IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 February 2024, accepted 10 March 2024, date of publication 25 March 2024, date of current version 29 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3380433

RESEARCH ARTICLE

SAVector: Vectored Systolic Arrays

SANGUN CHOI[1], SEONGJUN PARK[2], JAEYONG PARK[1], JONGMIN KIM[1], GUNJAE KOO[3], (Member, IEEE), SEOKIN HONG[4], (Member, IEEE), MYUNG KUK YOON[5], (Member, IEEE), AND YUNHO OH[1], (Member, IEEE)

[1]School of Electrical Engineering, Korea University, Seoul 02841, Republic of Korea
[2]Department of Semiconductor Systems Engineering, Korea University, Seoul 02841, Republic of Korea
[3]Department of Computer Science and Engineering, Korea University, Seoul 02841, Republic of Korea
[4]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea
[5]Department of Computer Science and Engineering, Ewha Womans University, Seoul 03860, Republic of Korea

Corresponding authors: Yunho Oh (yunho_oh@korea.ac.kr) and Myung Kuk Yoon (myungkuk.yoon@ewha.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grants funded by Korea government (Ministry of Science and ICT, MSIT) (NRF-2021R1C1C1012172), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub), and Samsung Electronics Co., Ltd (IO230419-05997-01).

ABSTRACT Conventional DNN inference accelerators are designed with a few (up to four) large systolic arrays. As such a scale-up architecture often suffers from low utilization, a scale-out architecture, in which a single accelerator has tens of pods and each pod has a small systolic array, has been proposed. While the scale-out architecture is promising, it still incurs increasing off-chip memory access as the pods are supposed to access the duplicate tiles of tensors. Prior work has proposed a shared buffer structure to address the problem, but those architectures suffer from performance degradation due to shared buffer access latency. We make an observation that all the pods access the same rows of input and weights within a short time window. With the observation, we propose a new inference accelerator architecture, called Vectored Systolic Arrays (SAVector). SAVector consists of a new two-level on-chip buffer architecture and a tensor tile scheduling technique. In the new buffer architecture, global buffers are shared by all the pods and they keep the rows shared by the pods. And each pod has a tiny dedicated buffer. SAVector monitors the memory access behavior and timely determines to prefetch the data and flush it. In our evaluation, SAVector exhibits a very similar off-chip memory access count to the scale-up architecture and achieves 52% energy-delay-product (EDP) reduction. Also, SAVector achieves 27% EDP reduction over prior work by mitigating performance degradation from global buffer access latency.

INDEX TERMS Inference accelerator, on-chip buffer, energy efficiency.

I. INTRODUCTION

Within a decade, various Deep Neural Networks (DNNs) have been proposed, and they are being employed in a myriad of services. Each DNN model has a varying number of layers, the size of tensors in each layer, and the number of tensors. For example, GPT-3, one of the popularly employed DNNs, consists of 96 layers and contains 175 billion parameters [1]. Unlike GPT-3, MobileNet has 25 layers, and its total parameters are 5.4 million [2].

In the post-Moore era, DNN hardware accelerators run inference or training with less energy than conventional CPUs or GPUs. Prior work has proposed DNN accelerators based on a design approach: a single pod offers high peak throughput, and a single accelerator equips a few (up to four) pods. Prior work refers to this architecture as a "scale-up architecture [3]." For example, Google TPUv4i consists of four pods, each featuring a 128×128 systolic array [4]. While DNN accelerators based on the scale-up architecture have been successfully employed in various systems, they still face challenges in achieving sustainable energy efficiency. One of the reasons for this challenge is that the size of a tensor is often smaller than the size of a systolic array. Google has

The associate editor coordinating the review of this manuscript and approving it for publication was Libo Huang.

reported that TPUs often suffer from underutilization (less than 10%) while running DNN workloads [5].

As a solution to the aforementioned challenge, prior work has proposed a scale-out architecture for DNN accelerators [3], [6], [7], [8], [9]. Unlike the scale-up architecture, the scale-out architecture comprises tens or even hundreds of pods, with each pod having small systolic arrays (e.g., 8×8 or 32×32). A DNN accelerator based on the scale-out architecture selectively activates pods as needed to match the demands of a target workload and carry out computations. With this approach, the scale-out architecture can enhance utilization, thereby achieving superior energy efficiency compared to the scale-up architecture.

While the scale-out architecture is promising, there still exists a challenge to improve energy efficiency and achieve sustainable scalability. The scale-out architecture increases off-chip memory access as all pods in an accelerator often access the same tiles within a short time window. As such, the total off-chip memory access count of the scale-out architecture is proportional to the number of active pods. The energy consumption of an off-chip memory access is two orders of magnitudes more significant than an on-chip SRAM buffer access [10]. To mitigate the excessive energy consumption caused by the off-chip memory access, existing DNN accelerators contain SRAM buffers dedicated to a pod. However, such an on-chip memory structure is not scalable, as an accelerator with tens or hundreds of pods would require hundreds of megabytes of SRAM resources for the dedicated buffers. Prior work has also employed a shared SRAM buffer connected with all pods [3]. Although the shared buffer effectively filters out the duplicated off-chip memory access, the prior work still suffers from performance degradation due to long buffer access latency.

We make the following observations. The pods in a scale-out accelerator access duplicate rows of input data and weights of a DNN model within a short time window. If an accelerator could capture and exploit such data locality across the pods, it could significantly reduce the number of off-chip memory accesses. A hardware buffer may be employed in an accelerator. However, simply equipping SRAM buffers that all pods share increases both hardware overhead and inference time significantly. As such, a careful microarchitecture design is required to utilize the shared buffer efficiently.

In this paper, we propose a novel microarchitecture of a scale-out DNN inference accelerator called Vectored Systolic Array (SAVector). The key concept of SAVector is the design of an on-chip buffer hierarchy. The key component of the on-chip buffer hierarchy is SRAM buffers that all the pods share, leveraging the dominant memory access pattern that we have observed. We refer to this new buffer structure as the ''global buffer.'' The size of the global buffer is smaller than the combined size of the SRAM buffers dedicated to each pod. SAVector reduces the overall SRAM overhead by 75% compared to prior work [3]. Furthermore, we observe that our design is suitable for scale-out architectures with hundreds of pods, making SAVector a promising solution for achieving DNN accelerator scalability.

SAVector incorporates two global buffers for input and weight data, both of which exhibit dominant access patterns during DNN workload execution. While the global buffer offers cost-effectiveness and scalability, it introduces a longer buffer access latency than the SRAM buffers dedicated to each pod. To address this latency, each pod of SAVector includes tiny SRAM buffers. Our analysis indicates that 1KB of dedicated buffers is sufficient to mask the latency of global buffer access. By incorporating the tiny buffers and global buffers, SAVector adopts a two-level on-chip buffer structure. To manage the two-level buffer structure, SAVector employs a software prefetching based on double buffering and a tile scheduling technique. SAVector monitors when a row in a global buffer is accessed by the pods that require it. Once all pods have accessed all rows in a global buffer, SAVector flushes the data and uses the prefetched data. Through this two-level buffer structure and its management, SAVector achieves superior energy efficiency compared to the scale-up architecture and previously proposed scale-out architectures. Also, the scheduling technique of SAVector divides tensors into tiles and assigns the tiles to pods as many as possible. Such a mechanism improves the utilization of the pods, thus resulting in better energy efficiency than prior work.

In our evaluation, SAVector achieves a 52% energy-delay-product (EDP) reduction compared to the scale-up architecture. Also, SAVector achieves 52% speedup and 27% EDP reduction compared to the prior work [3].

In this paper, we make the following contributions.

- We demonstrate that DNN inference workloads commonly exhibit a dominant memory access pattern: rows in input and weight matrices are accessed by all pods of the scale-out architecture within a short time window.
- We propose SAVector, a novel scale-out DNN inference accelerator. The key concept is a two-level on-chip buffer structure comprising global buffers and 1KB dedicated buffers.
- SAVector achieves a 52% and 27% EDP reduction compared to the scale-up architecture and the prior work [3], respectively. For reproducibility, we open-source our DNN accelerator simulator .[1]

The rest of this paper consists of the following sections. Section II explains why the scale-out architecture requires to reduce the off-chip memory access and SRAM overhead. Section III introduces the SAVector architecture. Section IV explains the experimental results. Section V explains related work. Section VI concludes this paper.

II. MOTIVATION

In this section, we first introduce the concept of scale-out Deep Neural Network (DNN) accelerators. Next, we analyze the advantages and disadvantages of the scale-out DNN accelerators, thus motivating to design SAVector.

[1] https://github.com/comsys-lab/SAVector

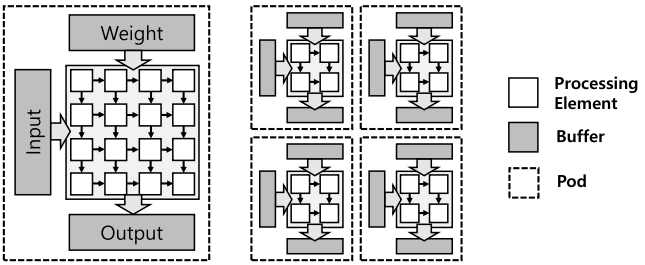


FIGURE 1. Scale-up architecture vs. scale-out architecture with systolic arrays. The scale-up architectures (left) use a single large systolic array pod to maximize data reuse across the array. The scale-out architectures (right) use multiple small systolic array pods to improve array utilization and speedup. Each pod consists of a single systolic array for computation and three SRAM-based buffers for keeping input, weight, and output.

