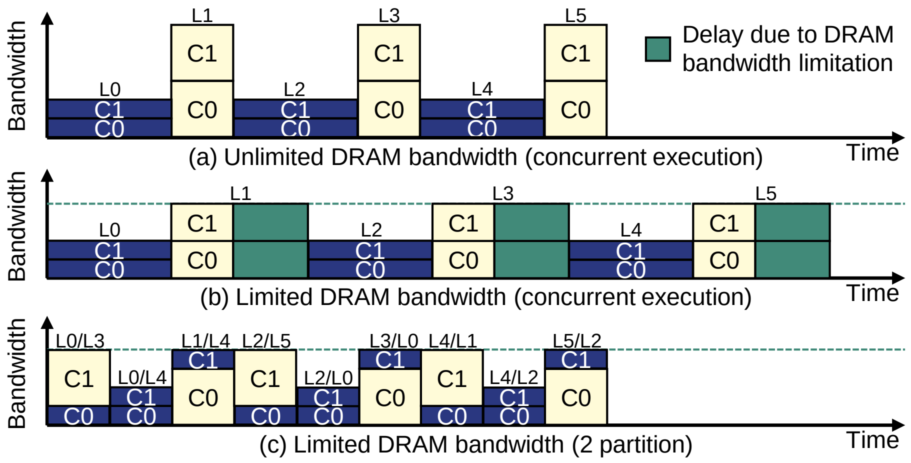
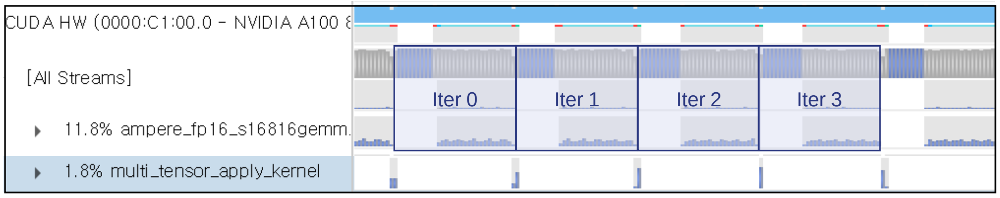
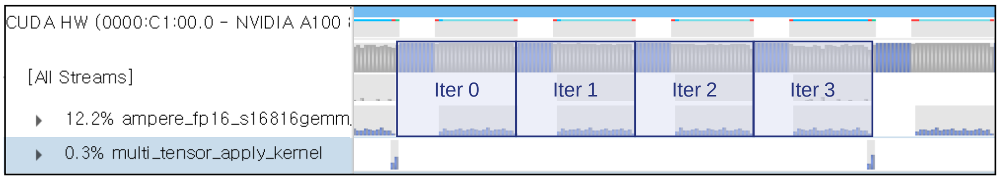

Bandwidth
L0 C1 C0
L1 C1 C0
L2 C1 C0
L3 C1 C0
L4 C1 C0
L5 C1 C0
🟩 Delay due to DRAM bandwidth limitation
Time
(a) Unlimited DRAM bandwidth (concurrent execution)
(b) Limited DRAM bandwidth (concurrent execution)
L0/L3 L0/L4 L1/L4 L2/L5 L2/L0 L3/L0 L4/L1 L4/L2 L5/L2
(c) Limited DRAM bandwidth (2 partition)

Figure 2.10: Example of a memory traffic distribution over time [51]: (a) concurrent execution with an unlimited DRAM bandwidth, (b) concurrent execution with a limited DRAM bandwidth and (c) partitioning of PE tiles into two clusters (C0 and C1). The blue layers (L0, L2, L4) are GEMM operations, and the yellow layers (L1, L3, L5) are memory-intensive operations.

depending on the layers. When the DRAM bandwidth is infinite (see Figure 2.10(a)), the execution time for GEMM and memory-intensive operations is determined by the MU and VU performance capabilities, respectively. For realistic systems with a limited bandwidth, however, memory-intensive operations are associated with a DRAM bandwidth bottleneck and there are additional delays at L1, L3, and L5 (see green squares in Figure 2.10(b)). When the accelerator is partitioned into two clusters and processes each cluster independently, the GEMM and memory-intensive operations are performed simultaneously (see Figure 2.10(c)). Thus, the load balancing technique can distribute the memory bandwidth demand, which improves the execution time.

However, the load balancing technique cannot be used during conventional training. To apply the load balancing technique, each task must be able to operate independently. During the training processes, only a single DNN model


CUDA HW (0000:C1:00.0 - NVIDIA A100
[All Streams]
11.8% ampere_fp16_s16816gemm
1.8% multi_tensor_apply_kernel
Iter 0
Iter 1
Iter 2
Iter 3
(a) Conventional training


CUDA HW (0000:C1:00.0 - NVIDIA A100
[All Streams]
12.2% ampere_fp16_s16816gemm
0.3% multi_tensor_apply_kernel
Iter 0
Iter 1
Iter 2
Iter 3
(b) Training with gradient accumulation

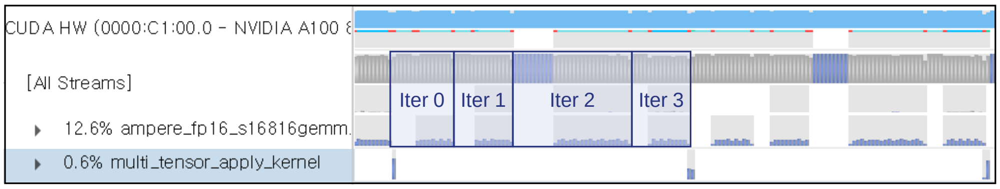

CUDA HW (0000:C1:00.0 - NVIDIA A100
[All Streams]
12.6% ampere_fp16_s16816gemm
0.6% multi_tensor_apply_kernel
Iter 0
Iter 1
Iter 2
Iter 3
(c) Training with gradient accumulation and sequence binning

Figure 2.11: Execution time of various training methods using the NVIDIA Nsight system profiler [66]: (a) conventional training, (b) training with gradient accumulation, and (c) training with gradient accumulation and sequence binning. We set the maximum SL size to 128 and used four gradient accumulation steps. The Nsight system uses *multi_tensor_apply_kernel* as the weight update kernel.

is utilized in a single DNN accelerator given the large batch size. Also, because training should update the weight parameters every time for each mini-batch, multiple mini-batches cannot be processed in parallel.

With gradient accumulation, the latest training technique becomes feasible for the application of load balancing techniques. Figure 2.11 shows the results of the NVIDIA Nsight systems profiler [66] with various training methods. Conventional training performs weight updates at every iteration as shown in

Figure 2.11(a). To obtain the weight parameters for iteration 1, training for iteration 0 must be finished. Thus, there is a dependency between iterations. Gradient accumulation eliminates the dependency between iterations within a gradient accumulation step. In Figure 2.11(b), iterations 0, 1, 2, and 3 use the same weight parameters, and four iterations can be processed in parallel.

Sequence binning varies the size of the SLs. Figure 2.11(c) depicts training with gradient accumulation and sequence binning. Iterations 0, 1, and 3 are performed with 64 SL because the padded token occupies half of the input. In contrast, iteration 2 consists of full text and is performed with 128 SL.

The load balancing technique can improve the performance when operations with opposite characteristics are performed concurrently in clusters. Therefore, sequence binning helps to obtain an additional performance improvement of our idea.

Chapter 3

DNN accelerator tailored for accelerating memory-intensive operations

We propose DNN accelerators tailored for accelerating memory-intensive operations based on a systolic array (see Figure 3.1(a)). Although the systolic array is the most known structure for accelerating DNN, this structure does not perfectly cover the trend of the latest DNN. We analyze the computational characteristics of inference and training of the latest DNN in detail. Through the analysis, we confirm that it is difficult to design an integrated structure for optimizing both inference and training because each process has different computational characteristics. Therefore, we design separate architectures for each process as shown in Figure 3.1(b) and (c).

A DNN accelerator that is specialized for inference fully utilizes the pipelined manner between layers to reduce off-chip memory access. The inference only performs a forward pass using a pre-trained model. During inference, when one layer is executed, the intermediate result of one layer is used to process the next layer. The intermediate result is used only once as the input of the next layer, and it is not required after processing the next layer. In most cases, the intermediate result is not stored in on-chip memory because this size is large.

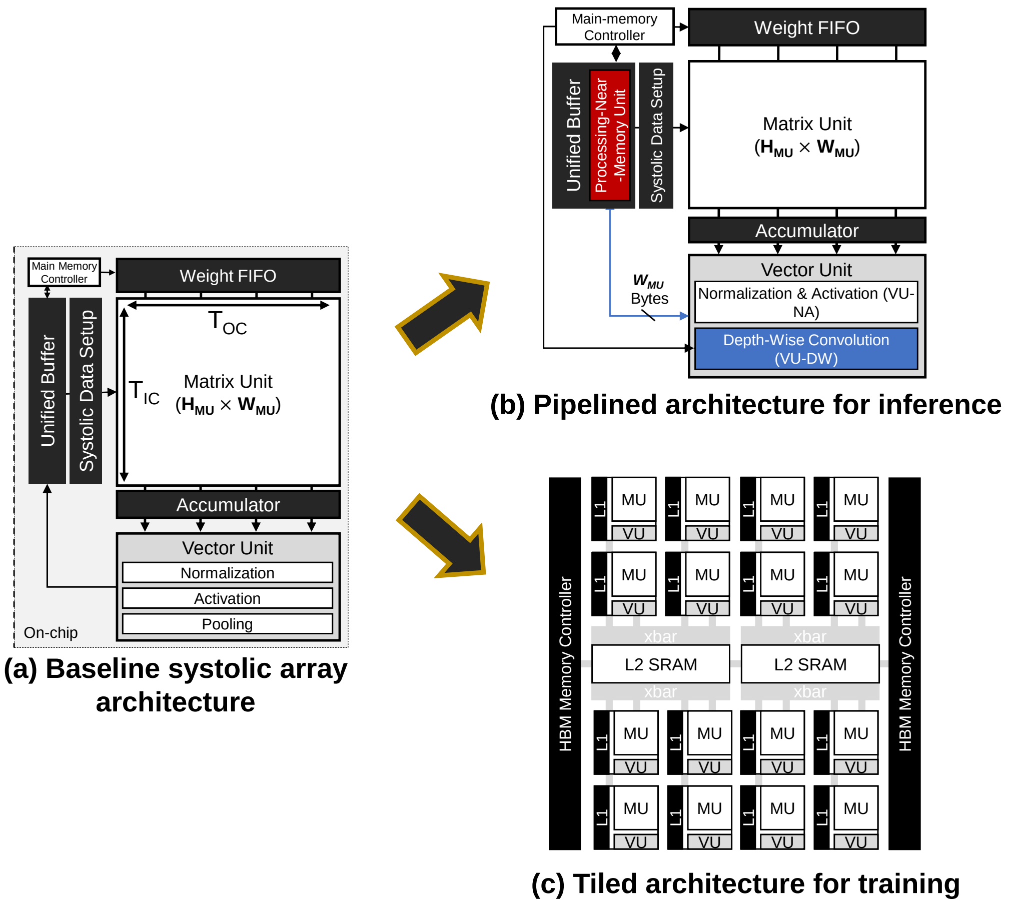


Figure 3.1: (a) Baseline systolic array architecture. (b) A pipelined architecture tailored for accelerating inference. (c) A tiled architecture tailored for accelerating training.

If a specific layer is a memory-intensive operation, off-chip memory access causes a bandwidth bottleneck and degrades the performance. Thus, reusing the intermediate result as much as possible on-chip has a major impact on the performance of the accelerator. To maximally utilize the inter-layer pipeline execution, it is necessary to establish which unit processes the proper operations considering computational characteristics. By reflecting the trend of the latest CNN, we assign operations to units in a slightly different position from that of the existing accelerators.

A DNN accelerator that is specialized for training utilizes computational parallelism to process the massive number of operations. Training performs both forward and backward passes to learn the model. In the forward pass, the intermediate result of one layer enters the input of the next layer and is used to obtain the weight gradient in the backward pass. Because the backward pass is performed after all of the layers of the forward pass are completed, the intermediate result cannot be stored in the on-chip memory. Therefore, the intermediate result is stored in off-chip DRAM. To minimize off-chip memory access, we propose a tiled architecture with multiple levels of memory because on-chip cache help to reuse input and weight data. Unlike inference, training lacks optimization methods to reduce off-chip access. To overcome these, we further propose a load balancing technique that optimizes memory-intensive operation when performing training using a tiled architecture. The details of the structure and operation of the two accelerators are covered in Chapters 4 and 5.

Chapter 4

MVP: A CNN accelerator with <u>M</u>atrix, <u>V</u>ector, and <u>P</u>rocessing-near-memory units

4.1 Contribution

4.1.1 MVP organization

We propose a novel CNN accelerator architecture called MVP, which includes an MU, a VU for NORM/ACT (VU-NA) and for DW-CONV (VU-DW), and a processing-near-memory unit (PNMU) as shown in Figure 4.1(a). As opposed to the baseline SysAr architecture that suffers from poor performance in memory-intensive layers such as DW-CONV and SE, MVP is pragmatically designed to efficiently process both compute-intensive and memory-intensive CNN layers and functions with a minimal area overhead.

