IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 December 2023, accepted 4 January 2024, date of publication 12 January 2024, date of current version 24 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3353313

RESEARCH ARTICLE

StarSPA: Stride-Aware Sparsity Compression for Efficient CNN Acceleration

NGOC-SON PHAM[1], (Member, IEEE), SANGWON SHIN[1], (Student Member, IEEE), LEI XU[2], (Member, IEEE), WEIDONG SHI[3], (Senior Member, IEEE), AND TAEWEON SUH[1], (Member, IEEE)

[1]Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea
[2]Department of Computer Science, Kent State University, Kent, OH 44240, USA
[3]Department of Computer Science, University of Houston, Houston, TX 77204, USA

Corresponding author: Taeweon Suh (suhtw@korea.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (Convergence security core talent training business (Korea University)) under Grant 2022-0-01198; in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through MSIT under Grant NRF-2022R1A2C1011469; and in part by Samsung Electronics Company Ltd., under Grant IO210204-08384-01.

ABSTRACT The presence of sparsity in both input features and weights within convolutional neural networks offers a valuable opportunity to significantly reduce the number of computations required during inference. Moreover, the practice of compressing input data serves to diminish storage requirements and lower data transfer costs, ultimately enhancing overall power efficiency. However, the compression of randomly sparse inputs introduces challenges in the input matching process, often resulting in substantial hardware overhead and increased power consumption. These challenges arise due to the irregular nature of sparse inputs and the variability in convolutional strides. In response to these challenges, this research introduces an innovative data compression method, named Stride-Aware Sparsity Compression (StarSPA), designed to effectively locate valid input values and expedite the multiplication process. To fully capitalize on this proposed compression method, a weight-stationary approach is employed for efficient convolution. Comprehensive simulations demonstrate that the proposed accelerator achieves speedup factors of 1.17×, 1.05×, 1.09×, 1.23×, and 1.12× when compared to the recent accelerator named SparTen for AlexNet, VGG16, GoogLeNet, ResNet34, and EfficientNetV2, respectively. Furthermore, FPGA implementation of the core reveals a noteworthy 2.55× reduction in hardware size and a 5× enhancement in energy efficiency when compared to SparTen.

INDEX TERMS AI accelerator, convolutional neural networks (CNNs), data compression, dataflow, network on a chip (NoC).

I. INTRODUCTION

The unprecedented success of deep neural networks (DNNs) has established their irreplaceable role in numerous modern applications, such as autonomous vehicles [1], computer vision [2], [3], [4], [5], [6], natural language processing [7], recommendation systems [8], and more [9], [10]. Among the prevalent algorithms of DNNs, convolutional neural networks (CNNs) have particularly excelled in computer vision. CNNs discern meaningful features from complex input images through the use of multiple filters, trained on extensive sample data [11]. With a sufficient training dataset, CNNs exhibit the capacity to learn high-level features and achieve performance surpassing human capabilities [12]. However, the inference processes of CNNs frequently necessitate extensive computations, often involving hundreds of millions of operations [13], [14]. This is due to the utilization of 2-D convolution operations applied to relatively large input feature maps [15]. Such a computational burden can constrain the utility of CNNs in applications that demand high-speed processing or operate within energy-constrained environments, such as mobile devices. Moreover, there is a foreseen trend towards the development of increasingly complex, deep, and large-scale CNNs to meet the requirements of more intricate processing tasks [16]. For instance, in comparison to AlexNet [2] from 2012, which had 60 million

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

parameters and required 630 million computations, by 2021, NFNet [17] has expanded its parameter size by over sevenfold and increased its computational load by a factor of 520. This highlights the pressing need for enhancements in both the computational speed and energy efficiency of CNN inference accelerators.

Leveraging the inherent sparsity present in CNNs provides an efficient and compelling approach to address these requirements. This advantage is rooted in the fundamental nature of convolution, which involves numerous Multiply-Accumulate (MAC) operations between weights and input features. When either or both of these operands are zero, the MAC operation has no impact on the final result. This understanding suggests the feasibility of straightforward and effective optimization techniques, such as halting or bypassing computations when one of the operands is zero. These optimization strategies not only conserve energy but also expedite the computation process. Additionally, it is important to highlight that sparse weights and features can be compressed, resulting in reduced storage requirements and minimized data transfer overhead [18], [19].