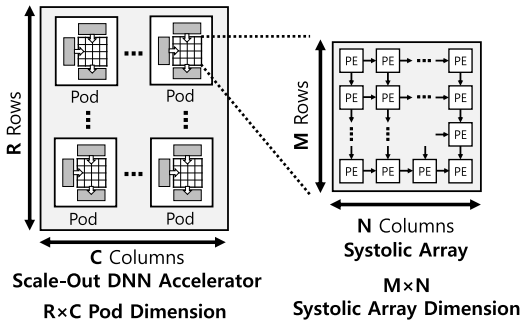


FIGURE 2. Pod dimension and systolic array dimension.

A. SCALE-UP AND SCALE-OUT DNN ACCELERATORS

In this paper, we focus on DNN inference accelerators that employ single or multiple systolic arrays, aligning with the trend seen in commercial DNN accelerators introduced by Google [4], [5] and prior work [3], [6], [7], [8], [9], [11], [12]. Based on those prior work, we assume that a pod of an inference accelerator equips a single systolic array.

DNN inference accelerators may feature either a scale-up architecture or a scale-out architecture. Figure 1 provides a visual representation of both the scale-up and scale-out architectures. The scale-up architecture consists of one or a few (less than 4) pods. With a fixed silicon budget, the scale-up architecture employs a large systolic array. For instance, the Google TPUv1 utilizes a single pod with a systolic array dimension of 256 × 256 [5]. This architecture provides a high peak throughput. However, prior work has demonstrated [4], [5], the scale-up architecture frequently experiences low utilization because DNN models often have smaller tensors than the systolic array size. Consequently, the actual performance of the scale-up architecture often falls below the peak throughput.

Unlike the scale-up architecture, the scale-out architecture comprises multiple pods. Generally, the dimension of a systolic array in the scale-out architecture is smaller than that in the scale-up architecture. Once a scale-out DNN accelerator processes a workload, it activates only the pods engaged in computation. Prior work has proposed the scale-out architectures that arrange pods in a two-dimensional array [6], [13], [14], [15]. We define this two-dimensional arrangement of pods in the scale-out architecture as the "pod dimension." Within each pod, the systolic array also consists of a two-dimensional array of processing elements (PE), which we term the "systolic array dimension." Figure 2 depicts an example that illustrates the pod dimension and systolic array dimension. In this example, an accelerator organizes pods in an array with R rows and C columns. Inside each pod, the systolic array contains a PE array with M rows and N columns. We represent the pod dimension and systolic array dimension of this accelerator as R×C and M×N, respectively.

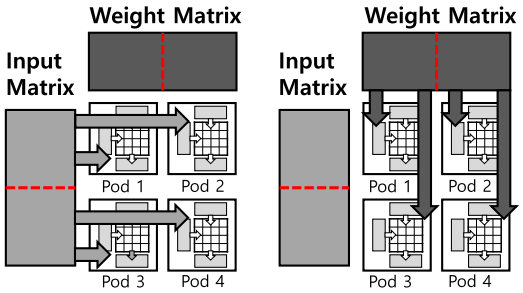


FIGURE 3. Operand matrix tiling in scale-out DNN accelerator with four pods. The input and weight operand matrix are divided into two parts. Pods in the same row read the same input (left), and pods in the same column read the same weights (right).

A DNN accelerator incorporates a hardware scheduler. The scheduler divides tensors into tiles and sends them to the pods [6]. The number of the divided tensors may be smaller than the number of pods. If so, some pods may not contain operands. In that case, the scale-out architecture deactivates the empty pods. By selectively utilizing a subset of the pods, the scale-out architecture can achieve higher utilization, thus improving energy efficiency. Also, the scale-out architecture achieves parallelism by making multiple pods simultaneously execute General Matrix Multiplication (GEMM) operations. This advantage has been a focal point in the prior work [3], [6], [14], [16].

In both the scale-up and scale-out architectures, each pod equips on-chip SRAM buffers. In this paper, we assume that each pod includes three SRAM buffers, each serving distinct purposes: an input buffer, a weight buffer (storing tensors with parameters), and an output buffer.

B. WHY SAVECTOR?

While the scale-out architecture is promising, it incurs a larger number of off-chip memory accesses than the scale-up architecture. The increased number of pods in the scale-out architecture results in a higher data movement overhead, which is the energy consumed by fetching data from off-chip memory to on-chip SRAM buffers. This redundant energy consumption is commonly known as "data movement energy overhead" and has been discussed in prior work [17], [18].

Figure 3 illustrates how the scale-out architecture carries out GEMM operations with pods arranged in a 2 × 2 configuration. We assume a system automatically creates operand matrices, data structures tailored for systolic arrays,

TABLE 1. DNN Accelerator configuration.

Number of pods (Pod dimension)	Systolic array per pod	Buffer size per pod		
		Input	Weight	Output
1 (1×1)	128×128	1.5 MB	1.5 MB	1 MB
4 (2×2)	64×64	384 KB	384 KB	256 KB
16 (4×4)	32×32	96 KB	96 KB	64 KB
64 (8×8)	16×16	24 KB	24 KB	16 KB
256 (16×16)	8×8	6 KB	6 KB	4 KB
1024 (32×32)	4×4	1.5 KB	1.5 KB	1 KB

adjusting their access order, and data allocation. DNN accelerators typically employ either of three dataflow types: input stationary, output stationary, or weight stationary. In this paper, all the accelerators, including both the baseline and our proposed design, employ the weight stationary dataflow, which is consistent with prior work [4], [5], [7], [12], [19].

In this example, the accelerator divides input and weight operand matrices into two tiles each. There are two groups of pods (Pod 1&2 and Pod 3&4) that access the same tile in input or weight. Each group independently retrieves the shared tiles from off-chip memory. Consequently, the data movement overhead increases compared to the scale-up architecture. In this example, the data movement overhead is 4× higher than in the scale-up architecture.

We conduct an analysis of the performance (inference time) and the data movement overhead for both the scale-up and scale-out architectures. To maintain consistency within a given silicon budget, we set fixed values for both the number of processing elements (PEs) and the total size of the on-chip SRAM buffers in an accelerator. Our choice is to set the number of PEs at 16,384, which aligns with the configuration of a single 128 × 128 pod in the Google TPUv4i [4]. A single TPUv4i core comprises four pods and offers 16MB of SRAM resources. In this analysis, we focus on modeling a single pod, and thus, we configure the total size of the SRAM buffers to be 4MB ($= 16MB/4$). Also, like the prior work [6], [9], [14], [15], all the scale-out architecture configurations employ a double buffering scheme. While a pod performs Multiply-Accumulate (MAC) operations, the scale-out architecture prefetches the next input or weight tile into its dedicated buffer. Such overlapped memory access and computations can hide off-chip memory access latency.

With the same silicon budget, we set five different configurations for the scale-out architecture by varying the number of pods, ranging from 1 to 1024. Table 1 describes the detailed configurations of the scale-out architecture. We model all these configurations using SCALE-Simv2 [15]. We use the scale-up architecture as the baseline. We use DNN models: Mobilenetv3-large [2], DenseNet-169 [20], ResNet-50 [21], BERT-base, BERT-large [22], and Vision Transformer (ViT) [23]. Those benchmarks include both traditional convolution neural networks and transformer-based workloads. The transformer-based models have gained significant popularity in recent applications. In this paper, by using those workloads, we offer a comprehensive evaluation of the performance and efficiency of various systems, including SAVector, across a broad spectrum of workloads.

TABLE 2. Experimental results. We calculate geometric means of speedup and EDP, arithmetic means of off-chip memory access and energy consumption. We normalize all results to those of the scale-up configuration. For off-chip memory access and energy consumption, the values in small brackets represent normalized values.

Pod dimension	Speedup	Off-chip memory access	Energy consumption	EDP
1×1	1×	27M (1×)	42.1 mJ (1×)	1×
2×2	1.04×	49M (1.79×)	52.6 mJ (1.25×)	1.2×
4×4	1.06×	93M (3.37×)	73.8 mJ (1.76×)	1.64×
8×8	1.09×	176M (6.02×)	115.2 mJ (2.74×)	2.51×
16×16	1.13×	352M (11.9×)	202.2 mJ (4.82×)	4.25×
32×32	1.2×	681M (21.79×)	344.8 mJ (8.22×)	6.83×

Table 2 shows the experimental results of inference time, off-chip memory access count, energy consumption, and Energy-Delay-Product (EDP). For the number of off-chip memory access and energy consumption, we calculate the arithmetic means of the results for all benchmarks. For speedup and EDP, we calculate the geometric means of the results for all six DNN models. We normalize those geometric mean, and the arithmetic mean results to those of the scale-up architecture. In our experiments, as the number of pods increases from 1 to 1024, the speedup on average increases up to 1.2×. With the scale-out architecture, the systolic array size decreases as the number of pods increases, resulting in a smaller peak throughput. Even if the number of pods is increased by a factor of N, the speedup may not exhibit linear growth.