VU-DW takes the OFmaps from MU through VU-NA (processing PW-CONV, NORM, and ACT) and performs DW-CONV in a pipelined manner (see Figure 4.1(b)). In case DW-CONV does not follow PW-CONV, the OFmaps bypass VU-DW and head to UB directly. Based on the key observation that a SysAr lane only processes an output channel, we place a depth-wise

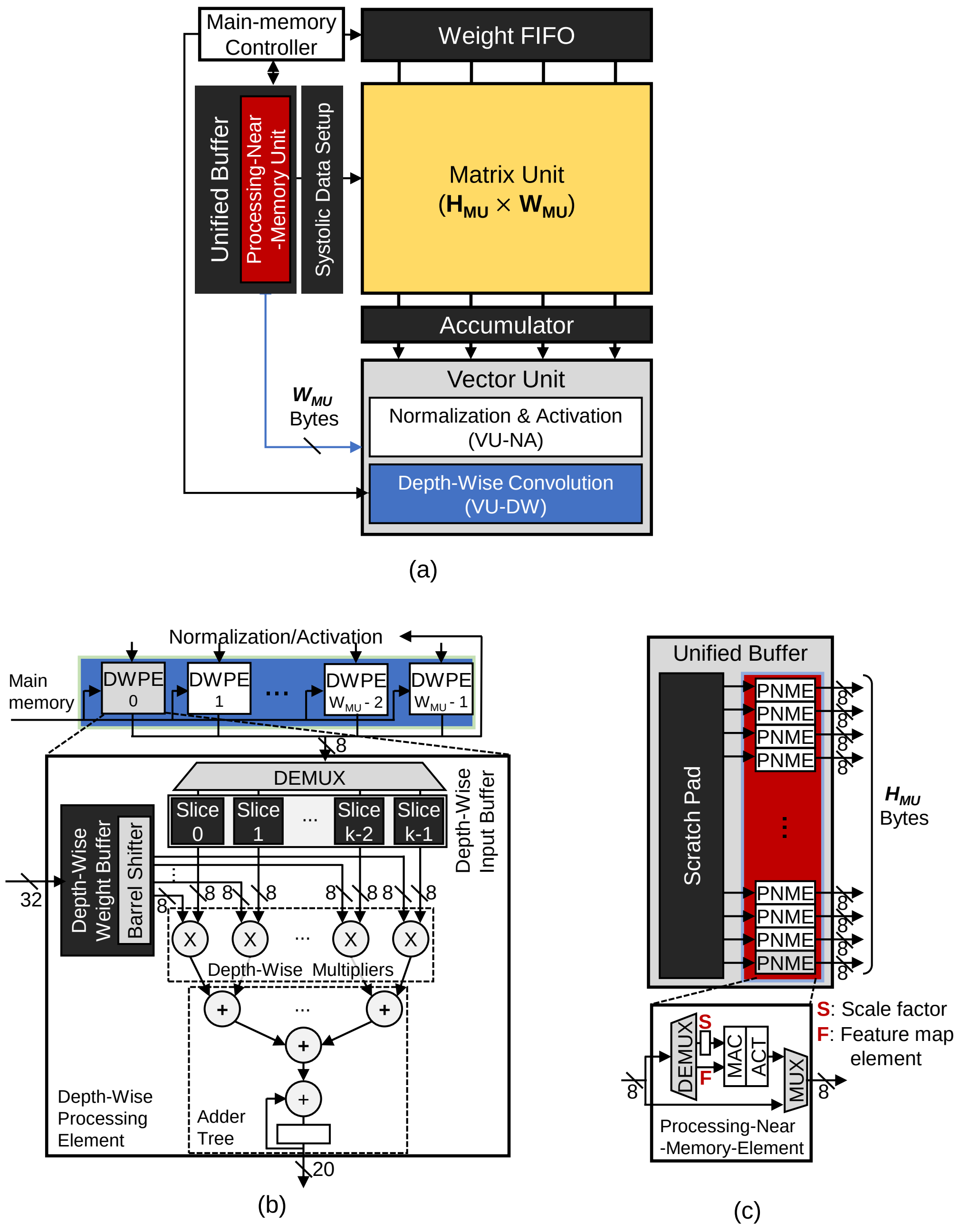


Figure 4.1: (a) MVP architecture (the red and blue parts represent the units added to the baseline SysAr), (b) VU-DW, and (c) PNMU.

processing element (DWPE) per SysAr lane. Each DWPE consists of depth-wise input/weight buffer (DWIB/DWWB), k depth-wise multipliers (DWMULs), an adder tree for partial reduction, and registers for accumulation. Each DWMUL takes the operands from DWIB and DWWB, multiplies them, and sends the output to the adder tree. DWIB is a buffer for IFmaps in DW-CONV; it consists of k DWIB slices, each being mapped to a DWMUL. Each DWIB slice is a register file with a word size of 1-byte, allowing the DWMULs to access a row of DWIB SRAM cells simultaneously. DWWB is a buffer for weights in DW-CONV. Because the weights are reused with repeating but shuffled patterns to be aligned with the IFmap values to be multiplied at DWMUL, DWWB consists of a barrel shifter and a small register file. VU and UB are connected bidirectionally. VU-DW also covers the POOL (pooling) function because recent CNN models have relatively simple POOL operations such as global average pooling.

PNMU (Figure 4.1(c)) is a preprocessing unit which is placed near on-chip unified memory to execute simple element-wise operations conducted prior to CONV. These operations are pipelined with the subsequent CONV, thereby hiding the execution time and saving memory accesses. PNMU includes H_{MU} PNM elements (PNMEs), each placed between a row of UB and MU in an aligned manner. Each PNME consists of a DEMUX, a MAC, a comparator for ReLU, a MUX, and tiny registers holding scale factors used in scaling operations such as SE-Scale and NORM. DEMUX selects either a scale factor or an IFmap element. PNME only supports ReLU because it is a popular ACT function that is also used for MBConv [39, 44], and the logics for other ACT types take up a large area. MUX decides whether or not to use preprocessing.

4.1.2 How depth-wise processing element (DWPE) operates

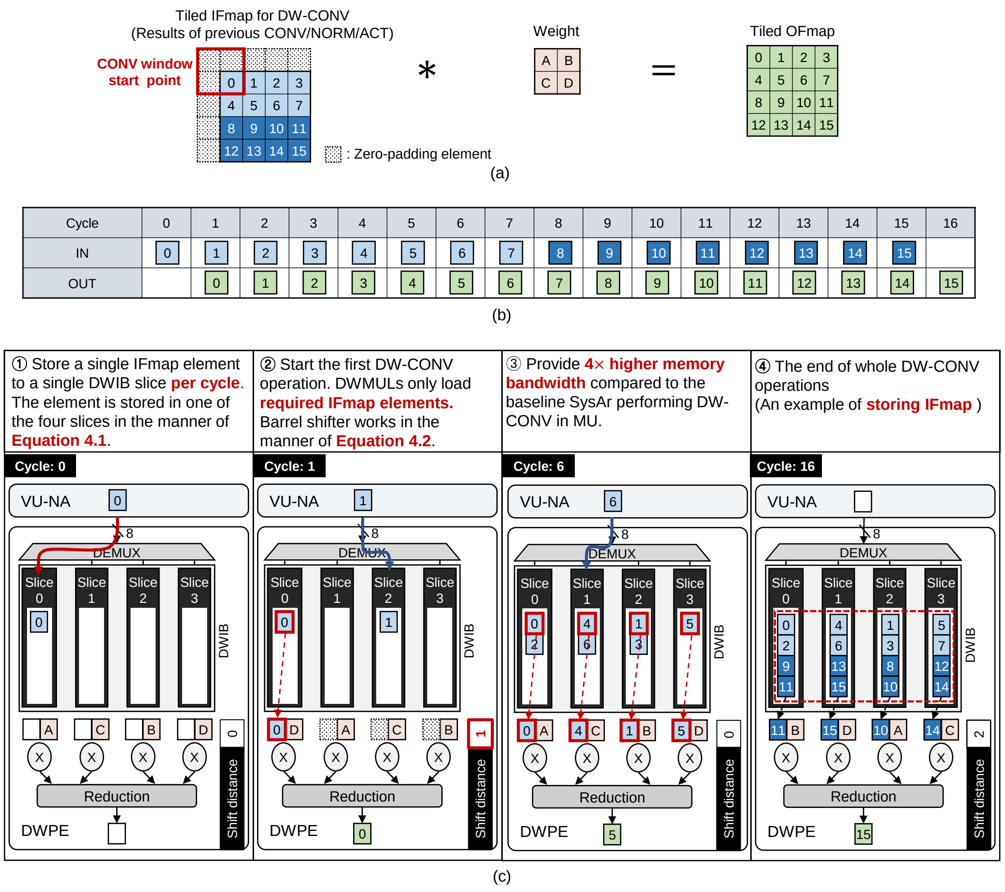
We describe how DWPE operates by showing an example of a DW-CONV layer with stride 1 and with zero-padding: the sizes of inputs are 4×4 for IFmaps and 2×2 for weights (see Figure 4.2(a)). Figure 4.2(b) presents the data movement in DWPE I/O at each cycle, while Figure 4.2(c) shows how DWPE operates internally in detail.

① VU-NA receives the OFmap elements of PW-CONV from MU and ACC, performs BN and ACT, and sends the resulting Fmap to DWPE per cycle at each lane. DWPE takes the IFmap element and stores it to DWIB. At cycle 0, DWIB stores the IFmap element in one of the four DWIB slices. The position of DWIB slice where an IFmap element is stored is determined by 4.1.

$$Position\ of\ DWIB\ slice = (row_{IFmap} + column_{IFmap} \times KH)\ \%\ DWMUL \tag{4.1}$$

In the equation, the row_{IFmap} and $column_{IFmap}$ are the positions of a single IFmap element on a 2-dimensional IFmap. For example, the row_{IFmap} and $column_{IFmap}$ of an IFmap element numbered 0 are both 0.

② At cycle 1, DWPE starts the first DW-CONV operation. The first CONV window requires a single IFmap element numbered 0 and three zero-padding elements. DWMULs only load an IFmap element numbered 0 from a DWIB slice for operation. DWMULs which need zero-padding elements do not work. Because weights are recycled for every OFmap elements in an OFmap, the barrel shifter rearranges the weights by a shift distance according to 4.2. If the zero-padding size is odd, a math symbol in the equation is a plus, and vice versa,


Tiled IFmap for DW-CONV
(Results of previous CONV/NORM/ACT)
CONV window start point
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
▨ : Zero-padding element
*
Weight
A B
C D
=
Tiled OFmap
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
(a)

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
OUT		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(b)
① Store a single IFmap element to a single DWIB slice per cycle. The element is stored in one of the four slices in the manner of Equation 4.1.
② Start the first DW-CONV operation. DWMULs only load required IFmap elements. Barrel shifter works in the manner of Equation 4.2.
③ Provide 4× higher memory bandwidth compared to the baseline SysAr performing DW-CONV in MU.
④ The end of whole DW-CONV operations
(An example of storing IFmap)
Cycle: 0
Cycle: 1
Cycle: 6
Cycle: 16
VU-NA
8
DEMUX
Slice 0
Slice 1
Slice 2
Slice 3
DWIB
Reduction
DWPE
Shift distance
(c)

Figure 4.2: An example of how DWPE operates when the number of DWMULs and DWIB slices is 4. Because the DWPEs perform the same operations on independent Fmaps, we describe an example for one DWPE. (a) An example of DW-CONV where IH and IW are the same as the tile size used in the previous PW-CONV (4×4 in the example). (b) DWPE data in/out movement status at each cycle. The result of PW-CONV, NORM, followed by ACT, is transferred to DWPE every cycle, assuming that IC of the previous PW-CONV is smaller than H_{MU} (weights are not shown for convenience of explanation). (c) DWPE operations in specific clock cycles. We assume that read/write operations of DWIB spend one cycle.

the math symbol is a minus. The shift distance numbered 0 means that the weight elements are allocated to the DWMULs in the same way as storing IFmap elements to the DWIB slices as shown in cycle 0. The equation of shift distance is similar to 4.1; however, there are slight differences in considering the OFmap element, ST, and the zero-padding size.

$$Shift\ distance = [ST(row_{OFmap} + column_{OFmap} \times KH) \pm padding]\ \%\ DWMUL \tag{4.2}$$

③ From cycles 1 to 16, DWPE performs a convolution operation for every cycle. Compared to the baseline SysAr performing DW-CONV in MU, DWPE provides up to 4× higher memory bandwidth, owing to the 4 DWIB slices operating in parallel. To fully utilize all the DWIB slices at every cycle, each IFmap element in a CONV window should be placed in a proper DWIB slice. Cycle 6 is an example of fully utilizing 4× bandwidth. The CONV window requires the IFmap elements numbered 0, 4, 1, and 5: the four elements are distributed, in column-major order, into the first row of the four DWIB slices. DWPE loads the IFmap elements 0, 4, 1, and 5 from each DWIB slice, multiplies them with the corresponding weights, sums up the output to compute OFmap element 5. ④ This data access pattern is repeated until DW-CONV finishes at cycle 16.

Even when a stride is larger than 1, the DEMUX for storing IFmap in the DWIB slice works the same as the stride 1. The pattern of loading IFmaps into DWMUL needs no change. For example, when the stride is 2, IFmaps are loaded in the same way as 6, 8, 14, and 16 cycles of Figure 4.2(b). The barrel shifter does not change the position of weights when the window moves in the row direction; it only shifts weights by 2 when the window moves in the column

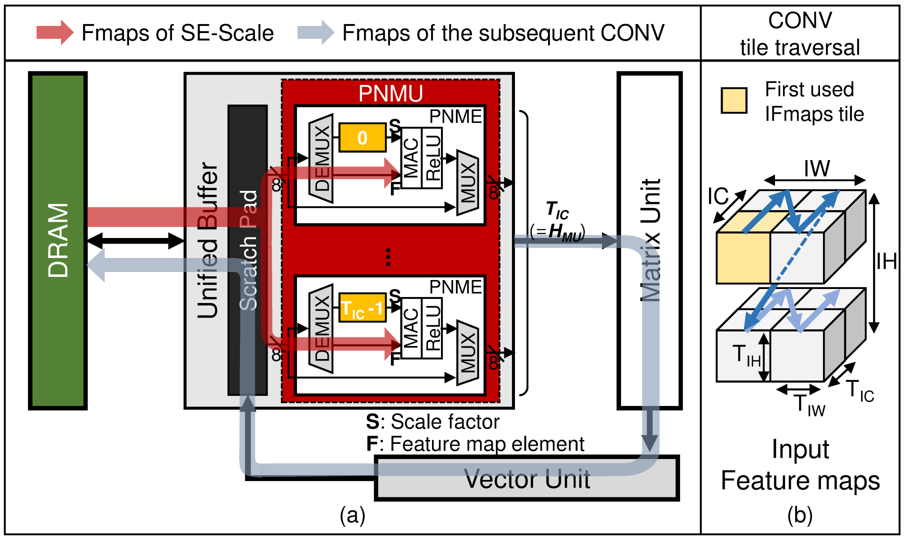


Figure 4.3: Illustration of how PNMU operates. PNMU reads Fmaps from DRAM and performs SE-Scale and the subsequent CONV concurrently. (b) Tile traversal of the next CONV.

direction. Thanks to such a straightforward access pattern, DWPE fully exploits the convolutional reuse on an IFmap, yet being flexible enough to support the various sizes of kernels and the number of DWMULs.

4.1.3 How processing-near-memory unit (PNMU) operates

PNMU processes element-wise operations that commonly precede a CONV layer, such as NORM, ACT, and SE-Scale. Figure 4.3 illustrates how PNMU processes SE-Scale. Here we describe an example of fetching IFmaps of SE-Scale from DRAM as it accompanies fetching from UB. First, scale factors required for SE-Scale are computed by VU, stored at UB, fetched from UB, and stored in the registers inside PNME. Each of T_{IC} PNME holds its own scale factor for a single input channel (see Figure 4.3(a)). Then, PNMU multiplies the scale factors with a tile of Fmaps sized $T_{IW} \times T_{IH} \times T_{IC}$ in an element-wise manner. The output of SE-Scale is directly fed into MU for the subsequent CONV.