Fortunately, sparsity is a prevalent characteristic within CNN models, and it is notably driven by the extensive use of Rectified Linear Unit (ReLU) activation functions [20], [21], which result in the transformation of negative feature values into zeros. In our empirical observations using popular datasets, the application of ReLU activation functions leads to approximately 50%-70% of the features being converted to zero values [22]. This level of sparsity tends to increase in deeper layers of the network. Furthermore, akin to biological neural systems, a significant proportion of weights within CNNs are zero-valued, typically ranging from 20% to 80% without any discernible impact on model accuracy [19], [23], [24]. In practice, CNNs are intentionally initialized with redundant weights, and pruning is subsequently applied using various algorithms, often followed by retraining. Common pruning algorithms include magnitude-based [23], energy consumption-based [25], or model accuracy-based [26] approaches.

Nonetheless, designing DNN accelerators capable of harnessing the advantages of sparsity poses considerable challenges for the following reasons:

1. **Irregular Data Access Patterns**: The data access pattern of CNNs inherently exhibits regularity and predictability. This characteristic remains consistent even when gating zero-involved computations (*ineffectual*) to enhance power efficiency. However, to truly maximize throughput, the ineffectual computations should be replaced with *effectual* ones where both inputs are non-zero. This typically necessitates additional hardware to actively search the input memories for effectual inputs and buffers to store these inputs for future computations. The complexity is further compounded because the input data is often subjected to compression for various advantages, resulting in unpredictability of exact locations of non-zero values. Dealing with this unpredictability typically demands sophisticated hardware for decoding and buffers for staging valid input values to facilitate seamless access by Processing Elements (PEs). Furthermore, the complexity escalates when considering variations in the stride of convolution layers. Different strides necessitate distinct input data patterns, requiring the input decoder logic to accommodate this stride information during the decoding process, further amplifying circuit complexity. For example, Spartan [27] employs prefix sums and priority encoders to locate *valid input pairs* (where both inputs are non-zero and located at matching positions for a valid convolutional operation), utilizing up to 62.7% of the area and consuming 46% of the total power. SCNN [22] relies on the Cartesian product, assuming that each filter weight multiplies with every feature. However, this assumption holds true only for unit-stride convolutions, limiting the applicability of SCNN to specific CNNs. GoSPA [15] mandates a complex circuit to decode position IDs and convolutional IDs for every input feature, in addition to a substantial input buffer to store the inputs.
2. **Workload Imbalance and Low PE Utilization**: In order to attain a high throughput while addressing the inherent data irregularity, it becomes essential to allocate input buffers for the PEs. Leveraging the nature of convolution, data is typically broadcast from memory to the PEs to minimize the number of memory accesses. However, the randomness of input data sparsity results in variations in the number of effectual computations among the PEs. This variance in computation leads to distinct processing times across the PEs. Consequently, the overall throughput of the CNN accelerator is constrained by the PE with the highest number of non-zero MAC operations, ultimately resulting in reduced PE utilization [18], [27].

Structured sparsity, on the other hand, intentionally introduces regularity to sparse inputs, aiming to partially mitigate the irregular access pattern and workload imbalances among the PEs [28]. However, this approach may come at the cost of a more significant compromise in accuracy compared to unstructured sparsity [29], [30], [31]. For instance, when applied to ResNet50 [32], unstructured pruning achieves a compression ratio of 5.96× while maintaining the same accuracy as the original network. In contrast, structured pruning only attains a compression ratio of 1× [33].

This paper introduces an innovative data compression format aimed at simplifying input data matching. The final output features of a layer are compressed based on parameters from the subsequent layer, such as stride information. As a result, the subsequent layer can read input data continuously without requiring complex input matching circuitry. FIGURE 1 illustrates the contrast between conventional architectures and the proposed architecture. In the proposed architecture, the output features are compressed in a ready-for-use format for the subsequent layer. Primarily, this approach alleviates the critical data path associated with input processing by conducting it during a less critical