As depicted in Table 2, the scale-out architecture experiences an exponential surge in off-chip memory access. With the scale-out architecture with 64 pods, it still exhibits 6.02× more off-chip memory access compared to the scale-up architecture, despite its speedup being only 1.06×. The 256 and 1024 pod configurations result in 11.9× and 21.79× more off-chip memory access compared to the scale-up architecture, respectively. Note that the 256 and 1024 pod configurations, with 8×8 and 4×4 systolic array dimensions, respectively, are not realistic designs, as such small systolic arrays do not reuse the data across the processing elements. Also, in our initial study, even the scale-out architecture with infinite-sized SRAM buffer reduces the off-chip memory access only up to 10% compared to scale-out architectures with a total 4MB SRAM buffer described in Table 1.

Given that off-chip memory access consumes two orders of magnitude more energy than computation, the overall trends mirror those seen in the results of off-chip memory access count. We conduct a study regarding the energy consumption of the scale-up and scale-out architectures. Despite the scale-out architectures achieving higher utilization than the scale-up architecture, energy consumption exhibits an exponential increase. The scale-out architecture with 64 pods

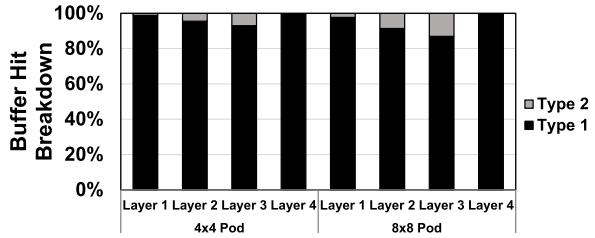


FIGURE 4. Ratio of buffer hit caused by data duplication type 1 and type 2.

consumes 115.2 mJ, which is 2.74 times more energy than the scale-up architecture.

With the speedup and energy consumption results, we calculate the EDP for all configurations. With an increase in the pod dimension, energy consumption grows faster than speedup. Therefore, scale-out architectures show up to 6.83× higher EDP than those of scale-up architecture. We observe that employing multiple downsized pods in the scale-out architecture does not lead to a reduction in the overall energy consumption and EDP of a system. With this analysis, we find that the scale-out architecture can achieve the same or better speedup, but incurs redundant energy consumption due to the excessive off-chip memory access. If an accelerator eliminates redundant memory access, it could achieve a higher energy efficiency than the scale-up architecture.

While accessing tiles, pods may access data in different rows of an input matrix as they reach the end of a particular row. Such behavior often increases off-chip memory access. We analyze prevalent memory patterns in DNN workloads. We use four input matrices derived from tensors in ResNet-50 and the 4×4 and 8×8 scale-out architectures in Table 1. Both architectures equip on-chip SRAM buffers capable of storing all necessary data. We measure the number of buffer hits and consider two types of buffer hits. We call the first type Type 1, which occurs if pods access the same row of the input matrix. We call the second type Type 2, which takes place if pods access different rows of the input matrix.

Figure 4 shows the experimental results, the ratio of data duplication types making buffer hits. Type 1 accounts for more than 90% of the total buffer hits as the pods in the same row exhibit the same data access pattern, resulting in buffer hits with every access to data in the matrix tile. Type 2 accounts for less than 10% of the total buffer hits. For Layer 4, data duplication does not occur. Therefore, if an accelerator effectively recognizes the memory access pattern involving the same row and efficiently manages the data in the buffers, there is potential for a significant reduction in off-chip memory access.

If an accelerator equips an SRAM buffer shared by all pods, the data movement overhead may be reduced. Ideally, scale-out architectures with a shared buffer consume less energy than the scale-up architecture. In our initial study, the 4×4 pod configuration with a shared buffer consumes 31% less energy than the scale-up architecture. Prior work has introduced the scale-out architecture with a shared SRAM buffer to alleviate data movement overhead [3], [4], [6].

While shared buffers alleviate data movement overhead, the extended latency associated with accessing these shared buffers leads to performance degradation. The previously proposed scale-out architectures interconnect all pods and a shared buffer via an interconnect network, which increases buffer access latency. Given that scale-out architectures typically consist of hundreds or even thousands of pods [3], [6], the latency for accessing the shared buffer can be significantly greater.

Prior work has introduced an inference accelerator architecture designed to minimize the performance impact of shared buffer access latency through the use of optimized interconnection and scheduling techniques [3]. However, these optimizations have limitations in hiding global buffer latency during computation. It has been noted that prior work often does not incorporate buffers in close proximity to the systolic array [7], [19], or they utilize buffers or registers within a pod that are too small (e.g., less than 1KB) to be effective [3], [4]. Such buffers are primarily intended to serve as intermediate storage for input rows over a short duration (e.g., 8 cycles) and are not capable of employing double-buffering strategies to mask the extensive latency associated with accessing shared buffers. In our simulations, we find that a scale-out architecture comprising 64 pods experiences a 61% decrease in performance relative to the TPUv4i when shared buffers are utilized. This increased inference time further diminishes the energy efficiency of the accelerator. This increased inference time further diminishes the energy efficiency of the accelerator. The effects of shared buffer access latency on performance are further discussed in Section IV.

The scale-out architecture enhances utilization compared to the scale-up architecture. However, the energy efficiency achieved by existing scale-out architectures remains suboptimal. On one side, scale-out architectures utilizing dedicated buffers in each pod outperform scale-up architectures in terms of performance but are hampered by data movement overhead resulting from extensive off-chip memory access. On the other side, scale-out architectures that employ a shared buffer across all pods eliminate redundant off-chip memory accesses but are adversely affected by prolonged shared buffer access latencies. As such, attaining scalability in DNN inference accelerators calls for the development of a new and more flexible buffer structure.

III. SAVECTOR: VECTORED SYSTOLIC ARRAY

In this section, we propose SAVector. We provide a detailed explanation of the SAVector architecture.

We design SAVector to achieve two primary objectives. First, SAVector aims to minimize data movement overhead caused by the duplicated tensor access across pods. Second, SAVector maintains quality-of-service during inference while scaling an accelerator with hundreds of pods. The key idea behind SAVector is to exploit the memory access

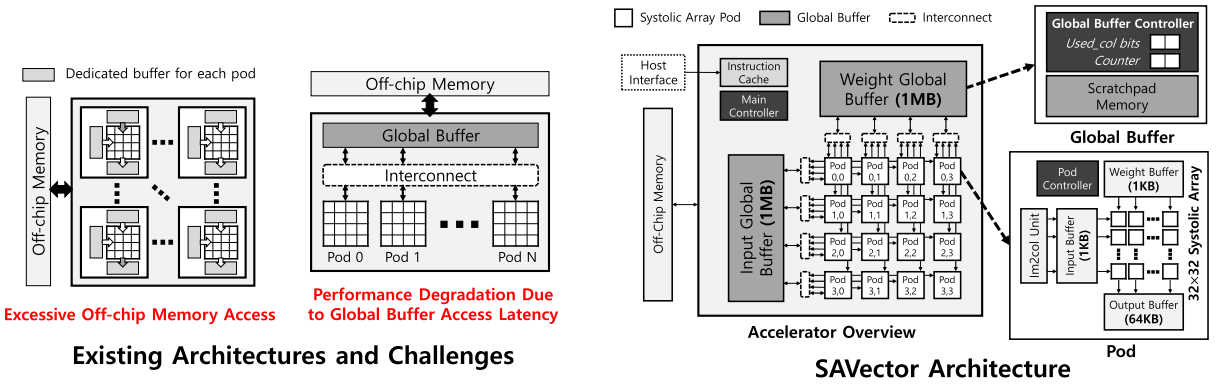


FIGURE 5. Microarchitecture overview of the existing scale-out architecture and SAVector. Existing scale-out architectures suffer from either data movement overhead due to excessive off-chip memory access (left) or performance degradation due to long global buffer access latency (right). SAVector addresses the challenges using a new matrix row-sharing policy with a two-level buffer system.

patterns inherent in the scale-out architecture, especially for Type 1 discussed in Section II-B. By leveraging these memory access patterns, SAVector contains a two-level on-chip buffer structure consisting of global buffers and tiny buffers. To resolve the redundant memory access, SAVector fetches the rows of the input and weight matrices in the global buffers and keeps them until all the corresponding pods complete accessing them. Tiny buffers within each pod mitigate performance degradation from global buffer access latency with negligible SRAM overhead. To ensure efficient inference, SAVector employs a scheduling technique that effectively utilizes the new global buffers and pods. The proposed scheme not only reduces data movement overhead but surpasses the performance of the existing scale-out architectures.

Figure 5 provides an overview of SAVector. In this paper, SAVector comprises a 4×4 array, totaling 16 pods. Each pod includes a systolic array and several hardware resources. All pods are directly connected to global buffers, each of which is equipped with a dedicated controller.

SAVector incorporates global SRAM buffers allocated outside the pods, inherently introducing longer buffer access latency than dedicated buffers. This increased latency may significantly impact inference time. To address this challenge, we design SAVector to equip each pod with tiny-sized (e.g., 1KB) dedicated SRAM buffers. These small SRAM buffers boast minimal latency (1 or 2 cycles) and efficiently store data the corresponding pod will access shortly. In scale-out architectures, including SAVector, there can be tens or even hundreds of pods within a single accelerator. In SAVector, each pod is furnished with 1KB SRAM buffers for input and weights. While effectively mitigating substantial performance degradation, the combined buffer capacity remains less than that of existing scale-out architectures. Regarding the output buffer, we determine that a buffer more significant than 1KB should be positioned near the systolic array to facilitate store operations without causing stalls. Consequently, we design a 64KB output buffer following thorough performance measurements. The SRAM resource requirement of SAVector in this paper is 3.02MB, about a quarter of the SRAM needed by existing scale-out architectures.