After one Fmap tile is complete, one needs to determine which direction to traverse the Fmap tiles because it determines whether the scale factors are reused or not. As explained in Section 2.2.2, SysAr sends a complete OFmap element from ACC to VU. For that, SysAr traverses in IC-direction first (see Figure 4.3(b)); then, the tiles are traversed in IW- and finally in IH-direction.

PNMU overlaps most of the execution time for SE-Scale with the subsequent CONV and saves memory access for SE-Scale, compared to the baseline SysAr. This design includes inevitable overheads, albeit minimal. Because any two IFmap tiles processed in a row have completely different input channels, the scale factors in the PNME registers should be updated for every IFmap tile. When the datapath between UB and PNME is busy updating the scale factors, it cannot feed the Fmap for SE-Scale and the subsequent CONV. However, once a scale factor is updated, it is reused by $T_{IW} \times T_{IH}$, amortizing the time overhead for the scale factor update to $\frac{1}{(T_{IW} \times T_{IH})}$. For the CNN models taking input images with a resolution of 224×224, the smallest $T_{IW} \times T_{IH}$ is 49; limiting the time overhead to 2% (more details in Section 4.2.2).

4.1.4 Overlapping the operation of DW-CONV with PW-CONV

VU-DW and MU can concurrently execute both PW-CONV and DW-CONV in a pipelined manner, resulting in additional speedup with the overlapped execution. Figure 4.4 shows timing diagrams of PW-CONV and the following DW-CONV with and without overlapping. Pipelining clearly reduces execution time. The degree of execution time saving depends on the time to produce

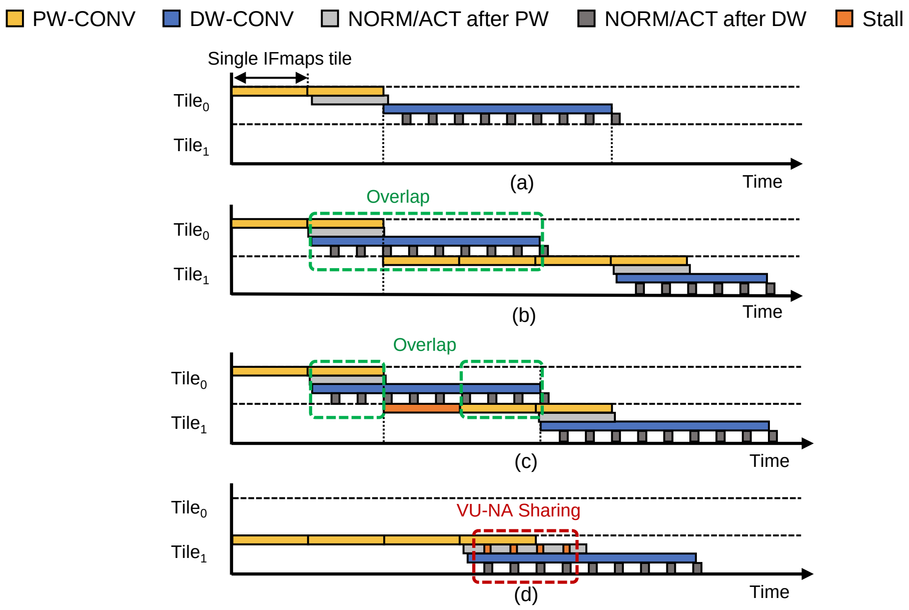


Figure 4.4: Timing diagrams for processing PW-CONV followed by DW-CONV. X-axis represents time and y-axis represents operation sequences about multiple OFmap tiles. MU and VU process PW-CONV and DW-CONV (a) without pipeline, (b) with pipeline ($Cycle_{PW} > Cycle_{DW}$), (c) with pipeline ($Cycle_{PW} < Cycle_{DW}$), and (d) with sharing VU-NA.

an OFmap tile during PW-CONV and DW-CONV, defined as $\frac{T_{IW} \times T_{IH} \times IC}{H_{MU}}$ for PW-CONV and $\frac{T_{OW} \times T_{OH} \times KH \times KW}{DWMUL}$ for DW-CONV. Figure 4.4(b) shows the case when the processing time for an OFmap tile of PW-CONV is larger than that of DW-CONV. Then, the execution time on DW-CONV fully overlaps with PW-CONV. If the processing time for an OFmap tile in DW-CONV is larger than that in PW-CONV (Figure 4.4(c)), there is a stall (shown as an orange bar) in MU, leading to less saving albeit still beneficial.

Meanwhile, the throughput requirement of VU-NA increases twice due to the concurrent processing. It is simple to duplicate VU-NA for DW-CONV and PW-CONV, but it requires 11% additional area (see Section 4.2.1). To avoid this, we share one VU-NA for both DW-CONV and PW-CONV through multiplexing (see Figure 4.4(d)). We present the performance overhead due to sharing VU-NA in Section 4.2.2.

4.1.5 Considerations for designing DWIB

Because the DWIB size affects the execution time and energy of both DW-CONV and PW-CONV and the area of MVP, it must be carefully designed. PW-CONV can suffer from slowdown and more energy consumption due to the additional DRAM memory access if DWIB size is too small. PW-CONV might experience more frequent DRAM accesses due to weight update when concurrently executing PW-CONV and DW-CONV due to a smaller PW-CONV tile size compared to the non-overlapped case [60].

DWIB also affects DW-CONV performance. Because DW-CONV uses a kernel size of 3×3 or larger, the edge part of the previously used IFmap tile is required (we call it an edge IFmap). All the edge IFmaps are stored in UB and must be moved to DWIB through a bidirectional path between UB and VU. However, because the same path is occasionally used for sending the OFmaps of DW-CONV to UB, contention can occur. The size of edge IFmaps increases in proportion to the number of the DW-CONV IFmap tiles, which is closely related to DWIB. We quantify overhead by the contention (Section 4.2.2), and evaluate MVP performance and energy changes according to the DWIB size (Section 4.2.4).

Table 4.1: SysAr, VU-DW, PNMU, and DRAM specifications.

Resource	Value
Systolic Array (TPU):	1
Matrix Unit	64×64 (INT8 MAC)
Accumulator (Unit)	64 (INT32 ADD)
Unified Buffer	1 (512 KB, 8192×512 SRAM)
Accumulator (Buffer)	1 (128 KB, 512×2048 SRAM)
Vector Unit (NORM, ACT)	1
Vector Unit (DW-CONV):	1 (64 Depth-Wise Processing Elements)
Depth-Wise Multiplier	64×5 (INT8 MUL)
Depth-Wise Input Buffer	64 (Each size is 260B, 260×8 SRAM)
Depth-Wise Weight Buffer	64 (Each size is 100B, 20×40 SRAM)
Adder tree	64 (Reduction Unit, Registers)
Barrel Shifter	64
DEMUX	64
Processing-Near-Memory Unit:	1 (64 Processing-Near-Memory Elements)
MAC	64 (INT8 MAC)
ACT	64 (Comparator)
Register	64 (1B)
MUX	64
DEMUX	64
DRAM [58]	LPDDR4-3200, 12.8 GB/s, 13 pJ/bit

4.2 Evaluation

We quantified the benefits of MVP architecture over SysAr on execution time, energy, and area. Then we explored the various design spaces of MVP, such as DWIB size and the number of DWMULs. We also demonstrated the MVP's performance improvement across various SysAr configurations with different MU dimensions and the size of UB and ACC.

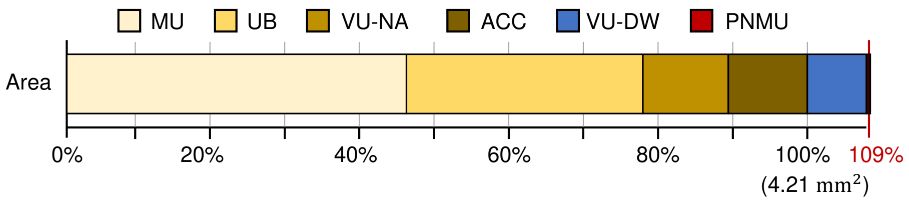


Figure 4.5: MVP area breakdown.

4.2.1 Experimental setup

SysAr is the baseline architecture for evaluation. We set the baseline to have an MU with 64×64 MACs, 512KB UB, and 128KB ACC buffer (see Table 4.1). This is the best-performing configuration on ResNet-50 and VGG-16 models based on EDA^2P (Energy Delay $Area^2$ Product) metric [61]. To get a reasonable SysAr configuration for ST/PW-CONV, we used ResNet-50 and VGG-16 because they are the most well-known CNN models among the ones not using DW-CONV and SE. VU-DW consists of 64 (=W_{MU}) DWPEs, each including five 8-bit DWMULs, a 260B DWIB with five slices, a 100B DWWB, and an adder tree. PNMU has 64 (=H_{MU}) PNMEs, each including DEMUX, 8-bit MAC unit, ACT unit, MUX, and register. Compared to SysAr, MVP adds an area overhead of 9% where VU-DW and PNMU incur 8% and 1%, respectively (see Figure 4.5).

We used Timeloop [70] for evaluation. Timeloop is a simulator for evaluating CNN accelerators. Timeloop has two main components: a *model* to provide performance, area, and energy projections of hardware components and a *mapper* to find the optimal mapping of tiling and loop order of any layers on the targeted architecture. Timeloop requires an ERT (energy reference table) and an ART (area reference table) that contain energy and area values for each hardware component. We used a custom ERT and ART, whose values were calculated as follows. We synthesized logic components such as multipliers and MACs based on TSMC 40nm technology and evaluated the SRAM component of MVP using CACTI [88]. We set the clock frequency and data precision to be 500MHz and 8-bit, respectively. We modified Timeloop to support DW-CONV and SE layers. The mapper of the existing Timeloop officially supports the layer shape only for CONV. However, Timeloop elaborates that they can support FC by modifying the layer shape of CONV (IH, IW, OW, OH, KW, and KH are fixed to 1, whereas only IC and OC are changed) as mentioned in [70]. We modified the mapper similarly to FC, considering that DW-CONV is a variant of CONV and SE consists of two FCs and one element-wise multiplication layer. Also, we modified the model to add architectures such as VU-DW and PNMU that are not supported by Timeloop. We validated VU-DW and PNMU by using the synthesis results of our RTL design.

We used EfficientNet-B0/B4/B7, MnasNet, and MobileNet-V1/V2 as our target CNN models. Although MobileNet-V1 does not include MBConvs, MVP can support the model as it has a similar layer pattern where DW-CONV follows PW-CONV and multiple depth-wise separable CONVs are repeated. We didn't evaluate NoisyStudent, FixEfficientNet, and Meta Pseudo Labels because they exploit the same MBConv and thus have a similar structure to EfficientNet.

4.2.2 Performance and energy evaluation

We demonstrate the performance and energy efficiency of MVP for the target CNN models. We quantify the performance improvements in detail by in-

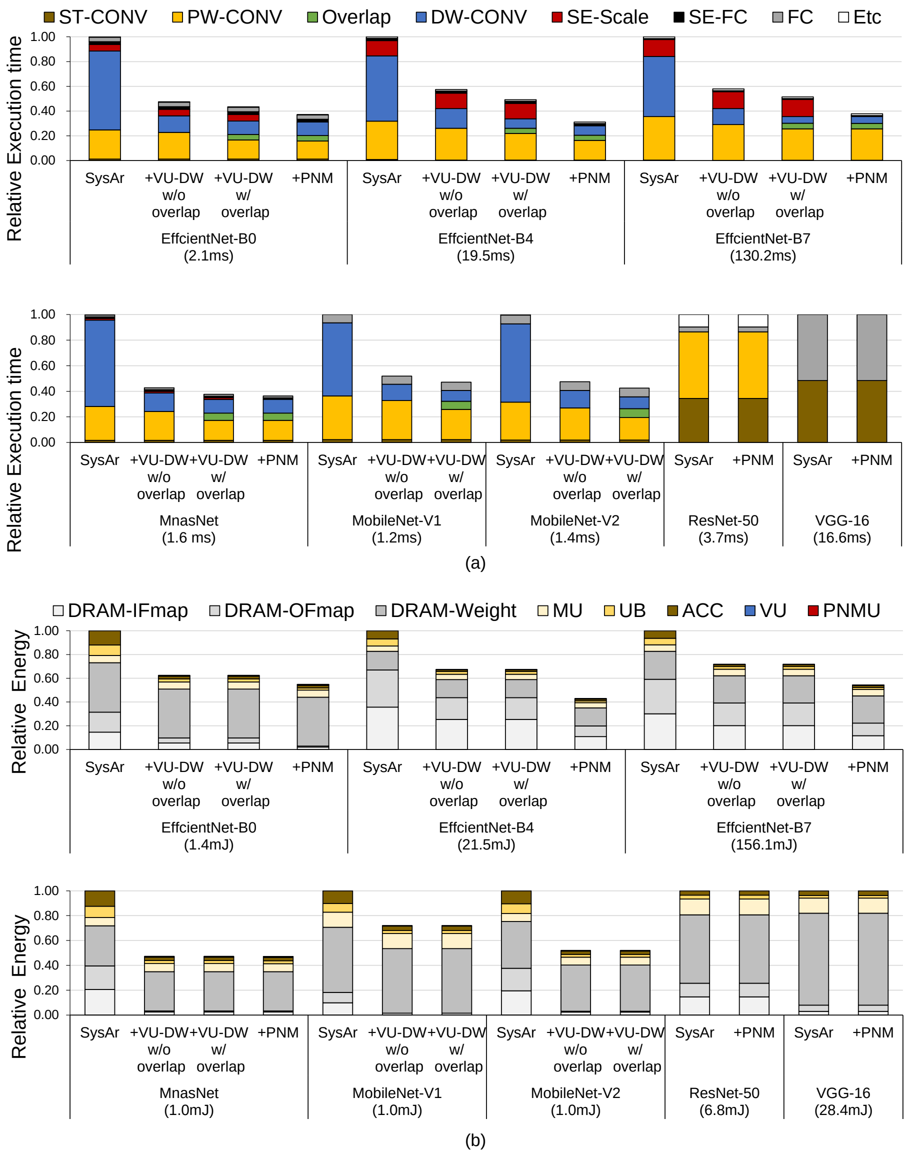


Figure 4.6: (a) Relative execution time and (b) relative energy consumption of MVP over SysAr by incrementally applying VU-DW, the overlapped execution between PW-CONV and DW-CONV (shown as overlap), and PNMU. The absolute execution time and energy comsumption of the baseline SysAr is shown below the name of each CNN model. The execution time of MobileNet-V2 with SysAr is in a similar ball park to the reported execution time of EdgeTPU [30] (2.6ms), which has the same peak FLOPS as the evaluated SysAr.

crementally applying VU-DW, the overlapped execution of PW-CONV and DW-CONV, and PNMU. We verify that there is no significant difference between SysAr and MVP when running ResNet-50 and VGG-16 as MVP can operate like SysAr by bypassing the added units and the additional wiring cost is negligible. We make the following key observations.