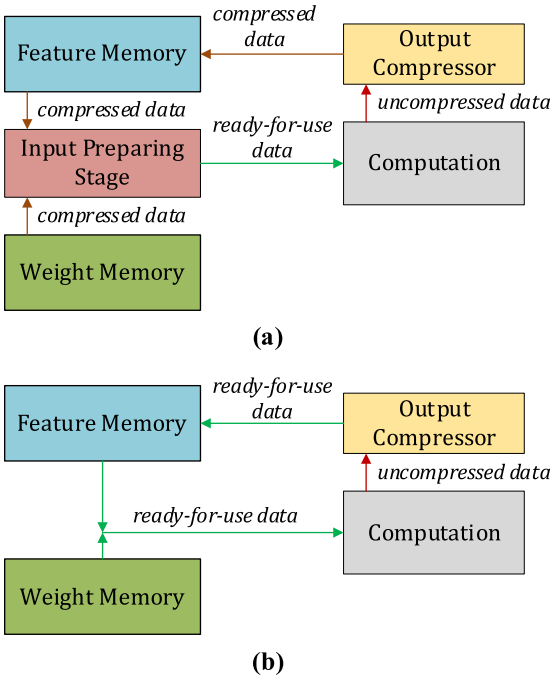


FIGURE 1. Data movement in (a) conventional architectures, and (b) proposed architecture.

duration without compromising the efficacy of data compression. The proposed data compression technique seamlessly integrates into a weight-stationary dataflow with ping-pong double buffering, facilitating contiguous PE operations. This approach reduces PE underutilization and mitigates workload imbalances. The key contributions of this work include:

- We introduce an efficient data compression format named *Stride-Aware Compressed Sparse Row (SCSR)*, which significantly reduces the complexity of input matching logic while ensuring high-performance operation.
- Additionally, we integrate the proposed data compression method into a weight-stationary dataflow with ping-pong double buffering to enable uninterrupted data flow through PEs. This maximizes input data reuse, reduces workload imbalances among PEs, and ultimately increases throughput.

Comprehensive experiments using cycle-accurate simulators and actual hardware implementations validate the efficacy of our solutions, demonstrating substantial improvements over state-of-the-art approaches in terms of speedup, hardware size, and energy efficiency. The subsequent sections of this paper are organized as follows: Section II discusses recent CNN accelerators, Section III introduces the proposed data compression method, Section IV explains the dataflow using this compression method in detail, Section V presents the accelerator's architecture, Section VI provides implementation details and simulation results, and finally, Section VII concludes the paper.

II. RELATED WORKS

The advantages of sparsity exploitation extend beyond the mere reduction of MAC operations, resulting in heightened energy efficiency and enhanced system throughput. By harnessing the inherent compressibility of intermediate output features and weights, a notable reduction in the generated data size can be achieved. This reduction not only minimizes the number of memory access but also enables the sufficient storage of data within low-energy internal SRAM. Given the substantial costs associated with external DRAM access, which normally 20 times more expensive than a SRAM access [34], this data compression represents a significant advantage for sparsity-aware accelerators. Consequently, most sparsity-aware CNN accelerators incorporate sparse compression to mitigate data size. These compression methods can be broadly categorized into three main types: symbol-based methods, absolute indexing methods, and relative indexing methods, as illustrated in FIGURE 2.

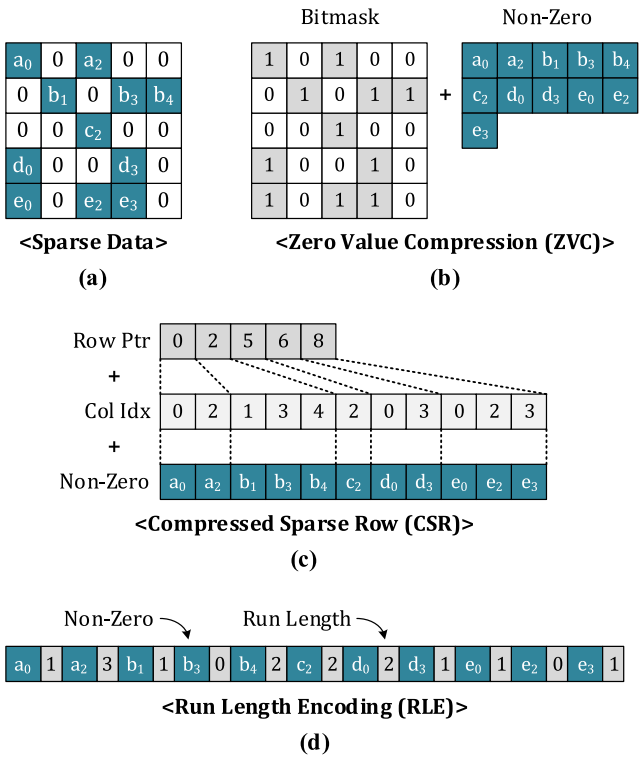


FIGURE 2. Sparse data compression methods. (a) Original sparse matrix. (b) Symbol-based method. (c) Absolute indexing method. (d) Relative indexing method.