As mentioned in section II, scale-out architectures selectively inactivate pods to reduce power consumption if a workload generates an insufficient number of tiles. In such a case, they cannot utilize the SRAM within inactive pods, and SRAM capacity to prefetch data from off-chip memory may be insufficient. SAVector addresses this concern as follows. First, while SAVector inactivates pods, it does not turn off the global buffer banks. Therefore, active pods can fully utilize the global buffer to prefetch. Second, as each pod contains only tiny buffers, inactivating these buffers does not have a significant impact on the overall SRAM capacity in use.

SAVector requires a novel buffer management technique. Given that the groups of pods sharing a global buffer typically demonstrate the same memory access pattern, SAVector can effectively manage the buffers with a straightforward policy, a software prefetching scheme for the 1KB buffers, and a new scheduling technique. Unlike tiny buffers dedicated to a pod, adopting the double-buffering on global buffers requires careful management techniques, as multiple pods operate independently. A new global buffer management technique monitors the memory access behavior of pods exhibiting the same memory access pattern. Then, the global buffer controller precisely determines when to prefetch data and when to flush the global buffer. With those schemes, SAVector significantly reduces the overall data movement overhead while efficiently performing inference.

A. WORKFLOW EXAMPLE

Figure 6 provides an illustrative workflow example of SAVector. This example comprises six phases, denoted as P (ranging from $P = 0$ to $P = 5$). Each phase corresponds to a specific behavior within the accelerator, such as prefetching operand data into the buffer or flushing the buffer. In this particular scenario, SAVector comprises a 4×4 array of pods, with each pod housing a 4×4 systolic array. SAVector runs a straightforward DNN layer featuring input and weight

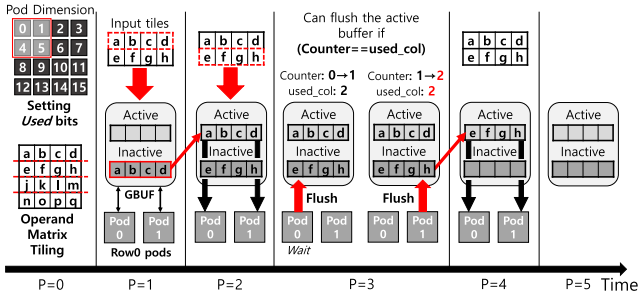


FIGURE 6. SAVector workflow example. In this example, SAVector selectively uses four pods out of 16 pods. The DNN layer has a 4 × 4 input matrix and a 4 × 4 weight matrix. Each input and weight parameter size is 1 byte, and the global buffer (GBUF) capacity is 8 bytes in total (4 bytes for the active buffer, 4 bytes for the inactive buffer).

matrices, both sized at 4 × 4. Leveraging the advantages of the scale-out architecture, SAVector selectively activates only four out of the 16 pods (2 × 2), optimizing resource utilization.

SAVector operates under a weight-stationary policy, fetching input data for each layer. In this example, we focus on the movement of the input matrix for simplicity. We assume that SAVector has 8-byte global buffers. The 1KB dedicated buffers in each pod primarily aim to alleviate stalls resulting from global buffer access. In our optimistic scenario, we assume these dedicated buffers work automatically and transparently. SAVector employs a double buffering technique, similar to existing methods in prior work. Its global buffers consist of two logical spaces: an active buffer and an inactive buffer. The active buffer stores operands for current computations, while SAVector prefetches data for the next computation phase into the inactive buffer. Each half of an SRAM buffer features a single bit that designates its role (active or inactive). In this example, the active buffer and the inactive buffer each have a capacity of 4 bytes. Furthermore, we assume that the size of an input parameter is 1 byte, allowing both buffers to hold 4 input parameters simultaneously.

1) INITIALIZION (P=0)

As the pods commence processing, SAVector maintains a count of active pods exclusively assigned to a global buffer. This information is stored in two hardware components: the SAVector scheduler and the global buffer controller. The SAVector scheduler monitors the runtime hardware status and allocates tiles to pods that are both activated and idle. The global buffer controller manages the active and inactive buffer spaces within the global buffer. Also, the global buffer controller handles the prefetching of data into the inactive buffer. In the example depicted in Figure 6, SAVector utilizes a 2 × 2 subset of the total 4 × 4 pods. The *used_row* bits and *used_col* bits will be set to 2 and 2, respectively. SAVector proceeds to divide the input matrix and weight matrix into four tiles each.

2) PREFETCHING (P=1)

With predefined sizes for active and inactive buffers (equal), SAVector initiates the prefetch of tiles whose combined size matches that of the inactive buffer. In our example, a matrix is divided into four tiles, each of which has the same size (four bytes). SAVector prefetches one tile at a time. Also, SAVector prefetches the rows of the input operand matrix tile for pods that share the same row index. Once SAVector completes the initial prefetch, the SAVector scheduler allocates the first tile to Pod 0 and 1. Figure 6 visually illustrates how SAVector prefetches data shared by two pods (Pod 0 and Pod 1) into the first row of the pod array (Row 0). In this figure, SAVector prefetches four input parameters (a, b, c, and d). For this purpose, we design a new prefetch instruction in SAVector. This instruction is incorporated at compile time and enables SAVector to perform software prefetching.

3) COMPUTATION (P=2)

Once the prefetch operation is complete, the pods initiate their computations using the prefetched data. To use the prefetched data, the global buffer controller designates an inactive buffer as the active buffer. The pods transfer operand data from the active global buffer to their respective dedicated 1KB buffers. Following this transfer, the active pods proceed with Multiply-Accumulate (MAC) operations.

4) FLUSHING (P=3)

Once all the pods that utilize the data in the active buffer complete their computations, SAVector performs a data flush. SAVector does not permanently erase the data from the active buffer. Instead, in the current phase, the roles of the inactive and active buffers are swapped by simply toggling these bits.

If a pod flushes the global buffer before other pods have utilized all the data in the active buffer, SAVector may yield incorrect computation results. To address this challenge, SAVector employs a counter in the global buffer controller. Whenever the controller receives a buffer flush signal from a pod, SAVector increments the counter value. With each increment, the controller compares the counter value to the *used_col* field, which indicates the number of pods sharing the global buffer. If the counter value matches the *used_col* bits, SAVector designates the inactive buffer as the new active buffer, effectively completing the flush phase. This mechanism ensures that data is only flushed if all pods have finished with the shared data, preventing any premature data loss that could impact computation accuracy.

In Figure 6, pod 0 and pod 1 send buffer flush signals at different cycles. Initially, the buffer flush signal from pod 0 arrives, causing the counter value to increase from 0 to 1. The controller then compares the counter value to the value in *used_col*, finding them to be different (1 and 2, respectively). SAVector waits for pod 1 to send a buffer flush signal. Upon the arrival of the buffer flush signal from pod 1, the counter value increases from 1 to 2, matching the value in *used_col*. The controller then proceeds to negate the bits in both the

active buffer and the inactive buffer. With the flushing phase completed, the pods can start their computations using the data from the newly designated active buffer. This mechanism ensures that the transition between active and inactive buffers only occurs once all pods have signified the end of their data utilization phase, guaranteeing the integrity of the computation process.

5) PROCESSING COMPLETE (P=5)

SAVector operates in iterative cycles, alternating between the computation and flushing phases until all pods finish the computation for a single DNN layer. Once the computation for a layer is complete, the global buffer controller initiates the flushing operations, following the same procedure described in phase P=3, in preparation for processing the subsequent layer. Upon completing the computation for a layer, the pods become idle, allowing the SAVector scheduler to schedule operations for the next layer. This cyclic process ensures a seamless transition from one layer to the next, optimizing the inference process.

B. ARCHITECTURAL DETAILS

1) POD

A pod comprises five components: a single systolic array, two 1KB SRAM buffers, a 64KB SRAM buffer, an Im2col unit, and a local pod controller. Each 1KB buffer stores input data and weights, respectively. The 64KB buffer stores the output data. Inspired by prior work, we incorporate a hardware block called Im2col unit that converts 4D convolutions into 2D GEMM operations near the systolic array [3], [24]. In this paper, we refer to this transformation from 4D convolution to 2D GEMM as "Im2col conversion" and the resulting 2D matrix as the "lowered matrix." The lowered input matrices often display data duplication among rows, resulting in redundant memory access, as explained in [24]. The Im2col unit plays a role in addressing this issue by preventing redundant access to the global buffer and the duplication of input data within the global buffer. The local pod controller oversees the systolic array and the three dedicated buffers.

The main advantage of the scale-out architecture over the scale-up architecture is its capacity to achieve higher utilization. To maximize accelerator utilization, determining the appropriate size for a single systolic array becomes a critical factor. Prior work has introduced a design methodology for systolic arrays within the scale-out architecture, as referenced in [3]. This methodology includes design space exploration with varying systolic array sizes to optimize effective throughput per watt. Inspired by the prior work, we conduct a similar design space exploration using the benchmarks detailed in Section II. In our initial analysis, we observe that the 32×32 systolic array size offers the best effective throughput per watt, coinciding with the outcomes presented in the prior work.