MVP significantly reduces the execution time of DW-CONV and SE-Scale. Figure 4.6(a) shows the relative execution time of MVP on the latest CNN models.[1] First, when VU-DW is applied without overlapping, the execution time reduces by 51% on average, mostly attributed to saving time in processing DW-CONV. VU-DW exploits its high internal memory bandwidth supported by DWIB; and it utilizes up to 5× more MACs than SysAr, which utilizes only N MACs (as shown in Figure 2.8), thereby reducing the execution time of DW-CONV by 78% on average. Also, VU-DW alleviates the off-chip memory bandwidth bottleneck by forwarding the OFmap elements of PW-CONV from ACC directly into VU-DW (through VU-NA) and thus removing the accesses to DRAM. The processing speed of PW-CONV also improves, especially for larger models such as EfficientNetB7. Second, with the overlapped execution of PW-CONV and DW-CONV, we further observe an average speedup of 5%. Last, except for MobileNet-V1/V2, which do not include SE-Scale, PNMU improves the processing speed additionally by 8% on average. As mentioned in Section 4.1.3, PNMU removes memory accesses to UB and DRAM while executing SE-Scale. The speedup is higher for models with large Fmap sizes. Overall, the total execution time is improved by 61% on average.

[1]We point out that the SysAr configured for evaluation has a similar performance to EdgeTPU [30]: the execution time of MobileNet-V2 with SysAr (1.4ms) is similar to that with EdgeTPU (2.6ms).

MVP significantly saves energy consumption primarily by reducing the DRAM memory accesses. Figure 4.6(b) shows the relative energy consumption of MVP over the baseline SysAr. When VU-DW is applied without overlapping, total energy consumption is reduced by 41%, where the energy reduction in DRAM accesses is dominant. It is because VU-DW eliminates the DRAM accesses that were necessary for delivering the intermediate Fmaps between PW-CONV and DW-CONV before. VU-DW also reduces the energy consumption of UB in two ways: 1) by eliminating the memory accesses on the intermediate Fmaps between PW-CONV and DW-CONV and 2) by reducing the repeated memory accesses on the same Fmap elements through exploiting the convolutional reuse stated in Section 4.1.2. The reduced energy consumption in ACC, on the other hand, attributes to the on-the-fly accumulation of partial sums through the adder tree in DWPE; the partial sums are accessed through small registers instead of large memory in ACC while processing DW-CONV. Meanwhile, there is little energy saving by overlapping because it affects only the timing of processing each layer. PNMU further reduces the energy consumption by 9%, owing to the elimination of DRAM and UB traffic in SE-Scale. Putting it all together, MVP reduces energy consumption by 47% in total.

We further quantify the performance overheads of MVP due to the limited DWIB capacity and the time-division multiplexing of VU-NA. When an IFmap of DW-CONV does not fit into the limited DWIB capacity, as explained in Section 4.1.5, we fetch the edge IFmaps from UB, which incurs small performance overhead. Figure 4.7 shows that the overhead of fetching edge IFmaps (shown as Edge overhead) is 4% for EfficientNet-B0. This overhead happens only in close-to-input BBs whose IFmaps are large. In contrast, Share overhead

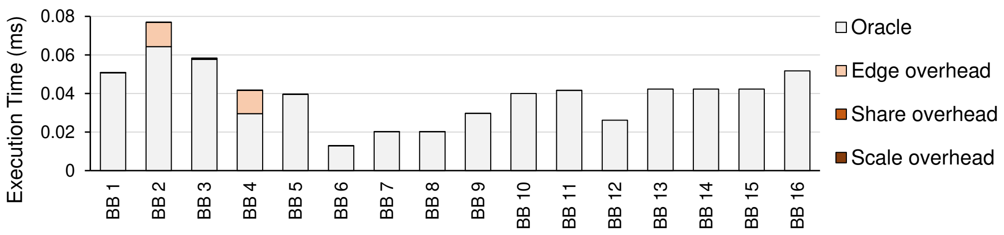


Figure 4.7: Execution time of EfficientNet-B0 over BBs. Edge overhead means performance overhead due to the datapath contention caused by the edge IFmaps (Section 4.1.5). Share overhead means performance overhead due to the resource contention by VU-NA sharing (Section 4.1.4). Scale overhead means performance overhead due to the datapath contention when SE-scale is performed in PNMU (Section 4.1.3). Oracle is an execution time when all three overheads do not exist.

(performance overhead due to the contention by VU-NA sharing) and Scale overhead (performance overhead caused by the contention when SE-scale is performed in PNMU) have little effect on execution time. The Share overhead is low because the frequency of traffic requested to the VU-NA is low. VU-NA is utilized when CONV produces OFmaps. In the case of PW-CONV, VU-NA requests occur in a period of $\frac{IC}{T_{IC}}$ cycles, as mentioned in Section 2.2.2. DW-CONV uses VU-NA in a period of 2 or 5 cycles for the kernel sizes of 3×3 and 5×5, respectively, when using 5 DWMULs per DWPE. Therefore, as VU-NA requests from PW-CONV and DW-CONV occur intermittently, MVP can operate with little overhead using a single VU-NA through time multiplexing. The Scale overhead is negligible because the period of reading scale factors is much less than that of Fmaps, as mentioned in Section 4.1.3. Overall, the total performance overhead for the target CNN models is 2.6% on average.

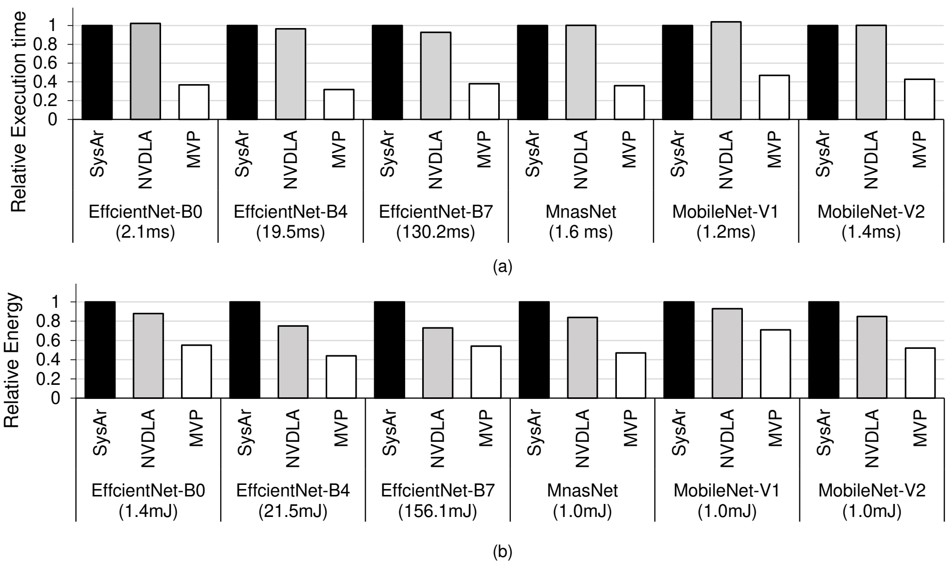


Figure 4.8: (a) Relative execution and (b) relative energy of three accelerators to compare MVP with NVDLA. The absolute execution time and energy consumption of the baseline SysAr is shown below the name of each CNN model.

4.2.3 Comparing MVP with NVDLA

We further compare the execution time and energy consumption of MVP with those of NVDLA. NVDLA consists of a convolution MAC unit, a convolution buffer, and a convolution accumulator. The convolution MAC unit consists of H_{MU} MAC cells, which include W_{MU} multipliers and an adder tree. A single MAC cell takes one of T_{IC} IFmap elements, while H_{MU} MAC cells take one of T_{OC} weight elements. The convolution buffer is an on-chip buffer that stores IFmaps and weights. The convolution accumulator stores the partial sums that are computed by the MAC cells until they become a complete OFmap element. We set NVDLA as a 64$\times$64 convolution MAC unit, a 512KB convolution buffer, and a 128KB convolution accumulator similar to the baseline SysAr ($\simeq$TPU).

MVP reduces execution time compared to NVDLA, and the reduction rate is almost the same as that of SysAr. Figure 4.8(a) shows the relative execution time of SysAr, NVDLA, and MVP for the latest CNN models. NVDLA is similar to SysAr in how to allocate the IFmap, weights, and OFmap tiles to the convolution MAC unit, and the OFmap element can be moved out of the convolution accumulator when partial sums become a complete value. Therefore, NVDLA has less than 2% of MAC utilization when processing DW-CONV as mentioned in Section 2.4.1. Overall, MVP reduces the execution time for the entire CNN by 60% on average.

MVP reduces energy consumption compared to NVDLA, but the reduction rate is less than that of SysAr. In the CNN models where PW-CONV, NORM, ACT, and DW-CONV are repeated, MVP executes PW-CONV, NORM, and ACT from MU through VU-NA and performs DW-CONV in VU-DW by taking results directly from VU-NA. In contrast, NVDLA must access UB or DRAM after processing PW-CONV, NORM, and ACT. Thus, MVP reduces energy by 34% on average compared to NVDLA as shown in Figure 4.8(b). However, the energy consumption of NVDLA is smaller than that of SysAr. Because NVDLA reuses weights in the on-chip memory, it reads fewer weights from DRAM. In addition, the gap between the two accelerators is getting larger as the image resolution of the CNN model increases because the accelerator has to read weights repeatedly from DRAM or UB. Therefore, NVDLA saves energy consumption by 12% on average with 224×224 resolution (EfficientNet-B0, MnasNet, MobileNet-V1, and MobileNet-V2) and by 26% on average with 300×300 and 600×600 resolution (EfficientNet-B4 and EfficientNet-B7) compared to SysAr.

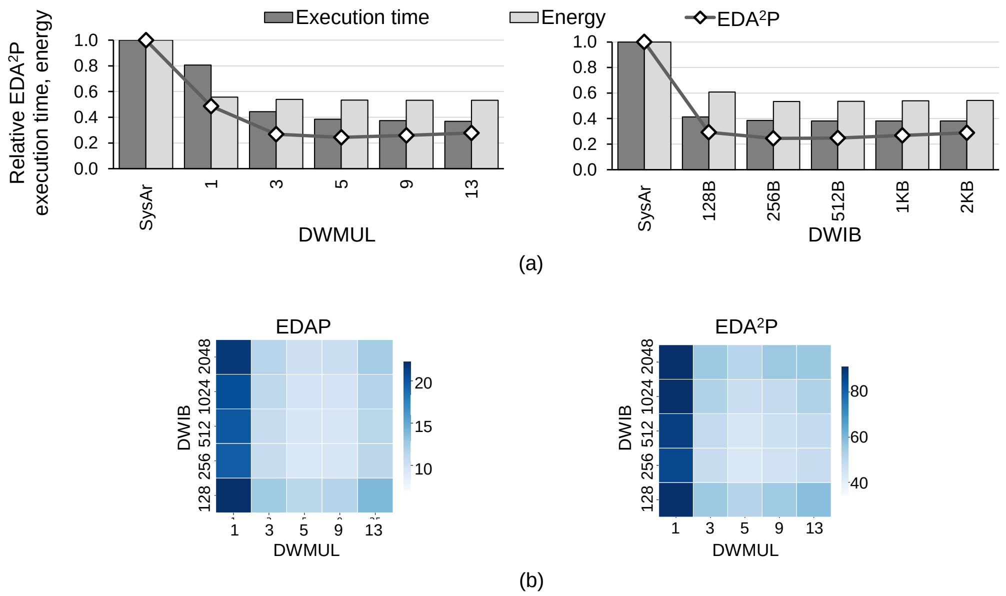


Figure 4.9: (a) Execution time, Energy, and EDA^2P when the number of DW-MULs is varied for a 256B DWIB size (Left side) when the DWIB size is varied for 5 DWMULs (Right side). (b) $EDAP$ and EDA^2P heatmap by the number of DWMULs and the DWIB size (the lower, the better [61]).

4.2.4 Exploring the design space of MVP architecture

Increasing design parameters of MVP, such as the DWIB size and the number of DWMULs, can improve performance and energy efficiency but worsen area efficiency. We explore the design space of MVP and find optimal design parameters for the given SysAr configuration by using $EDAP$ and EDA^2P metrics, where the latter puts more weights on area efficiency. We use the geometric mean of all target CNN models to show simple results.

The number of DWMULs significantly affects the execution time. The left side of Figure 4.9(a) shows the execution time, energy, and EDA^2P of MVP for a various number of DWMULs with 256B DWIB. Even only adding one DWMUL to SysAr greatly reduces the execution time and energy consumption;

it gives the same internal memory bandwidth as SysAr but eliminates DRAM and UB accesses between PW-CONV and DW-CONV, and overlaps the executions. PNMU additionally removes the DRAM and UB access, and hides the execution. Increasing the number of DWMULs improves the processing speed by utilizing more ALUs and memory bandwidth until the performance is saturated when the number of DWMULs is 5. The performance saturates because the execution time of PW-CONV remains unchanged after adding one DWMUL and thus dominates. In contrast, populating more than 1 DWMUL has little impact on total energy consumption. This is because the number of on-/off-chip memory accesses, which dominate energy consumption, remains the same after adding one DWMUL. Finally, EDA^2P decreases until 5 DWMULs and then starts increasing because of the growing area overhead.

Even MVP with small DWIBs outperforms SysAr, with high energy efficiency. Increasing the DWIB size has pros and cons. It enables DWIB to accommodate larger DW-CONV tiles, reducing both UB accesses on edge IFmaps and repeated DRAM accesses on the same weights of PW-CONV. However, the energy per memory access grows as DWIB becomes larger. The right side of Figure 4.9(a) shows the execution time, energy, and EDA^2P with various sizes of DWIBs. DWIB of 128B size causes repeated memory accesses to DRAM for fetching weights of PW-CONV, diluting the energy savings of MVP. Increasing DWIB over 256B worsens the energy efficiency because the energy per memory access increases. In contrast, an Fmap tile already, at 256B, becomes large enough to eliminate the repeated DRAM accesses on the same weights. The energy spent by the edge traffic is below 0.3% on average, which is negligible. Because of these reasons, the optimal size minimizing the energy consumption

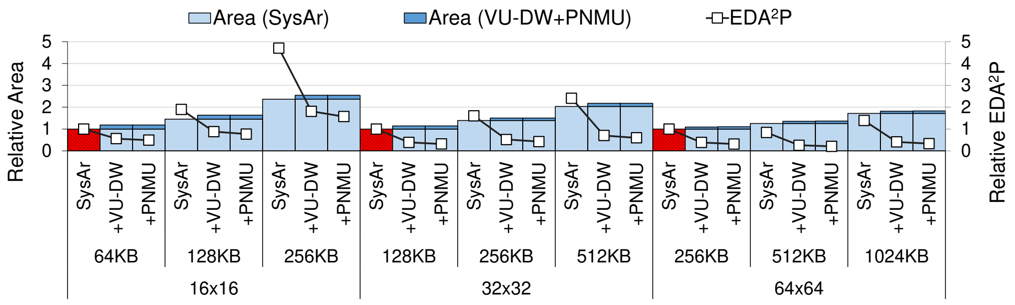


Figure 4.10: Area and EDA^2P of MVP while varying MU, UB, and ACC parameters. The outermost category on the x-axis represents the MU size and the inner category represents the UB size. We select a candidate configuration that fits UB and ACC capacity in proportion to the change in MU size and set ACC as one fourth of the UB size. We obtain the design parameters of MVP for each SysAr configuration by using the design space exploration conducted in Section 4.2.4. EDA^2P values are calculated as the geometric mean for the target CNN models. Both EDA^2P and area are expressed as relative values based on each SysAr configuration, which has the smallest UB in each MU size (colored red).

for DWIB is 256B. Meanwhile, as the DWIB size increases, the area increases, and the area efficiency worsens. As the DWIB size increases, it has an area overhead up to 18%, and EDA^2P increases as well.

We choose the parameters for MVP, the DWIB size and the number of DW-MULs, for design space exploration based on the $EDAP$ and EDA^2P metrics (see Figure 4.9(b)). In general, increasing each of the two parameters leads to better performance while increasing area and energy overhead. We find that the point with 5 DWMULs and 256B DWIB gives the best (lowest) $EDAP$ and EDA^2P; we set them as default parameter values for MVP.

4.2.5 Evaluating MVP with various SysAr configurations

The proposed MVP is flexible enough to improve both the performance and energy efficiency across a wide range of configurations. We quantify the performance improvement of MVP with various sizes of MU, UB, and ACC. As shown in Figure 4.10, all the configurations show significant EDA^2P improvement with VU-DW, which is further increased by PNMU, with only a small area overhead. In conclusion, MVP reduces the EDA^2P by at least 51% and up to 76% for a range of configurations of SysAr.

4.3 Related Work

Conventional DNN inference accelerators: Accelerators often use 2D MAC array structure to accelerate CNN inference by using their own dataflows in exploiting the data reuse characteristics of ST/PW-CONV. NVDLA [65] consists of T_{OC} 1D vector engines that process T_{IC} MAC operations. NVDLA does not use systolic execution; however, dataflow, data mapping, and UB bandwidth are similar to TPU. Because NVDLA suffers from the same challenges as TPU as mentioned in Section 2.4.1, it also processes DW-CONV inefficiently. ShiDianNao [24] specializes in reusing IFmaps between sliding windows. Although ShiDianNao is good for processing ST and DW-CONV with a large kernel size, it cannot process PW-CONV efficiently because it does not utilize OC parallelism as mentioned in [55]. As opposed to SysAr, TPU v3 [49] changes VU structure to accelerate less arithmetically intensive operations such as inverse-square-root of BN while training, albeit not elaborating on details about processing DW-CONV in the modified VU. In addition, as TPU v4 [48] reuses hardware designs of TPU v3 except for several components such as on-chip memory capacity, on-chip interconnect, and DMA, the VU of TPU v4 is the same structure as that of TPU v3. There have been processing-near-DRAM studies [17, 26, 53] to provide high off-chip memory bandwidth during inference. Because [17, 26] use dataflow architecture such as Eyeriss v1 [14] and systolic array, they still do not process DW-CONV efficiently. In contrast, [53] has advantages for memory-intensive operations but has weaknesses for compute-intensive ST-CONV operations.

Prior works supporting both ST- and DW-CONV: Previous architectural solutions have been mainly proposed to process both ST- and DW-CONV in an MU. Eyeriss v2 [15] provides flexibility in moving data from UB to PE by composing the memory hierarchy in two stages, mesh and all-to-all, so that both types of CONV could be processed while maintaining the existing dataflow. However, a major drawback of this design is the large area overhead as it needs to provide local SRAM for all the three data types. [9, 98, 102] make hardware reconfigurable to process two types of CONV in an MU by changing dataflow and data mapping according to the CONV type. These solutions can accelerate DW-CONV using the existing MU, but require $\mathcal{O}(N^2)$ UB bandwidth directly to the MU and incur a huge amount of wiring cost, one of the most expensive resources in conventional CNN accelerators. In-memory accelerators [4, 5] that accelerate DW-CONV have been proposed, albeit they target CNN models using binarized DW-CONV. [95] proposes a scheduling method that reduces off-chip access by fusing three consecutive layers inside the MB-Conv. However, it does not deal with the under-utilization when conventional accelerators operate DW-CONV. HPIPE [35] generates a customized dataflow inference accelerator for FPGA by considering layer information of the targeted CNN model and hardware resources such as DSP and BRAM. HPIPE minimizes unnecessary off-chip memory access and computation by processing multiple layers in a pipelined manner and by skipping zero activation and weight. HPIPE is efficient where the user processes a single CNN model or hardware changes are flexible.

A full-stack accelerator search technique (FAST) [104],which is highly similar to our motivation, deals with the problem of extreme under-utilization of the systolic array when processing DW-CONV in TPU v3. To solve the under-utilization of systolic arrays, FAST proposes an automated search framework to find optimal software and hardware combinations. The software technique focuses on the scheduling method for the fusion of DW-CONV with other layers. The hardware technique conducts design space exploration by using the number of MUs, MU size, L1 SRAM, and L2 SRAM as input configurations for the server-scale tiled architecture. Through the results, FAST suggests the best combinations for the performance compared to TPU v3 and FAST found best design in which the architecture has smaller systolic arrays, smaller L1 SRAM, and larger last level SRAM. Although increasing the capacity of last level SRAM proposed by FAST is an easy and sure way to reduce off-chip memory access, it requires a large area. In contrast, our method of extending the vector unit is expected to benefit the area because SRAM needs much higher area cost than logic components [90].

NX27V [3] is the first commercial RISC-V vector processor introduced by Andes and it is developed to process AI efficiently. The NX27V contains a decoupled out-of-order vector processing unit (VPU), specialized for processing operations used in AI models. VPU handles matrix multiplication, convolution, depth-wise convolution, fully connected. For example, if matrices A and B are 32x4, and 4x32, respectively, VPU performs matrix multiplication using eight vector registers in parallel (assuming that the precision is FP16). VPU aims to process a massive number of computations, which has a similar purpose to a systolic array. In contrast, the vector unit in MVP aims to target operations with low OP/B. Therefore, although the name of the VPU is similar to the vector unit inside the MVP, there is a difference in processing both standard convolution and depth-wise convolution in a single computing unit or not.

Chapter 5

Load Balancing Techniques for BERT Training

5.1 Contribution

We propose novel load balancing techniques called traffic shaping and resource sharing for transformer training acceleration. The proposed technique utilizes the advantage that there is no dependency between mini-batches within a gradient accumulation step, and performs multiple iterations independently in a single accelerator. We use the same hardware resources for the baseline architecture.

5.1.1 Tiled architecture

The baseline tiled architecture is composed of multiple clusters which consist of processing element (PE) tiles (see Figure 5.1). The baseline architecture can operate independently in a cluster unit. Multiple PE tiles are connected to a shared L2 SRAM through an on-chip crossbar interconnection network. The crossbar interconnection network supports broadcasting to multiple PEs.

To process various operations efficiently, a PE tile is composed of a matrix

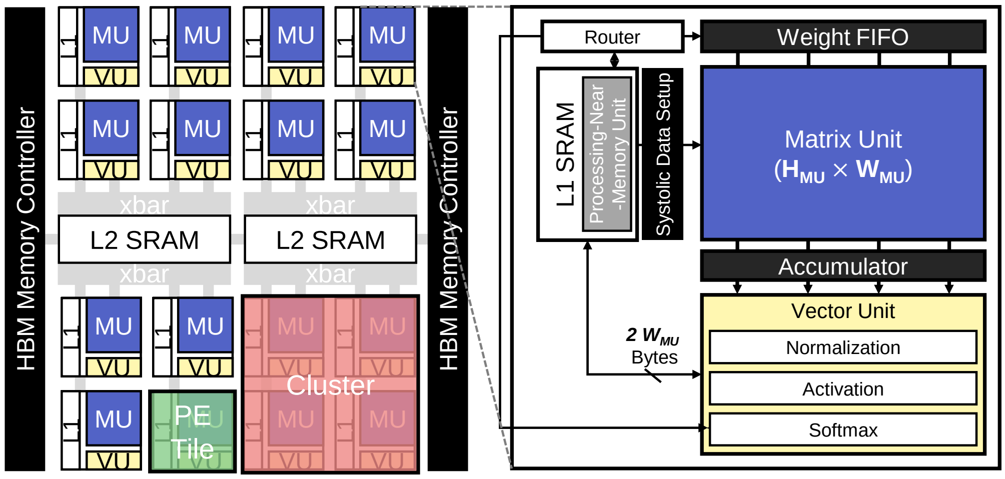
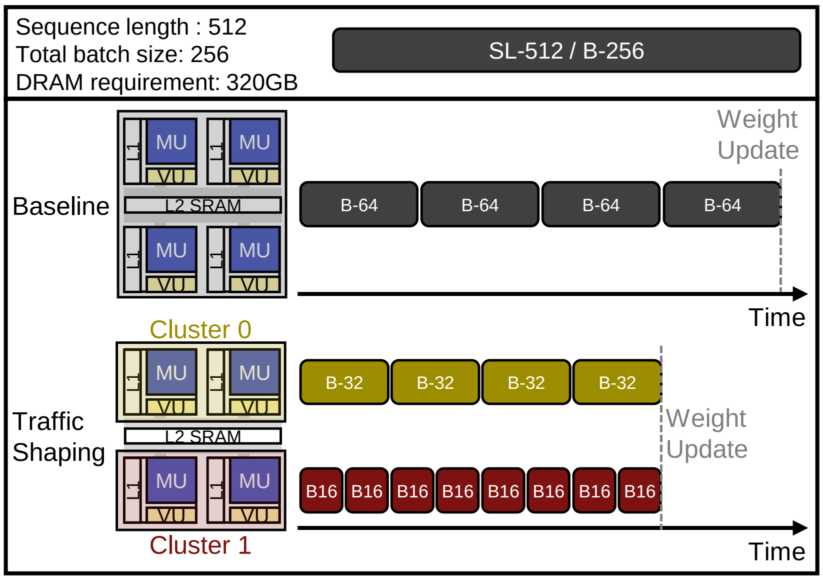
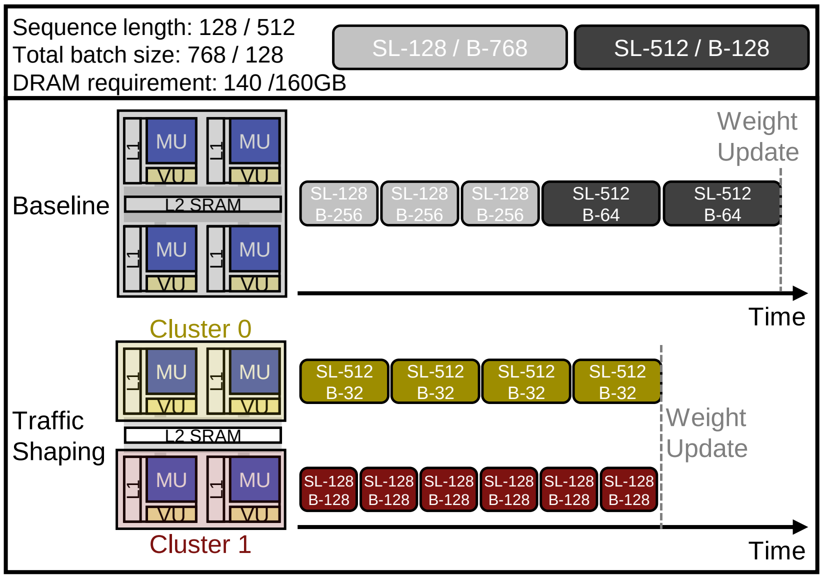


Figure 5.1: Baseline tiled architecture.

unit (MU), a vector unit (VU), a processing-near-memory unit (PNMU), an L1 SRAM, a systolic data setup, a weight FIFO, an accumulator, and a router. The PE is similar to an MVP [59]. The MU is a systolic array that performs $H_{mu} \times W_{mu}$ MAC operations. The VU handles normalization, activation, and softmax layers. The PNMU performs element-wise operations and simple activation processes, such as ReLU. The MU and VU inside the PE can operate in both a pipelined and an independent manner. The L1 SRAM stores the input, weight, and output. We assume that the precision is the 16-bit floating point for the training. The weight FIFO temporarily stores a part of weights transferred from the L1 SRAM, sending the weight to the registers inside the MU. The accumulator accumulates the partial sums calculated from the MU and stores them in a buffer until the partial sums become complete output elements.



(a) Traffic shaping with the same 512 sequence length



(b) Traffic shaping with different sequence lengths

Figure 5.2: DRAM Traffic shaping example for BERT training: (a) traffic shaping with the same SL, and (b) Traffic shaping with the different SLs. The accelerator is divided into two clusters. We assume that the DRAM capacity is 80GB.

5.1.2 DRAM traffic shaping

Traffic shaping is a bandwidth optimization technique that alleviates fluctuations in DRAM access demands by dividing an accelerator into multiple clusters and forcing different clusters to operate asynchronously. During the training process, traffic shaping can be applied during the gradient accumulation step. Because multi-tenant inference can freely adopt DNN models with various computational characteristics, this approach is more likely to show improved performance with traffic shaping. However, as training uses a single DNN model, how tasks are mapped to clusters has a major impact on the performance outcome.

Depending on whether the SLs are identical or different, the mapping of tasks to the cluster can differ.

Traffic shaping with the same SL: when a dataset consists of one SL, traffic shaping improves the performance by allocating a different mini-batch size to each cluster. Figure 5.2(a) presents an example of traffic shaping when the dataset consists of 512 SL. The accelerator processes the total batch size by dividing it into several mini-batches due to the DRAM capacity limitation. Training with 512 SL and the 256 batch size requires 320GB of DRAM capacity. The baseline allocates 64 mini-batches and executes all PE tiles at the same time. The accelerator performs a weight update after processing four runs of 64 mini-batches.