As stated earlier in this section, a pod in SAVector incorporates tiny dedicated buffers to alleviate the performance degradation from exposed stalls. We model the global buffer access latency, estimated at 11 cycles using Booksim2 [25] under the assumption that four systolic array pods share a global buffer. We also observe that 1KB dedicated buffers, assigned to input and weights, and a 64KB buffer for output proved sufficient for masking the global buffer access latency. Each dedicated output buffer in SAVector features several ports equal to the count of PEs in a row or column. In SAVector, each dedicated output buffer is equipped with 1,024 accumulators like prior work [4], [5], [12], [26].

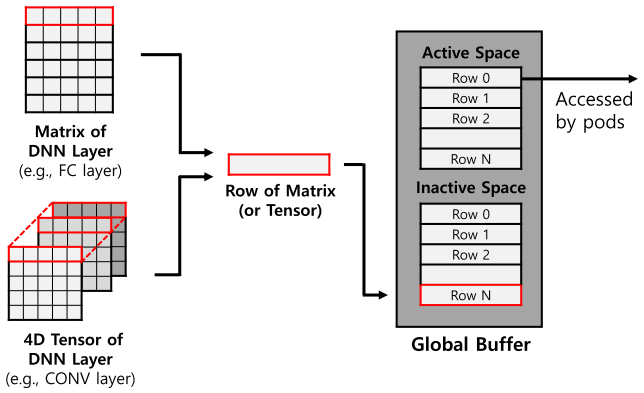


FIGURE 7. Row sharing of SAVector global buffer. The global buffer keeps and manages the rows of input or weight matrix inside the buffer. For convolutional layers, SAVector keeps the rows of input tensor across all channels. SAVector shares those rows with pods to mitigate duplicated off-chip memory access.

Like the prefetching process in the global buffer, the local pod controller employs one-half of a 1KB buffer as an active buffer and the other half as an inactive buffer. The local pod controller conducts prefetching for the input and weight data, directing them into the dedicated inactive buffers. It proceeds to utilize this data once all the prefetched information is accessible. Tiny buffers require more frequent prefetching and flush compared to the hundreds of KB buffers equipped by existing scale-out architectures. SAVector typically prefetches 16 32-byte rows from the global buffer to the tiny buffer every 16-20 cycles. Despite this overhead, SAVector significantly reduces the expensive off-chip memory access by utilizing the two-level buffer system.

2) GLOBAL BUFFER AND GLOBAL BUFFER CONTROLLER

We design the global buffers using double-buffered scratchpad memory, a concept proposed in prior work [27]. To determine the appropriate size for the global buffers dedicated to input and weight data, we analyze sizing trade-offs for the best SAVector performance. Each global buffer comprises 4 SRAM banks, each equipped with 256KB of SRAM and a single 32-byte port. We assume that a single parameter occupies 32 bytes. SAVector does not incorporate a global buffer for output since the dedicated output buffers directly facilitate the transfer of output data to off-chip memory.

Figure 7 shows the structure and mechanism of the global buffer of SAVector. As mentioned in Section II-B, pods frequently access the same rows of input or weight matrices within a short time window. To effectively manage the duplicated access patterns of pods, SAVector retains and shares these rows within the global buffer. Many DNN layers can be simplified as the multiplication of input and weight matrices [3]. In the process described in Section III-A, SAVector fetches and organizes rows from these matrices.

If the global buffer preserves these rows of the lowered input matrix, it redundantly stores duplicated input elements. To circumvent input data duplication in the global buffer, SAVector maintains rows of the original input tensor for convolutional layers. Pods access these rows stored in the global buffer, converting them into a 2D matrix upon retrieval into the small buffers within the pods. Additionally, because the global buffer of SAVector has a granularity of 32B for access, padding can occur if the height or width of the input tensor is less than 32 (e.g., height=7, width=7 input in ResNet-50). To mitigate this concern, SAVector concatenates the rows of the input tensor across all channels, ensuring that the size of each row remains consistently greater than 32. SAVector does not explicitly concatenate rows but instead uses a Height-Width-Channel (HWC) data format, a concept demonstrated in previous work [28], [29]. This format guarantees continuous memory addressing for tensor elements in the same position across all channels, thus ensuring continuous memory addressing for rows of tensors across all channels.

As described in the section III-A, the global buffer controller employs *used_bits* to maintain information about the currently utilized pods. Given that pods in each row share input data, the input global buffer controller should track how many pods are active in a row. The input global buffer controller stores this information in *used_col* bits. Similarly, the weight global buffer necessitates *used_row* bits to indicate the number of active pods in a column. For a system with pods of dimensions N×M, the global buffer controller mandates $logN$ bits for *used_row* and $logM$ bits for *used_col*. The counter also demands an equivalent number of bit fields as *used* bits. The input global buffer controller requires $2logM$ bits, while the weight global buffer controller requires $2logN$ bits.

3) INTERCONNECTION NETWORK

Scale-out DNN accelerators often utilize multistage networks to leverage their low latency and reduced hardware cost [3], [6], [30], [31]. In our design, we employ a butterfly network to connect pods and the global buffer, inspired by prior work [3]. This butterfly network has $O(N \log N)$ hardware cost and $O(\log N)$ latency. In the prior work, the network interconnected N pods with the N buffers (or banks) in a single network, as all pods shared these buffers. In contrast, SAVector connects a global buffer to every $\sqrt{N}$ pods. This design, informed by the memory access patterns analyzed in Section II-B, separately connects these pods with a global buffer bank to reduce the number of nodes in the interconnect network. The number of nodes in the network is contingent on the stages in multistage networks, and SAVector has fewer network stages than prior work. With this scheme, SAVector achieves lower network latency and reduced area overhead compared to prior work when considering a fixed number of pods.

TABLE 3. Microarchitectural configuration of SAVector.

Systolic array size per pod	32×32
Pod dimension	4×4
Global buffer size (for each input and weight)	1MB
Dedicated buffer size (input and weight)	1KB
Dedicated buffer size (output)	64KB
Off-chip memory	HBM2
Off-chip memory bandwidth	614GB/s
Dataflow	Weight Stationary
MAC energy (pJ)	0.48
Global buffer access energy (pJ/byte)	3.69
Dedicated buffer access energy (pJ/byte)	0.15
Off-chip memory access energy (pJ/byte)	31.2

TABLE 4. DNN models for benchmark.

Class	DNN model name
CNN	Mobilenetv3-large, DenseNet-169, ResNet-50
Transformer	BERT-Base, BERT-Large, ViT

4) TILING AND SCHEDULING

If a workload generates an insufficient number of tiles, scale-out architectures may suffer from throughput degradation. Prior work has proposed finer-grained tiling techniques to maximize the number of active pods [3], [6]. These techniques tile the matrix to match the size of the systolic array (e.g., 32 × 32). While the prior work could improve throughput, it may still incur redundant off-chip memory access caused by the Type 1 memory access pattern. To address the above challenge, we design a new scheduling technique that utilizes pods as many as possible and mitigates redundant memory access. The SAVector scheduler assigns tiles to pods based on the following policies. First, SAVector assigns tiles from the same input matrix rows to pods in the same row of the pod dimension. Second, SAVector assigns tiles from the same weight matrix rows to pods in the same column of the pod dimension. With the proposed scheduling technique, SAVector can achieve better speedup than the prior work on the scale-out architectures.

5) HARDWARE OVERHEAD

In the SAVector architecture, each global buffer has a dedicated global buffer controller. With 16 pods (arranged

as 4 × 4), each input global buffer controller requires 4 bits for *used_col* bits and counter bits (= *log*4 + *log*4 bits), respectively. Likewise, each weight global buffer controller also necessitates 4 bits for *used_row* bits along with counter bits. The total SRAM overhead of SAVector is only 3.02MB, which is just 25% of the overhead in prior work [3]. Consequently, the overhead for the controllers is almost negligible. For an accelerator with 1024 pods (32 × 32), the added overhead from these bits accounts for a mere 80 bytes, while SAVector significantly reduces the SRAM overhead by 576MB.

IV. EVALUATION

A. EVALUATION SETUP

1) HARDWARE CONFIGURATION

Table 3 and Table 4 describe the hardware configuration of SAVector and the benchmarks used in our evaluation, respectively. Except for the hardware specification of SAVector, the hardware configuration is the same as described in Section III-B. In our evaluation, given a fixed number of PEs (16,384), SAVector has 4 × 4 pods, and each pod consists of a 32 × 32 systolic array. Also, each pod of SAVector contains 1 KB input and weight dedicated buffers, and a 64 KB output buffer. All the pods share 1MB input and weight global buffers. We use a memory system with two HBM2 chips whose bandwidth is 614 GB/s [32], by referring to the off-chip memory bandwidth of TPUv4i [4].

2) SIMULATION INFRASTRUCTURE

We model SAVector and other architectures with SCALE-Simv2 [15]. We model the memory access latency with Booksim 2.0 [25]. Also, we model the power and energy consumption of all the hardware resources with Accelergy [33]. We estimate the energy consumption for MAC operation, off-chip memory access, and SRAM access at a clock frequency of 1GHz. Table 3 describes estimated energy consumptions.

3) BENCHMARKS

We use widely-used DNN models as benchmarks. First, we select three CNN models: Mobilenetv3-large [2], DenseNet-169 [20], and Resnet-50 [21]. Second, we select three transformer models: BERT-base, BERT-large [22], and Vision Transformer (ViT) [23].