To apply traffic shaping, a single accelerator is divided into several clusters and each cluster assigns a task. Each cluster assigns a task with a different mini-batch. If two clusters allocate tasks of the same mini-batch size, there is a high probability that the layers processed in the two clusters will be similar. In such a case, the performance improvement is small because training is performed similarly to the baseline. To improve the performance, therefore, clusters 0 and 1 allocate one 32 mini-batch and two 16 mini-batches, respectively. Due to the DRAM capacity limitation, the sum of the DRAM requirements for these two tasks is less than 80GB.

Because dividing the batch size reduces the reuse of weight parameters, there is a trade-off in which the weight parameters must be read several times from storage. However, because the weight parameter is significantly smaller than the input and output, this side-effect is negligible.

Traffic shaping with different SLs: when a dataset consists of multiple SLs, traffic shaping achieves a performance improvement by allocating different SLs to each cluster. Figure 5.2(b) illustrates an example of traffic shaping when the dataset consists of 128 and 512 SLs. Clusters 0 and 1 allocate 512 and 128 SLs, respectively. Because the two SLs utilize a different proportion of the execution time of the layers (see Figure 1), there is a high probability that the two clusters operate layers with different characteristics. To maximize the utilization of PE tiles fully, the mini-batch size per task is set to the DRAM capacity provided by the accelerator.

Most of the tasks performed for each cluster are terminated at different times. When a task in one cluster finishes first, the other cluster uses the entire accelerator. In this case, as the accelerator functions as a baseline, the accelerator does not have the benefit of traffic shaping. Thus, it is easier to realize a performance improvement if the two tasks have similar execution times.

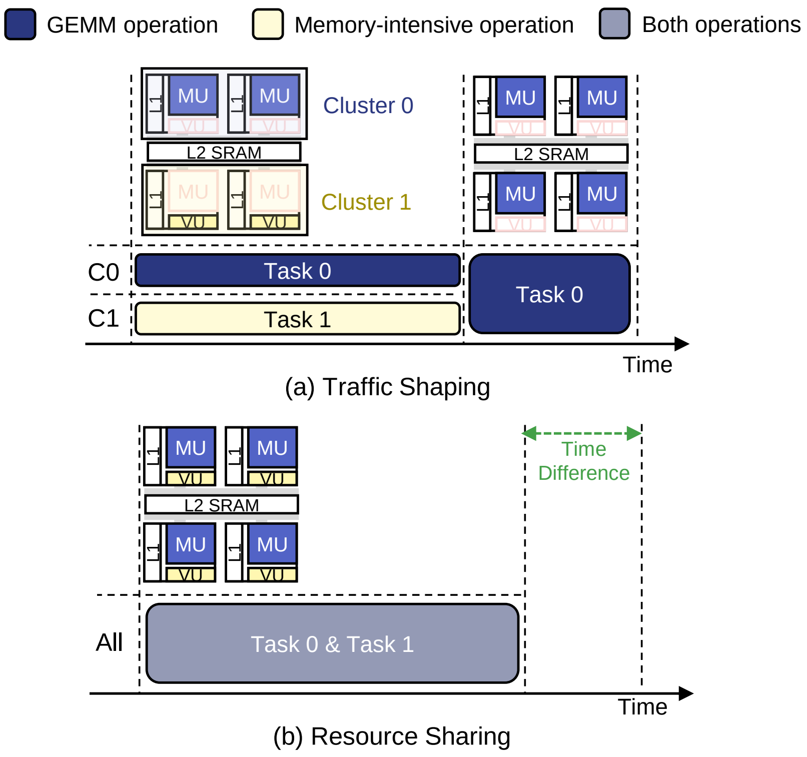


Figure 5.3: Figure 8. Mapping description and timing diagram of (a) traffic shaping and (b) resource sharing. Red indicates that the computation unit does not operate. C0 and C1 denote clusters 0 and 1, respectively.

5.1.3 Resource sharing

Resource sharing increases the utilization of computation units by simultaneously operating MU and VU in a single PE tile. It can be applied when one cluster processes a GEMM operation and the other processes a memory-intensive operation. Traffic shaping reduces the execution time of the memory-intensive operation by distributing the fluctuations in the DRAM bandwidth demand. In contrast, resource sharing further reduces the execution time of GEMM operations by utilizing the MU and VU more than traffic shaping.

Figure 5.3 illustrates a mapping description and timing diagram of traffic shaping and resource sharing. For simplicity of the explanation here, tasks 0 and 1 are assumed to be GEMM and memory-intensive operations, respectively. Traffic shaping assigns task 0 to cluster 0 and task 1 to the remaining clusters (see Figure 5.3(a)). During task 0, cluster 0 uses only MUs for the GEMM operation but does not use the VUs (red in the figure). In contrast, cluster 1 does not use MUs. Thus, under-utilization of computation units affects both clusters.

Resource sharing operates two clusters synchronously when task 0 is the GEMM operation and the other is the memory-intensive operation (see Figure 5.3(b)). It processes task 0 at the MUs in all PE tiles and task 1 at the VUs. Compared to traffic shaping, resource sharing can utilize twice the numbers of MUs and VUs. However, this only reduces the execution time of the GEMM operation because the memory-intensive operation is already a memory bandwidth bottleneck even when using half of the VUs.

There is a trade-off that slows down memory-intensive operations because resource sharing doubles the bandwidth requirements of GEMM operations. However, the decreased execution time of GEMM operations is greater than the increased execution time of memory-intensive operations.

During resource sharing, both tasks share L1 SRAM. Task 0 is highly likely to utilize data reuse in the L1 SRAM. To increase the reusability, most of the L1 SRAM stores the data of task 0. In contrast, task 1 rarely reuses data and it is processed in a streaming manner. To process task 1 without stalls, the L1 SRAM needs a capacity of at least $2 \times W_{mu}$ in consideration of double-buffering. In the experiment, we set the L1 SRAM capacity $1024 \times W_{mu}$ for task 1 to support bulk data transfers.

Load balancing techniques determine the optimal mapping through a search tool. Mapping is determined according to the combination of the SL and the total batch size of multiple tasks. Because training repeats the same operation during the period of a single gradient accumulation step, we only need to consider mapping for a single gradient accumulation step. Thus, if the compiler generates code for a single gradient accumulation step, it can be used during the entire training process.

5.2 Evaluation

5.2.1 Experimental setup

We model the baseline tiled architecture for an evaluation, as shown in Figure 4. The baseline architecture has 32 PE tiles and 40MB of L2 SRAM. We set a PE tile to have an MU with 64×64 MACs, a VU with 1×64 computation units, 1MB of L1 SRAM, and 1MB accumulator. We set the off-chip memory to HBM2e with a capacity of 80GB and a bandwidth of 2TB/s. We set the interconnection network as the crossbar that connects multiple PE tiles and L2 SRAM. The interconnection network supports broadcasting. The specifications of the baseline architecture are similar to those of A100 GPU [19]. Our simulator operates similarly to Timeloop [70].

We used BERT-Large [22] as our target transformer model. We set the SL size to 128, 512, and 2048, values that are used in BERT-Large and Megatron-LM [64]. The total batch size was set to 4096, which is mainly used in the latest DNN training methods [32, 101].

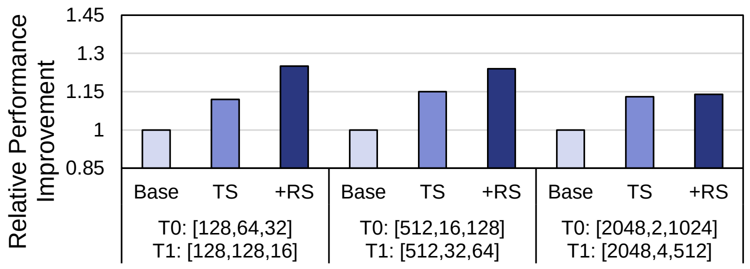


Figure 5.4: Relative performance improvement with the same SL during BERT-Large training. We incrementally apply traffic shaping (TS) and resource sharing (RS) over the baseline (Base). The task is expressed as [SL, mini-batch, gradient accumulation step].

5.2.2 Performance evaluation

We demonstrate the performance efficiency of our load balancing techniques for the target BERT-Large models. We quantify the performance improvements in detail by incrementally applying traffic shaping (TS) and resource sharing (RS). To apply the load balancing techniques, we grouped the baseline architecture into two clusters. We set the total batch size for one weight update to 4096. Task 0 (T0) and task 1 (T1) are allocated to clusters 0 and 1, respectively. Each task is expressed as [SL, mini-batch, gradient accumulation step]. The total batch size is the sum of (mini-batch × gradient accumulation step) of T0 and T1. We determine the mini-batch size of two tasks by considering the DRAM capacity of the accelerator. We make the following key observations.

The performance improvement of traffic shaping and resource sharing is closely related to the time proportion of memory- and compute-intensive operations. Figure 5.4 shows the relative performance improvement of traffic shaping and resource sharing with the same SL over the baseline architecture. Traffic shaping achieves a performance improvement by alleviating the memory bandwidth demands of BMM and memory-intensive operations. In contrast, resource sharing reduces the execution time of GEMM operations by using MU more actively than traffic shaping. In the 128 SL case, GEMM operations account for 60% of the total execution time, hence, the performance improvement of traffic shaping (1.12×) is less than that of resource sharing (1.13×). In the 512 SL case, as the execution time for GEMM, BMM and memory-intensive operation is 40%, 10% and 50% of the total time, respectively, the performance improvement by traffic shaping (1.15×) is greater than that by resource sharing (1.09×).

The 2048 SL case shows only a slight difference in the performance improvement between traffic shaping and resource sharing. The execution time of the BMM and memory-intensive operations occupies about 80% of the total in the 2048 SL case. Traffic shaping reduces the execution time by distributing the DRAM access demands of BMM and memory-intensive operations to 20% of the GEMM operations. Hence, most GEMM operations are utilized for traffic shaping. If resource sharing reduces the execution time of GEMM operations, there is a trade-off in that the effectiveness of traffic shaping is reduced. Therefore, there is little difference in the results. Overall, our load balancing techniques are shown to improve the performance by up to 1.25× at all SLs.

The more two tasks have different computational characteristics, the more the proposed techniques improve the performance. Figure 5.5 shows the relative performance improvement of traffic shaping and resource sharing with different SLs. In the 2048 maximum SL case, the total performance improvement is small if two tasks use the same SL size (shaded in blue in the figure). In contrast, the combination of 128 and 2048 SLs results in the best performance improvement,

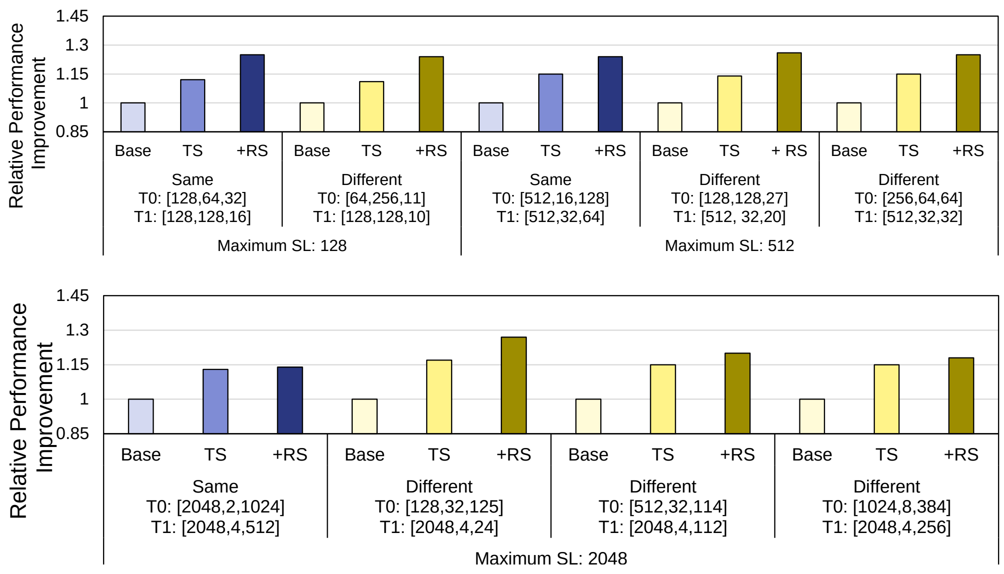


Figure 5.5: Relative performance improvement with different SLs. We incrementally apply traffic shaping (TS) and resource sharing (RS) the over baseline (Base). The task is expressed as [SL, mini-batch, gradient accumulation step]. Blue represents the results of the same SL, and yellow denotes the results of different SLs.

as BMM and memory-intensive operations in the 2048 SL case and the GEMM operations in the 128 SL case are likely to be performed concurrently. Therefore, our technique improves the performance by up to 1.27×.

In contrast, the more similar the computational characteristics of the tasks, the less effective the proposed techniques are. In the 128 maximum SL case, the combination of 64 and 128 SLs has a low performance improvement compared to the use of the same SL size. This occurs because the 64 SL case requires a longer execution time for GEMM operations compared to the 128 SL case, and the 128 SL case already has a sufficiently high proportion set for GEMM operations. The 512 maximum SL case shows a high performance improvement regardless of whether the same or different SLs are used. Overall, 128 and 512 maximum SLs improve the performance by up to 1.26×.

Chapter 6

Discussion

Places to handle DW-CONV: Due to the special computational characteristics of DW-CONV, modern accelerators have been proposed to handle DW-CONV in various places. These accelerators have their own pros and cons. We classify the locations where conventional accelerators process DW-CONV and describe the characteristics of each location. Figure 6.1 shows where the conventional accelerators process DW-CONV. This figure further subdivides the optimization methods about a matrix unit (MU).

① Processing DW-CONV in matrix unit: because DW-CONV is included as a type of CONV, most accelerators process DW-CONV in the MU. The representative accelerators that process DW-CONV in MU are TPU series [49,50] (we call this the baseline). As the baseline utilizes systolic execution to maximize feature maps and weight parameters, the MU suffers from under-utilizations when processing DW-CONV which has low reusability. To process DW-CONV efficiently in the MU, the latest works [9, 15, 98, 102] increase on-chip memory bandwidth from an L1 SRAM to the MU. [9, 15, 98, 102] provide $\mathcal{O}(N^2)$ on-chip memory bandwidth to efficiently process both CONV and DW-CONV. The advantage of this structure is that the MU can be used to pro-

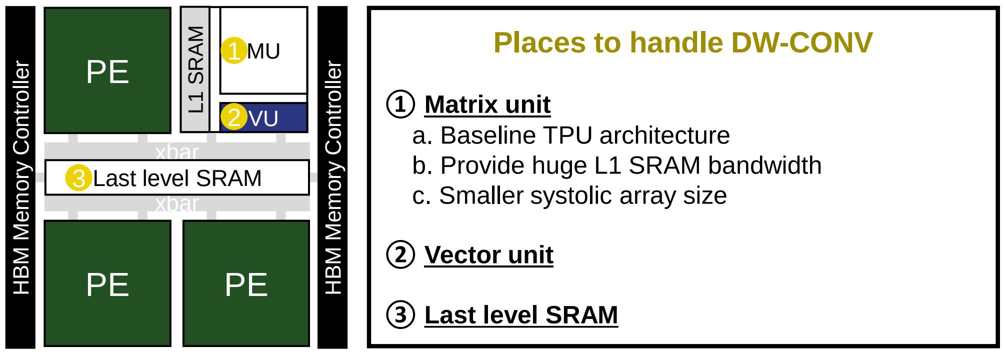


Figure 6.1: Classification of places handling DW-CONV in modern DNN accelerators. We further subdivide the optimization methods about a matrix unit (MU).

cess various operations. For example, this can also process batched GEMM that requires a lot of on-chip memory bandwidth. The disadvantage of this structure is that on-chip memory bandwidth is required as squared as the baseline. Considering that most operations in the CNN model are CONV, the accelerators may underutilize on-chip memory bandwidth most of the time. [15] takes less than $\mathcal{O}(N^2)$ on-chip memory bandwidth by allocating bulk register files for MAC units. However, the area overhead of the entire chip becomes larger than the baseline due to the addition of the register files [56].