4) BASELINE AND OTHER ARCHITECTURES

We compare SAVector to four architectures. For a fair comparison, we configure each architecture with the same number of total PEs ($= 2^{14}$). First, Scale-Up architecture has a single 128 × 128 systolic array. The scale-up architecture has a 1.5MB input buffer, 1.5MB weight buffer, and 1MB output buffer.

Second, Scale-Out Naïve architecture splits the large 128 × 128 PE array into 16 32 × 32 systolic array pods. The scale-out naïve architecture has no global buffer and each pod has three 256KB dedicated buffers for input, weight, and output, respectively. As shown in section II, this approach can improve utilization and performance, but suffers from duplicated off-chip memory access.

Third, Scale-Out Systolic Arrays (SOSA) architecture includes an optimal systolic array size (32 × 32), interconnection, tiling and scheduling mechanism for scale-out architectures [3]. As SOSA does not contain tiny buffers inside pods to prefetch tiles, pods fetch next input tile from the global buffer immediately after processing the current tile is completed. For a fair comparison, we set the total global buffer size of SOSA as 3MB, which is almost the same as SAVector.

Our investigation into the work published after SOSA [7], [8], [9] focuses on developing efficient scheduling techniques for scale-out architectures. Based on these findings, we model an idealized architecture called Reconfigurable (RC) architecture for comparison. The RC architecture can optimally reconfigure pod dimensions, functioning similarly to scale-up architectures but with the most advantageous dimensions. The RC architecture effectively eliminates redundant off-chip memory accesses by facilitating data propagation between adjacent pods via additional datapaths. Although the RC architecture can operate in a scale-out configuration, it encounters performance degradation due to interconnect latency, a limitation also observed in the SOSA architecture. Like SOSA, the RC architecture features 16 pods, each with dimensions of 32 × 32, and a total SRAM buffer capacity of 3MB.

Finally, Ideal Buf architecture contains an ideal global buffer that has a very large capacity (256 KB/bank × 10K banks) and infinite bandwidth. The ideal global buffer can keep all the operand data to complete the computation, thus minimizing off-chip memory access count. We optimistically assume that the ideal global buffer comprises access latency of Ideal Buf is one cycle and turns off the unused banks. Therefore, Ideal Buf does not experience performance degradation due to the global buffer access latency.

B. EXPERIMENTAL RESULTS

To evaluate the performance and energy efficiency of all the architectures, we measure and compare the runtime and energy consumption. Figure 8 shows the speedup of all the configurations over the scale-up architecture. We calculate the geometric mean of the results for all six benchmarks. SAVector achieves 1.42× speedup over the scale-up architecture. SAVector successfully conceals global buffer access latency by utilizing tiny buffers, achieving the ideal performance. SAVector also achieves the performance improvement over Scale-Out naïve as the tiling and scheduling technique described in Section III-B effectively increases the number of active pods. Due to exposed global buffer access latency, SOSA only exhibits 64% of its ideal performance. SOSA suffers from performance degradation in DenseNet and ViT as it could not provision enough tiles to the accelerators. SOSA could achieve better performance than the baseline in ResNet and BERT-base, however the

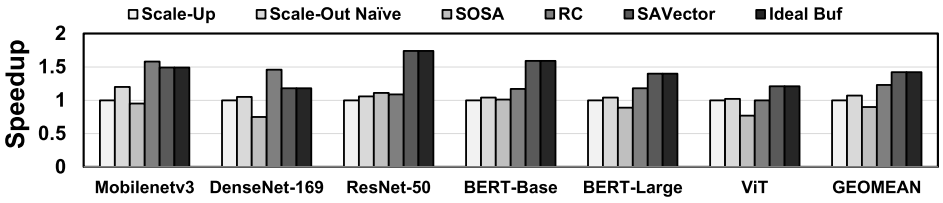


FIGURE 8. Performance comparison between SAVector and other architectures (Normalized to Scale-Up).

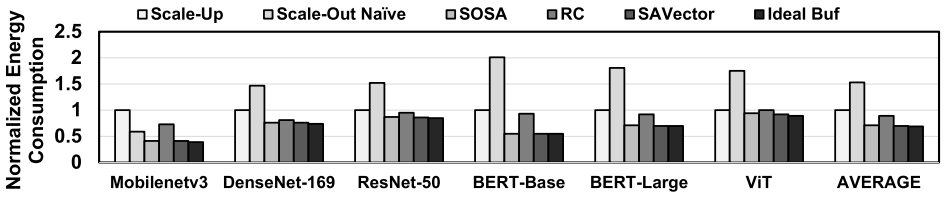


FIGURE 9. Energy consumption comparison between SAVector and other architectures (Normalized to Scale-Up).

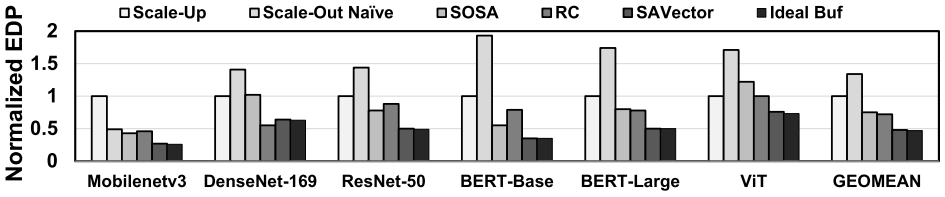


FIGURE 10. EDP comparison between SAVector and other architectures (Normalized to Scale-Up).

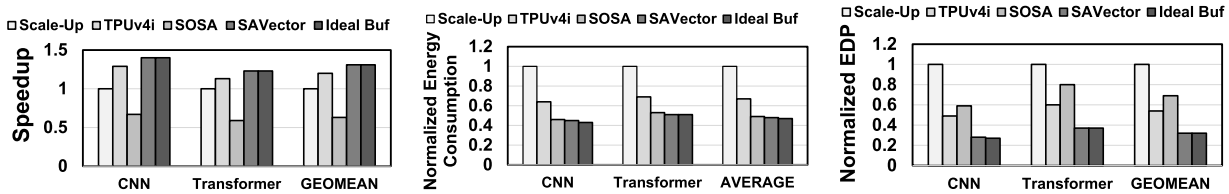
overall performance of SOSA is 10% lower than the scale-up architecture in our evaluation. RC architecture achieves a 1.23× speedup over scale-up architecture by reconfiguring the pod dimension. For two benchmarks (Mobilenetv3 and DenseNet-169), RC architecture shows better performance than SAVector. We observe that these models contain many DNN layers with small tensors (e.g., 7×7 inputs and 3×3 weights), leading SAVector to inactivate most pods to reduce energy consumption. On average (Geometric mean), RC exhibits 19% lower performance than SAVector.

Figure 9 shows the experimental results for energy consumption. We normalize all results to the scale-up architecture. On average, SAVector reduces the energy consumption by 30% compared to the scale-up architecture. As SAVector eliminates Type 1 data duplication, it achieves an energy consumption that is only 2% more than Ideal BUF. Due to the redundant off-chip memory access explained in Section II-B, Scale-Out naïve shows the highest energy consumption. SOSA consumes 29% less energy than the scale-up architecture by improving PE utilization and eliminating the duplicated off-chip memory access. Although SOSA experiences an increase in runtime due to global buffer access latency, it does not consume dynamic energy during stalls. As such, SOSA consumes energy similar to SAVector. RC architecture exhibits higher energy consumption compared to SAVector and SOSA. Unlike other scale-out architectures, RC architecture cannot selectively turn off pods due to data movement between pods, resulting in additional on-chip energy consumption.

Figure 10 shows the EDP results of all configurations. SAVector achieves 52% and 27% EDP reduction compared to the scale-up architecture and SOSA, respectively. As SAVector offers very similar performance and energy consumption to Ideal Buf, it achieves almost the same EDP results as the Ideal Buf. SOSA reduces EDP by 23% compared to the scale-up architecture but exhibits higher EDP than SAVector due to performance degradation. RC architecture also shows a 24% higher EDP than SAVector. Although RC architecture does not suffer from slowdown due to interconnect latency, it consumes extra energy in systolic arrays. Overall, SAVector shows the best EDP among all configurations.

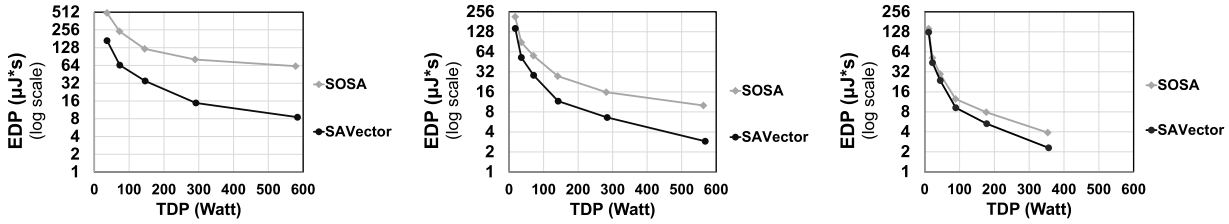
C. EFFECTS OF SCALING OUT

The main purpose of scale-out architecture is to achieve better energy efficiency at scale. We analyze how SAVector scales well while increasing the number of pods. For analysis, we increase the total PE budget from 2^{14} to 2^{16}. Then, we analyze the performance and energy efficiency of SAVector with the 32×32 systolic arrays. The pod dimension of SAVector is 8×8. We compare SAVector to the following three architectures: a scale-up architecture with



(a) Speedup (b) Normalized energy consumption (c) Normalized EDP

FIGURE 11. Comparison between SAVector and other architecture with 2^{16} processing elements (PEs). We normalize all experimental results to Scale-Up. Values of CNN and Transformer are the means (geometric means for speedup and EDP, and arithmetic mean for energy consumption) of the results for each category shown in Table 4.



(a) 16×16 Systolic array (b) 32×32 Systolic array (c) 64×64 Systolic array

FIGURE 12. EDP comparison of SAVector and SOSA for various TDP (log scale). We use three different systolic array sizes (16 × 16, 32 × 32, and 64 × 64) in this study. For each TDP value, we configure the accelerators that equip PEs and other hardware resources as many as possible, but do not exceed the TDP.

a single 256 × 256 systolic array pod, TPUv4i architecture with four 128 × 128 systolic array pods [4], and SOSA with 64 pods.

We observe that the RC architectures may incur significant hardware overhead at scale due to high-cost crossbar interconnect or long hardware links between buffers and PEs. For this reason, the prior work on the reconfigurable architectures did not evaluate their scalability with more than 2^{14} PEs [7], [8], [9]. Therefore, we mainly compare the scalability of SAVector and SOSA, excluding the RC architecture that we model for comparison.

Figure 11 shows the experimental results of speedup, energy consumption, and EDP. We calculate the geometric means (speedup and EDP) and arithmetic mean (energy consumption) of the results for all six benchmarks. Also, we separately calculate the means of the results for CNN and transformer models. With the 2^{16} PE configuration, SAVector achieves a 68% EDP reduction compared to Scale-Up, while it achieves a 52% EDP reduction with the 2^{14} PE configuration. Scale-Up and TPUv4i exhibit 68% and 22% higher EDPs than SAVector, respectively, as they often suffer from PE underutilization. TPUv4i achieves 1.2× speedup compared to Scale-Up, but it consumes 20% more energy than SAVector to manage large-sized systolic arrays. SOSA suffers from performance degradation as its global buffer access latency becomes longer than the SOSA with 2^{14} PEs.

As there are demands of various types of computing systems, DNN accelerators should be designed depending on the requirements of target systems, ranging from mobile to large-scale supercomputers. We evaluate the effects of various systolic array sizes at scaling out an accelerator.

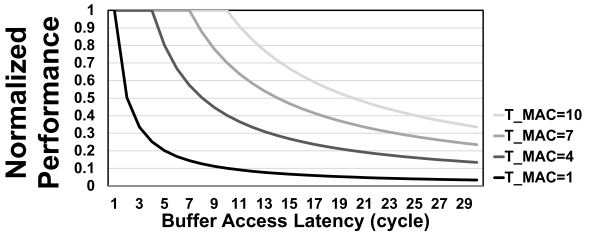


FIGURE 13. Normalized performance (Baseline: buffer access latency = 1 cycle). T_MAC represents the cycles for each MAC operation.

We compare the EDP of SAVector and SOSA by increasing the number of pods with a maximum TDP of 600 W. Like the prior work [3], we design two scale-out accelerators (SOSA and SAVector) that equip PEs and other hardware resources as many as possible, but do not exceed TDP given a configuration.

While we determine the systolic array size to 32 × 32, a target workload running on a system may run the best in another systolic array dimension. To analyze the effect on the systolic array dimension of scale-out accelerators, we evaluate three different systolic array (SA) sizes: 16 × 16, 32 × 32, and 64 × 64. We use a custom benchmark containing a large 4096 × 4096 input and weight matrix to generate a sufficient number of tile operations so that both architectures exploit parallelism across pods at scale. As we design tiny 1KB buffers of SAVector for 16 pods, we increase the size of the tiny buffer for each configuration to hide global buffer access latency. Even with 256 pods, the average global buffer access latency is 34 cycles and the aggregated size of tiny buffers is only 2MB (4% of total SRAM capacity).

Figure 12 shows the experimental results. SAVector outperforms SOSA with all three systolic array sizes. In the 64 × 64 systolic array configuration, SAVector achieves the best (lowest) EDP, which is 2.3 μJ·s. For the target workload with the large matrix (4096 × 4096), SAVector and SOSA with the 64 × 64 systolic array configuration show better EDPs than those with the 32 × 32 systolic arrays as both architectures have enough tiles to provision to pods. SAVector exhibits better performance than SOSA as the number of pods increases. In particular, SAVector with 16 × 16 and 32 × 32 systolic array configurations show significantly better performance than SOSA. It is because SOSA suffers from a slowdown in these configurations. At 600W TDP, SOSA with 16 × 16 systolic array and 32 × 32 systolic arrays exhibit 86% and 71% lower performance than SAVector, respectively. Even if we exclude such extreme cases, SAVector achieves 41% EDP reduction for 64 × 64 systolic array at 600W TDP.

D. EFFECTS OF GLOBAL BUFFER ACCESS AND MAC OPERATION LATENCIES

The performance of SAVector depends on two factors: the global buffer access latency and the latency required for a single MAC operation. We call them T_{buf} and T_{MAC}, respectively. For example, If T_{buf} is longer than T_{MAC}, the exposed stall may degrade the overall performance.

We analyze the performance of SAVector by varying these two latencies in our simulator. Figure 13 shows the experimental results. We vary T_{MAC} from 1 to 10 cycles and T_{buf} from 1 to 30 cycles. For each T_{MAC} configuration, the experimental results show the relative performance compared to the case of $T_{buf} = 1$. For all the cases of $T_{MAC} = 1$, SAVector suffers from slowdown as T_{buf} increases. Buffer access latency is exposed as the computation cannot hide it. With more than nine cycles of T_{buf}, performance decreases by over 90%. If the MAC operation takes multiple cycles (e.g., $T_{MAC} = 7, 10$), the performance is less sensitive to buffer access latency. By overlapping the buffer access and MAC operations, T_{MAC} can hide buffer access latency shorter than T_{MAC}. Even with T_{buf} longer than T_{MAC}, an exposed stall is short because most latency can be hidden in computation.

T_{MAC} depends on the hardware design of an accelerator. Prior work has proposed pipelined PE architecture that increases the parallelism in a single PE [16], [34]. Those architectures enable 1-cycle MAC operation by using more than three pipeline stages for memory access, multiplication, and addition. We also assume that each MAC operation takes one cycle ($T_{MAC} = 1$), and design tiny buffers to hide global buffer access latency. Therefore, SAVector is effective for all hardware configurations in the design space.

The experimental results in this section show that SAVector achieves high performance and energy efficiency over the various scale-up and scale-out architectures. SAVector scales efficiently by eliminating data duplication while enjoying the advantages of scale-out architecture.

V. RELATED WORK

A. DNN ACCELERATORS

With the increase in the number of DNN applications and the number of parameters in DNN models, many domain-specific architectures have been proposed to achieve a higher throughput per watt over CPUs and GPUs [5], [35], [36], [37]. DNN accelerators often use the systolic array [38] as their computation logic thanks to their high energy efficiency [4], [5], [6], [7], [8], [9], [12], [19].

To design systolic array-based DNN accelerators efficiently, prior work has proposed a full-stack DNN accelerator generator called Gemmini [26]. Gemmini allows design space exploration of DNN accelerators and evaluation of system-level trade-offs such as OS overhead.

Also, several studies have proposed non-systolic array-based architectures to resolve the underutilization caused by the rigid structure of the systolic array [16], [30], [31], [34], [39]. Those studies mainly focus on improving PE utilization within a single pod while targeting irregular and sparse matrices in DNN workloads. Prior work has proposed a flexible architecture called SIGMA. SIGMA improves PE utilization regardless of the shape of target matrices [31]. Unlike the systolic array that only allows one-directional distribution and reduction, SIGMA supports flexible mapping through a non-blocking distribution network and an adder tree reduction network. Also, Kwon et al. have proposed MAERI, which is a reconfigurable architecture that supports various dataflow in an accelerator [30]. The cost of non-systolic array-based architectures is additional hardware and complex routing/control algorithm to support their dataflow.

B. SCALE-UP DNN ACCELERATORS

Tensor Processing Units (TPUs) use large systolic arrays. The first generation of TPU has a monolithic 256 × 256 systolic array [5]. Unlike TPUv1, which targets inference tasks only, TPUv2 is for both training and inference. TPUv2 and TPUv3 consist of two and four 128 × 128 systolic arrays, respectively [40]. From the second version, TPU uses HBM as off-chip memory to mitigate the memory bottleneck. TPUv4i targets inference, and they consist of four 128 × 128 systolic arrays per core [4]. Such large systolic arrays can be effective for DNN models that have large tensors. However, as DNN models (or layers) have diverse GEMM sizes, the scale-up architectures often suffer from severe underutilization.

C. SCALE-OUT DNN ACCELERATORS

To address the PE underutilization problem, several studies proposed scale-out architectures. Yüzügüler et al. have proposed three key optimizations for array granularity, interconnect, and tiling strategy [3]. Xu et al. have proposed new processing elements that support multiple dataflows to process depthwise convolution layers efficiently [7]. They also have proposed a flexible buffer structure to reconfigure a pod dimension. Lym et al. and Samajdar et al.

have proposed reconfigurable architecture with additional datapaths between systolic arrays [8], [9]. Based on the target workload, the proposed architectures logically reconfigure their systolic array size or pod dimension. To address the memory bottleneck of DNN accelerators, Kung et al. and Gao et al. have proposed their 3D-IC architectures [6], [41].