FAST [104] improves the performance of DW-CONV by reducing the height and width size of the systolic array and increasing the number of PEs.

Figure 6.2 compares the baseline and FAST architectures when processing DW-CONV. The baseline has a systolic array of 128x128 per PE. Among 65,536 MAC units based on 4 PEs, 512 units are actually operating, and MAC utilization is 0.78%. In contrast, The FAST architecture has a systolic array of 32x32 per PE. Among 65,536 MAC units based on 64 PEs, 2,048 units are actually operating, and MAC utilization is 3.125%. Therefore, the FAST archi-

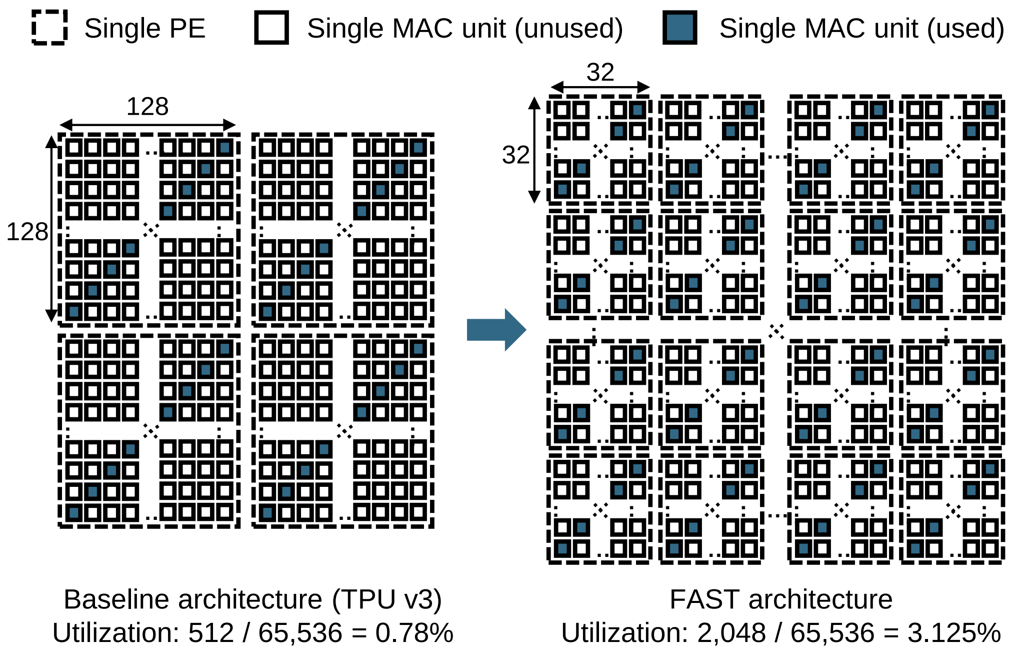

Single PE
Single MAC unit (unused)
Single MAC unit (used)
128
128
32
32
Baseline architecture (TPU v3)
Utilization: 512 / 65,536 = 0.78%
FAST architecture
Utilization: 2,048 / 65,536 = 3.125%

Figure 6.2: Comparison of baseline TPU v3 and FAST architectures when processing DW-CONV

tecture increases utilization 4 times compared to the baseline. The advantage of the FAST architecture is that it can increase the performance without changing the structure of the MU. However, a small systolic array size requires more last level SRAM traffic. Because the height and width of the systolic array are T_{IC} and T_{OC}, respectively, the smaller height and width reduces the tiling size. Therefore, the FAST architecture increases the burden on the last level SRAM.
② Processing DW-CONV in vector unit: MVP and ENLIGHT NPU [69] increase efficiency of processing DW-CONV through a customized vector unit (VU-DW). VU-DW can hide the execution time of DW-CONV in CONV through pipelined execution. In addition, it reduces on/off-chip memory access that consumes lots of energy. However, due to the characteristics of dedicated

hardware, VU-DW cannot process other operations. If the use of DW-CONV is reduced by the change of the trend of the CNN model, the VU-DW may not be needed. ③ Processing DW-CONV in last level SRAM: simba [78] processes DW-CONV in the global buffer placed on the chiplet. Because VU-DW has a small temporal buffer, the previously processed and stored edge data in the last level must be read back when performing the sliding window. Stall occurs in the process of re-reading data, further reducing performance. In contrast, the global buffer stores all feature maps for processing DW-CONV. Therefore, Processing DW-CONV in last level SRAM has a performance gain compared to VU-DW. However, the method is disadvantageous in terms of energy. As shown in Eyeriss [14], the ratio of normalized energy of local buffer and global buffer differs by several times. This method requires higher energy than processing in VU-DW because data is read from the last level SRAM for every operation.

Using MVP for CNN training: We designed MVP to process in a pipelined manner using the characteristic that intermediate feature maps in inference are discarded after they are used only once. In contrast, to perform the backward pass, training requires the intermediate feature map generated in the forward pass. Thus, the training is hard to apply in a pipelined manner. In addition, batch normalization, which is often used in CNN, only needs to process element-wise operation in inference, but it needs to obtain mean and variance in training. Therefore, there is a dependency between layers, so training cannot utilize the MVP pipeline.

Use case for load balancing technique: Our technique can be used to search for optimal execution time according to various component configurations in

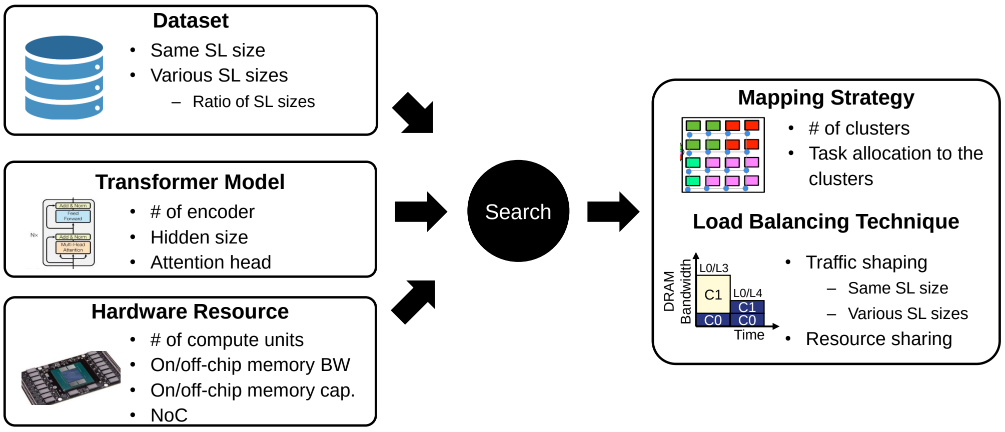


Figure 6.3: Use case for load balancing technique.

training (see Figure 6.3). One example of a component is a dataset. The input includes information on whether the dataset consists of sequence lengths of the same size or different sizes, and what percentage of each sequence length occupies when composed of sequence lengths of different sizes. A transformer model can also be a component. The transformer model uses the number of encoders, hidden size, and attention head as input parameters. If the parameter changes, the operation characteristics of the layers vary significantly. Finally, the configuration of hardware resources can also be used as an input to obtain the optimal execution time. Through a search using this information, we can find out which mapping and load balancing techniques are needed for optimal training time.

Chapter 7

Conclusion

Advances in both hardwares and DNN algorithms have created new challenges. In CNN, the change to reduce the computational amount of the model broke the compatibility with the existing CNN accelerators. In Transformer, the execution time of memory-intensive operations is no longer negligible. In this dissertation, we propose a novel DNN accelerator and load balancing techniques to address the existing challenges.

We have proposed an MVP architecture composed of Matrix (MU), Vector (VU), and Processing-near-memory units (PNMU) to effectively accelerate the inference of the latest CNN models. As opposed to the conventional area-efficient accelerators which mainly focus on convolution (CONV) with high arithmetic intensity (e.g., point-wise CONV), MVP effectively supports both compute- and memory-intensive layers. We significantly reduce the execution time of depth-wise CONV by adding a high-bandwidth low-capacity register file (called DWIB) in the VU of the baseline systolic-array architecture to

This chapter is based on [59].
"MVP: An Efficient CNN Accelerator with Matrix, Vector, and Processing-Near-Memory Units", ©2022 by Sunjung Lee and Jaewan Choi and Wonkyung Jung and Byeongho Kim and Jaehyun Park and Hweesoo Kim and Jung Ho Ahn, is licensed under CC BY 4.0. https://doi.org/10.1145/3497745.

complement the low-bandwidth high-capacity unified buffer (UB). This design enables the pipelined execution of point-wise CONV and depth-wise CONV, further improving the performance. Also, a lightweight PNMU added to UB substantially reduces the execution time of squeeze and excitation, a critical building block of the latest CNN models. Our evaluation shows that MVP improves the performance of EfficientNet-B0/B4/B7, MnasNet, and MobileNet-V1/V2 by 2.6× and total energy by 47% on average while incurring only a 9% area overhead compared to the baseline.

We also have proposed load balancing techniques to alleviate temporal fluctuations in DRAM bandwidth and to efficiently utilize computation units. As opposed to the conventional training methodologies which operate all processing elements in a single chip synchronously, our proposed techniques partition multiple processing units into clusters and process multiple tasks in each cluster asynchronously. Traffic shaping reduces the execution time of the memory-intensive operations by alleviating DRAM bandwidth fluctuations during a weight update. Resource sharing processes matrix units and vector units of all clusters simultaneously when the compute- and memory-intensive operations are performed concurrently on different clusters, reducing the execution time of the compute-intensive operations. Our evaluation shows that proposed load balancing techniques improve the performance of BERT-Large up to 1.27× compared to the baseline tiled architecture.

REFERENCES

[1] "Wafer-scale deep learning," in IEEE Hot Chips 31 Symposium, 2019, pp. 1–31.

[2] D. Abts, J. Ross, J. Sparling, M. Wong-VanHaren, M. Baker, T. Hawkins, A. Bell, J. Thompson, T. Kahsai, G. Kimmell, J. Hwang, R. Leslie-Hurd, M. Bye, E. Creswick, M. Boyd, M. Venigalla, E. Laforge, J. Purdy, P. Kamath, D. Maheshwari, M. Beidler, G. Rosseel, O. Ahmad, G. Gagarin, R. Czekalski, A. Rane, S. Parmar, J. Werner, J. Sproch, A. Macias, and B. Kurtz, "Think fast: A tensor streaming processor (tsp) for accelerating deep learning workloads," in ACM/IEEE 47th Annual International Symposium on Computer Architecture, ser. ISCA, 2020, pp. 145–158.

[3] Andes, "Andes infuses into artificial intelligence," http://www.andestech.com, 2020.

[4] S. Anginzi, Z. He, and D. Fan, "DIMA: A Depthwise CNN In-Memory Accelerator," in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ser. ICCAD, 2018.

[5] S. Anginzi, Z. He, A. S. Rakin, and D. Fan, “CMP-PIM: An Energy-Efficient Comparator-based Processing-In-Memory Neural Network Accelerator,” in Proceedings of the ACM/ESDA/IEEE Design Automation Conference, ser. DAC, 2018.

[6] Apple, “On-device Deep Neural Network for Face Detection,” https://machinelearning.apple.com/research/face-detection, 2017.

[7] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, “MCM-GPU: Multi-chip-module GPUs for continued performance scalability,” in Proceedings of the 44th ACM/IEEE International Symposium on Computer Architecture, ser. ISCA, 2017.

[8] E. Baek, D. Kwon, and J. Kim, “A multi-neural network acceleration architecture,” in ACM/IEEE 47th Annual International Symposium on Computer Architecture, ser. ISCA, 2020, pp. 940 – 953.

[9] L. Bai, Y. Zhao, and X. Huang, “A CNN Accelerator on FPGA Using Depthwise Separable Convolution,” IEEE Transactions on Circuits and Systems II, 2018.

[10] P. Bannon, G. Venkataramanan, D. D. Sarma, and E. Talpes, “Computer and Redundancy Solution for the Full Self-Driving Computer,” in IEEE Hot Chips Symposium, ser. HCS, 2019.

[11] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The Long-Document Transformer,” arXiv:2004.05150, 2020.

[12] J. Burgess, “RTX on—The NVIDIA Turing GPU,” Micro, IEEE, vol. 40, no. 2, pp. 36–44, 2020.

[13] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning,” in Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS, 2014.

[14] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” in IEEE International Solid-State Circuits Conference, ser. ISSCC, 2016.

[15] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 292–308, 2019.

[16] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “cuDNN: Efficient Primitives for Deep Learning,” 2014. [Online]. Available: https://arxiv.org/abs/1410.0759

[17] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, “McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge,” IEEE Access, vol. 8, pp. 135 223–135 243, 2020.

[18] F. Chollet, “Xception: Deep Learning With Depthwise Separable Convolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2017.

[19] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, “NVIDIA A100 Tensor Core GPU: Performance and Innovation,” Micro, IEEE, vol. 41, no. 2, pp. 29–35, 2021.

[20] J. Choquette, O. Giroux, and D. Foley, “Volta: Performance and Programmability,” Micro, IEEE, vol. 38, no. 2, pp. 42–52, 2018.

[21] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, “Rethinking Attention with Performers,” arXiv:2009.14794, 2020.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv:1810.04805, 2018.

[23] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, “RepVGG: Making VGG-style ConvNets Great Again,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2021.

[24] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “ShiDianNao: Shifting Vision Processing Closer to the Sensor,” in Proceedings of the 42nd ACM/IEEE International Symposium on Computer Architecture, ser. ISCA, 2015.