Prior work that has proposed scale-out architectures focuses on increasing PE utilization and reducing on-chip data movement traffic. Unlike prior work, SAVector focuses on reducing the number of off-chip memory access. Also, to the best of our knowledge, SAVector is the first scale-out architecture that has a new multi-level on-chip buffer structure and its management mechanism. SAVector significantly reduces the energy consumption in off-chip memory by exploiting the memory access patterns of pods.

Also, prior work has proposed the scheduling and mapping strategy for scale-out architectures [19], [42], [43]. Cai et al. have proposed a systematic notation to enable design space exploration of inter-layer scheduling for scale-out architectures [42]. Based on the notation, Cai et al. have proposed SET, a scheduling framework for scale-out architecture. Zheng et al. have proposed the tile-centric notation and analysis to explore 3D design space including computation ordering, resource binding, and tiling [43]. The proposed scheduling techniques are a general solution for scale-out architecture and are orthogonal to our main contribution through innovation in the on-chip memory system.

VI. CONCLUSION

While scale-out architecture offers better utilization of DNN accelerators, it incurs redundant off-chip memory access. Such duplicated off-chip memory access incurs significant energy consumption. We observe that scale-out DNN inference accelerators commonly exhibit a memory access pattern in that multiple pods access the data in the same row of input and weights within a short time window. Based on this observation, we propose the SAVector architecture. SAVector has a separate global buffer for each pod dimension row and column, and pods with the same memory access pattern share a global buffer. By exploiting the dominant memory access patterns, SAVector eliminates most redundant off-chip memory access. Our experimental results show that SAVector achieves EDP reduction over scale-up architecture and existing scale-out architecture by 52% and 27%, respectively.

REFERENCES

[1] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 30318–30332.

[2] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[3] A. C. Yüzügüler, C. Sönmez, M. Drumond, Y. Oh, B. Falsafi, and P. Frossard, "Scale-out systolic arrays," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, pp. 1–25, Jun. 2023.

[4] N. P. Jouppi, D. Hyun 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: Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1–14.

[5] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[6] H. T. Kung, B. McDanel, S. Q. Zhang, X. Dong, and C. C. Chen, "Maestro: A memory-on-logic architecture for coordinated parallel use of many systolic arrays," in *Proc. IEEE 30th Int. Conf. Application-Specific Syst., Architectures Processors (ASAP)*, vol. 2160, Jul. 2019, pp. 42–50.

[7] R. Xu, S. Ma, Y. Wang, Y. Guo, D. Li, and Y. Qiao, "Heterogeneous systolic array architecture for compact CNNs hardware accelerators," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2860–2871, Nov. 2022.

[8] S. Lym and M. Erez, "FlexSA: Flexible systolic array architecture for efficient pruned DNN model training," 2020, *arXiv:2004.13027*.

[9] A. Samajdar, E. Qin, M. Pellauer, and T. Krishna, "Self adaptive reconfigurable arrays (SARA): Learning flexible GEMM accelerator configuration and mapping-space using ML," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 583–588.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[11] J. Lee, J. Choi, J. Kim, J. Lee, and Y. Kim, "Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array NPUs," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 247–252.

[12] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 681–697.

[13] 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 *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 14–27.

[14] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 807–820.

[15] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Aug. 2020, pp. 58–68.

[16] 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 J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[17] A. Arunkumar, E. Bolotin, D. Nellans, and C.-J. Wu, "Understanding the future of energy efficiency in multi-module GPUs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 519–532.

[18] P. Gu, X. Xie, S. Li, D. Niu, H. Zheng, K. T. Malladi, and Y. Xie, "DLUX: A LUT-based near-bank accelerator for data center deep learning training workloads," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1586–1599, Aug. 2021.

[19] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 940–953.

[20] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016, *arXiv:1608.06993*.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

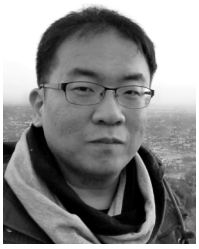
[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16 × 6 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy
[24] Z.-G. Liu, P. N. Whatmough, and M. Mattina, "Sparse systolic tensor array for efficient CNN hardware acceleration," 2020, *arXiv:2009.02381*.
[25] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 86–96.
[26] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 769–774.
[27] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *Proc. 10th Int. Symp. Hardw./Softw. Codesign.*, May 2002, pp. 73–78.
[28] L. Lai and N. Suda, "Enabling deep learning at the LoT edge," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2018, pp. 1–6.
[29] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Nov. 2021, pp. 214–225.
[30] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, Nov. 2018.
[31] 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 *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.
[32] K. Sohn, W.-J. Yun, R. Oh, C.-S. Oh, S.-Y. Seo, M.-S. Park, D.-H. Shin, W.-C. Jung, S.-H. Shin, J.-M. Ryu, H.-S. Yu, J.-H. Jung, H. Lee, S.-Y. Kang, Y.-S. Sohn, J.-H. Choi, Y.-C. Bae, S.-J. Jang, and G. Jin, "A 1.2 V 20 nm 307 GB/s HBM DRAM with at-speed wafer-level IO test scheme and adaptive refresh considering temperature distribution," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 250–260, Jan. 2017.
[33] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
[34] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[35] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225134–225180, 2020.
[36] 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," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 269–284, Apr. 2014.
[37] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.
[38] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Proc. Sparse Matrix*, vol. 1. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1979, pp. 256–282.
[39] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STIFT: A spatio-temporal integrated folding tree for efficient reductions in flexible DNN accelerators," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 4, pp. 1–20, Oct. 2023.
[40] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for Google's training chips: TPUv2 and TPUv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, Mar. 2021.
[41] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 751–764.
[42] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma, "Inter-layer scheduling space definition and exploration for tiled accelerators," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–17.
[43] S. Zheng, S. Chen, S. Gao, L. Jia, G. Sun, R. Wang, and Y. Liang, "TileFlow: A framework for modeling fusion dataflow via tree-based analysis," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2023, pp. 1271–1288.

SANGUN CHOI received the B.S. degree in railroad electrical and electronics engineering from Korea National University of Transportation, in 2021, and the M.S. degree in electrical and computer engineering from Sungkyunkwan University, in 2024. He is currently pursuing the Ph.D. degree in electrical engineering with Korea University. His research interest includes energy-efficient neural network accelerator architectures.

SEONGJUN PARK received the B.S. degree from the School of Electronic and Electrical Engineering, Sungkyunkwan University, in 2023. He is currently pursuing the M.S./Ph.D. degree with the Department of Semiconductor System Engineering, Korea University. His research interests include neural network accelerator architectures and processing-in-memory architectures.

JAEYONG PARK received the B.S. degree from the School of Electronic and Electrical Engineering, Sungkyunkwan University, in 2023. He is currently pursuing the M.S./Ph.D. degree with the School of Electrical Engineering, Korea University. His research interests include memory systems for energy-efficient neural network accelerators.

JONGMIN KIM received the B.S. degree from the School of Electronic and Electrical Engineering, Sungkyunkwan University, in 2023. He is currently pursuing the M.S./Ph.D. degree with the School of Electrical Engineering, Korea University. His research interest includes DNN training accelerator architectures.



GUNJAE KOO (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Seoul National University, in 2001 and 2003, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California, in 2018. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Korea University. Prior to joining Korea University, he was an Assistant Professor with Hongik University. His industry experiences include a Senior Research Engineer with LG Electronics and a Research Intern with Intel. His research interests include computer system architecture and span parallel processor architecture, storage and memory systems, accelerators, and secure processor architecture.

SEOKIN HONG (Member, IEEE) received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2015. From 2015 to 2017, he was a Senior Engineer with Samsung Electronics, where he was involved in a project that developed the 3D-stacked memory. In 2017, he moved to the IBM Thomas J. Watson Research Center, where he worked on secure processor architectures and emerging memory/storage systems. He is currently an Assistant Professor with Sungkyunkwan University, South Korea. His current research interests include the design of low-power, reliable, and high-performance processor architectures and memory systems. He was a recipient of the Best Paper Awards from the International Conference on Computer Design (ICCD), in 2010, and the Design Automation and Test in Europe (DATE), in 2013.

MYUNG KUK YOON (Member, IEEE) received the B.S. degree in computer engineering and computational mathematics from Washington State University (WSU), Pullman, WA, USA, in 2011, and the Ph.D. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2018. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Ewha Womans University. Prior to joining Ewha Womans University, he was a Software Developer with Samsung Inc. His research interests include GPU micro-architecture, machine learning accelerators, and parallel programming.

YUNHO OH (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, South Korea, in 2009, 2011, and 2018, respectively. From 2011 to 2014, he was a Software Engineer in the mobile communications business with Samsung Electronics. From 2019 to 2021, he was a Postdoctoral Researcher with the Parallel Systems Architecture Laboratory (PARSA), EPFL, Switzerland. He is currently an Assistant Professor with the School of Electrical Engineering, Korea University. Prior to joining Korea University, he was an Assistant Professor with Sungkyunkwan University. His research interests include hardware and software architectures for energy-efficient data centers, processor architectures (CPUs, GPUs, and neural network accelerators), in-storage processing, memory systems, and high-performance computing.