[25] Y. Fu, E. Bolotin, N. Chatterjee, D. Nellans, and S. W. Keckler, "GPU Domain Specialization via Composable On-Package Architecture," ACM Transactions on Architecture and Code Optimization, vol. 19, no. 4, 2022.

[26] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS, 2017.

[27] Gartner, "Gartner Highlights 10 Uses for AI-Powered Smartphones," https://www.gartner.com/en/newsroom/press-releases/2018-03-20-gartner-highlights-10-uses-for-ai-powered-smartphones, 2020.

[28] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2019, p. 151–165.

[29] Google, "BERT: Out-of-memory issues," https://github.com/google-research/bert, 2018.

[30] Google, "Edge TPU," https://cloud.google.com/edge-tpu, 2018.

[31] Google, "Pixel 4 is here to help," https://blog.google/products/pixel/pixel-4/, 2019.

[32] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," arXiv:1810.04805, 2018.

[33] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Wire-Aware Architecture and Dataflow for CNN Accelerators," in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2019.

[34] S. Gupta and B. Akin, "Accelerator-aware Neural Network Design using AutoML," in Proceeding of the Conference on Machine Learning and Systems, ser. MLSys, 2020. [Online]. Available: https://arxiv.org/abs/2003.02838

[35] M. Hall and V. Betz, "From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation," in Proceeding of the IEEE International Conference on Field-Programmable Technology, ser. ICFPT, 2020.

[36] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, and D.-K. Jeong, "A3: Accelerating attention mechanisms in neural networks with approximation," in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2020, pp. 328–341.

[37] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-

Attention Mechanism in Neural Networks," in ACM/IEEE 48th Annual International Symposium on Computer Architecture, ser. ISCA, 2021.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2016.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. ECCV, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[40] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[41] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[42] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2018.

[43] C.-T. Huang, Y.-C. Ding, H.-C. Wang, C.-W. Weng, K.-P. Lin, L.-W. Wang, and L.-D. Chen, "Ecnn: A block-based and highly-parallel cnn accelerator for edge inference," in Proceedings of the 52nd Annual

IEEE/ACM International Symposium on Microarchitecture, ser. MI-CRO, 2019, p. 182–195.

[44] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2017.

[45] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proceedings of the International Conference on Machine Learning, ser. ICML, 2015. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[46] G. James, D. Witten, T. Hastie, and R. Tibshirani, An introduction to statistical learning. Springer, 2013, vol. 112.

[47] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the Graphcore IPU Architecture via Microbenchmarking," 2020. [Online]. Available: https:// arXiv:1912.03413

[48] N. P. Jouppi, D. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten Lessons From Three Generations Shaped Google's TPUv4i," in Proceedings of the 48th ACM/IEEE International Symposium on Computer Architecture, ser. ISCA, 2021.

[49] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, "A Domain-Specific Supercomputer for Training Deep Neural Networks," Communications of the ACM, 2020.

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

[51] D. Jung, S. Lee, W. Rhee, and J. Ahn, "Partitioning compute units in cnn acceleration for statistical memory traffic shaping," IEEE Computer Architecture Letters, vol. 17, no. 1, pp. 72–75, 2018.

[52] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," in IEEE International Symposium on High Performance Computer Architecture, 2022.

[53] B. Kim, J. Chung, E. Lee, W. Jung, S. Lee, J. Choi, J. Park, M. Wi, S. Lee, and J. Ahn, "MViD: Sparse Matrix-Vector Multiplication in Mobile

DRAM for Accelerating Recurrent Neural Networks," IEEE Transactions on Computers, vol. 69, pp. 955–967, 2020.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[55] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach," in Proceedings of the 52st Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2019.

[56] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS, 2018.

[57] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

[58] S. Lee, H. Cho, Y. H. Son, Y. Ro, N. S. Kim, and J. Ahn, "Leveraging Power-Performance Relationship of Energy-Efficient Modern DRAM Devices," IEEE Access, vol. 6, pp. 31 387–31 398, 2018.

[59] S. Lee, J. Choi, W. Jung, B. Kim, J. Park, H. Kim, and J. Ahn, “MVP: An Efficient CNN Accelerator with Matrix, Vector, and Processing-Near-Memory Units,” ACM Transactions on Design Automation of Electronic Systems, 2022.

[60] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, “SmartShuttle: Optimizing Off-Chip Memory Accesses for Deep Learning Accelerators,” in Design, Automation and Test in Europe Conference, ser. DATE, 2018.

[61] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” in Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2009.

[62] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, “Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper,” in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2021, pp. 789–801.

[63] H. Liao, J. Tu, J. Xia, and X. Zhou, “A scalable unified architecture for neural network computing from nano-level to high performance computing,” in IEEE Hot Chips Symposium, ser. HCS, 2019.

[64] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, “Efficient Large-Scale Language Model

Training on GPU Clusters Using Megatron-LM," arXiv:2104.04473, 2021.

[65] NVIDIA, "The nvidia deep learning accelerator," in IEEE Hot Chips Symposium, ser. HCS, 2018.

[66] NVIDIA, "NVIDIA Nsight Systems," https://developer.nvidia.com/nsight-systems, 2021.

[67] NVIDIA, "Language Datasets and Data Loaders," https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/ LanguageModeling/BERT/lddl, 2022.

[68] NVIDIA, "NVIDIA H100 Tensor Core GPU Architecture," https://resources.nvidia.com/ en-us-tensor-core/ gtc22-whitepaper-hopper, 2022.

[69] Opendeges, "Opendeges ENLIGHT NPU," https://www.openedges.com/npu, 2022.

[70] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, ser. ISPASS, 2019.

[71] H. Pham, Z. Dai, Q. Xie, M.-T. Luong, and Q. V. Le, "Meta Pseudo Labels," 2020. [Online]. Available: https://arxiv.org/abs/2003.10580

[72] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, “Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training,” in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2020, pp. 58–70.

[73] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving Language Understanding by Generative Pre-Training,” 2018.

[74] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized Evolution for Image Classifier Architecture Search,” in Proceedings of the Conference on Association for the Advancement of Artificial Intelligence, ser. AAAI, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014780

[75] J. Ross and A. E. Pheps, “Computing Convolutions Using a Neural Network Processor,” 2015, US Patent App. 62/164,902.

[76] J. Ross and G. M. Thorson, “Rotating Data for Neural Network Computations,” 2015, US Patent App. 62/164,908.

[77] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2018.

[78] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, “Simba:

Scaling deep-learning inference with multi-chip-module-based architecture," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2019, p. 14–27.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proceedings of the International Conference on Learning Representations, ser. ICLR, 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[80] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "An 11.5TOPS/W 1024-MAC Butterfly Structure Dual-Core Sparsity-Aware Neural Processing Unit in 8nm Flagship Mobile SoC," in Proceedings of the IEEE International Solid-State Circuits Conference, ser. ISSCC, 2019.

[81] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Accpar: Tensor partitioning for heterogeneous deep learning accelerators," in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2020, pp. 342–355.

[82] statista, "Forecast number of mobile users worldwide from 2020 to 2024," https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/, 2020.

[83] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," in Proceedings of the IEEE International Conference on Computer Vision, ser. ICCV, 2017.

[84] C. Szegedy, W. Liu, Y. jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper With Convolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2015.

[85] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2016.

[86] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2019.

[87] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[88] S. Thoziyoor, J. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, “A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies,” in Proceedings of the 35th ACM/IEEE International Symposium on Computer Architecture, ser. ISCA, 2008.

[89] H. Touvron, A. Vedaldi, M. Douze, and H. Jegou, “Fixing the train-test resolution discrepancy: FixEfficientNet,” 2020. [Online]. Available: https://arxiv.org/abs/2003.08237

[90] TSMC, “TSMC Details 3nm Process Technology: Full Node Scaling for 2H22 Volume Production,” https://www.anandtech.com/show/16024/tsmc-details-3nm-process-technology-details-full-node-scaling-for-2h22, 2022.

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[92] R. Venkatesan, Y. S. Shao, B. Zimmer, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, “A 0.11 PJ/OP, 0.32-128 Tops, Scalable Multi-Chip-Module-Based Deep Neural Network Accelerator Designed with A High-Productivity vlsi Methodology,” in IEEE Hot Chips Symposium, ser. HCS, 2019.

[93] H. Wang, Z. Zhang, and S. Han, “Spatten: Efficient sparse attention architecture with cascade token and head pruning,” in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2021, pp. 97–110.

[94] O. Wechsler, M. Behar, and B. Daga, “Spring hill (nnp-i 1000) intel’s data center inference chip,” in IEEE Hot Chips Symposium, ser. HCS, 2019.

[95] H.-N. Wu and C.-T. Huang, "Data Locality Optimization of Depth-wise Separable Convolutions for CNN Inference Accelerators," in Design, Automation and Test in Europe Conference and Exhibition, ser. DATE, 2019.

[96] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with Noisy Student improves ImageNet classification," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2020.

[97] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2017.

[98] R. Xu, S. Ma, Y. Wang, and Y. Guo, "CMSA: Configurable Multi-directional Systolic Array for Convolutional Neural Networks," in Proceedings of the IEEE Conference on Computer Design, ser. ICCD, 2020.

[99] A. Yang, "Deep learning training at scale spring crest deep learning accelerator," in IEEE Hot Chips Symposium, ser. HCS, 2019.

[100] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A Unified MIMD-SIMD Acceleration for Generative Adversarial Networks," in Proceedings of the 45th ACM/IEEE International Symposium on Computer Architecture, ser. ISCA, 2018.

[101] Y. You, J. Li, J. Hseu, X. Song, J. Demmel, and C. Hsieh, "Reducing BERT pre-training time from 3 days to 76 minutes," 1904.00962, 2019.

[102] Y. Yu, T. Zhao, K. Wang, and L. He, “Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks,” in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA, 2020.

[103] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, “Big Bird: Transformers for Longer Sequences,” arXiv:2007.14062, 2020.

[104] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, “A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators,” in Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS, 2022.

[105] Z. Zhang, H. Wang, S. Han, and W. J. Dally, “Sparch: Efficient architecture for sparse matrix multiplication,” in IEEE International Symposium on High Performance Computer Architecture, ser. HPCA, 2020, pp. 261–274.

[106] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR, 2018.

국문초록

딥 뉴럴 네트워크(DNN)는 인간에 근접한 인식 정확도를 토대로 이미지 분류, 자연어 처리, 음성 인식과 같은 다양한 분야에서 사용된다. DNN의 계속된 발전으로 인해, DNN에서 가장 많은 연산량을 요구하는 컨볼루션과 행렬 곱셈(GEMM)을 전용으로 처리하는 가속기들이 출시되었다. 하지만, 컴퓨팅 집약적인 연산들을 가속하는데에만 치중된 가속기 연구 방향으로 인해, 이전에는 잘 보이지 않았던 메모리 집약적인 연산들의 수행 시간 비중이 증가하였다.

컨볼루션 뉴럴 네트워크 추론(CNN inference)에서, 컨볼루션의 연산 비용을 줄이기 위해 최신 CNN 모델들은 깊이방식의 컨볼루션(depth-wise convolution, DW-CONV)과 스퀴즈-엑사이테이션(squeeze-and-excitation, SE)을 채택한다. 그러나, 기존의 CNN 가속기는 컴퓨팅 집약적인 표준 컨볼루션 계층에 최적화되었기 때문에, 데이터 재사용이 제한된 DW-CONV 및 SE는 연산의 효율성을 떨어뜨린다. 따라서, DW-CONV 및 SE의 연산량은 전체 연산의 10%만 차지하지만 시스토릭 어레이(systolic-array) 기반의 가속기에서 메모리 대역폭의 병목으로 인해 처리 시간의 60% 이상을 소비한다.

트랜스포머 학습(transformer training)에서, GEMM의 수행시간이 상대적으로 감소함에 따라 소프트맥스(softmax), 레이어 정규화(layer normalization), GeLU, 컨텍스트(context), 어텐션(attention)과 같은 메모리 집약적인 연산들의 수행 시간 비중이 증가하였다. 특히, 입력 데이터의 시퀀스 길이(sequence length)

가 증가하는 최신의 트랜스포머 추세로 인해 시퀀스 길이에 따라 데이터 크기가 제곱배가 되는 소프트맥스, 컨텍스트(context), 어텐션(attention) 레이어들의 영향도가 커진다. 따라서, 메모리 집약적인 특성을 가진 연산들이 최대 80%의 수행시간을 차지한다.

본 논문에서, 우리는 CNN을 가속하기 위해 시스토릭 어레이 기반 아키텍처 위에 작은 영역 오버헤드로 컴퓨팅 및 메모리 집약적 작업을 모두 효율적으로 처리하는 연산 유닛을 추가한 MVP 아키텍처를 제안한다. 우리는 높은 메모리 대역폭 요구 사항을 충족하기 위해 곱셈기, 덧셈 트리(adder tree), 다중의 다중-뱅크 버퍼를 포함한 DW-CONV 처리에 맞춤화된 벡터 유닛(vector unit)을 제안한다. 또한, 우리는 시스토릭 어레이에서 사용하는 통합 버퍼를 확장하여 SE와 같은 요소단위(element-wise) 연산을 뒤따르는 CONV와 파이프라인(pipeline) 방식으로 처리하는 프로세싱-니어-메모리 유닛(processing-near-memory-unit, PNMU)을 제안한다. MVP 구조는 베이스라인(baseline) 시스토릭 어레이 아키텍처에 비해 9%의 면적 오버헤드만을 이용하여 EfficientNet-B0/B4/B7, MnasNet 및 MobileNet-V1/V2에 대해 성능을 평균 2.6배 향상하고 에너지 소모량을 47% 줄인다.

그리고, 우리는 트랜스포머 학습 가속을 위해 DNN 가속기 내에 존재하는 여러 개의 연산 유닛들을 클러스터(cluster) 단위로 분할하는 기술들을 제안한다. 트래픽 성형(traffic shaping)은 클러스터들을 비동기 방식으로 수행시켜 DRAM 대역폭의 출렁임을 완화시킨다. 자원 공유(resource sharing)는 컴퓨팅 집약적인 연산과 메모리 집약적인 연산이 서로 다른 클러스터에서 동시에 수행될 때 모든 클러스터의 매트릭스 유닛과 벡터 유닛을 동시에 수행 시켜 컴퓨팅 집약적인 연산의 수행 시간을 줄인다. 트래픽 성형과 자원 공유를 적용하여 BERT-Large 학습 수행시 1.27배의 성능을 향상시킨다.

주요어: DNN 가속기, 맞춤화된 벡터 유닛, 컨볼루션 뉴럴 네트워크, 부하 균형, 연산 유닛 분할, 트랜스포머, 트래픽 성형, 자원 공유

학번: 2018-38972