TABLE 1 provides an overview of recent noteworthy sparsity-aware architectures, encompassing both *one-sided* and *two-sided* approaches, along with the data compression formats they employ. One-sided sparsity exploitation, which targets the mitigation of sparsity in either filter weights or input features, is generally characterized by its relatively straightforward implementation and less intricate hardware design compared to two-sided sparsity techniques. For example, architectures like Cnvlutin [16], Eyeriss [35],

and NullHop [36] employ different data compression formats to simplify the omission of zero values in features. However, they only gate computations involving zeros in weights rather than completely excluding them, and they do not compress the weights. In contrast, Cambricon-X [37] is specifically designed to exclude zero values in weights while preserving the sparse representation of features. Another illustration, Tensaurus [38], focuses on the flexible acceleration of sparse tensors, dense tensors, and mixed tensors by introducing a novel sparse storage format known as *Compressed Interleaved Sparse Slice (CISS)*. This format not only reduces data transport costs but also enhances the computational efficiency. It is worth noting that Tensaurus is exclusively applicable to weights and does not address sparsity in features.

TABLE 1. Recent sparsity-aware CNN accelerators.

Architecture	Sparsity exploitation type	Data compression format
Cnvlutin [16]	F	CSR
Eyeriss [35]	F	RLE
NullHop [36]	F	ZVC
Cambricon-X [37]	W	RLE
Tensaurus [38]	W	CISS
SIGMA [39]	F+W	ZVC
ExTensor [40]	F+W	CSR
SpArch [41]	F+W	CSR
OuterSPACE [42]	F+W	CR
MatRaptor [43]	F+W	C^2SR
Eyeriss v2 [18]	F+W	CSC
SCNN [22]	F+W	RLE
SparTen [27]	F+W	ZVC
GoSPA [15]	F+W	CSR, ZVC
S^2 Engine [44]	F+W	CSC
Sparse-PE [13]	F+W	ZVC
CSSpa [45]	F+W	ZVC

F = Feature; W = Weight

Recent efforts to improve the acceleration of Generalized Sparse-Matrix-Matrix Multiplication (SpGEMM) through efficient data organization have gained significant attention. For example, SIGMA [39] has adopted the ZVC format to simplify the process of locating valid input pairs. It makes use of a dynamic stream and a search window to identify data that can be paired, relying on the coordinate information from the static stream. Notably, this architecture employs a search window with just four elements, a choice that may prove inadequate in generating valid input pairs, especially in situations with high levels of sparsity. In contrast, ExTensor [40] considers both dynamic streams from the two inputs and uses an intersection operation to identify valid pairs. To streamline the process of skipping unnecessary values, both input streams are compressed in the CSR format. When a "*skipto()*" function is triggered on one of the inputs, the expectation is that the corresponding destination point in the other input contains a non-zero value, ensuring the creation of a valid input pair. Otherwise, it leads to a period of inactivity for PE.

On the other hand, SpArch [41], OuterSPACE [42], and MatRaptor [43] have been developed to implement an *outer-product* (or *input-stationary*) dataflow. This choice serves to mitigate the inefficiencies associated with the *inner-product* dataflow's input matching process. SpArch employs the conventional CSR format, while OuterSPACE and MatRaptor introduce modified versions of the CSR format, denoted as *Compressed Row (CR)* and *Channel Cyclic Sparse Row* (C^2SR), respectively, with the aim of improving data reuse and enhancing memory read-write efficiency. However, it is important to note that these architectural solutions do not address the handling of convolutional stride variations. This oversight can lead to inefficient computations when not appropriately managed or potentially result in additional hardware and power overhead.

Eyeriss v2 [18] addresses the challenge of two-sided sparsity by employing a strategy that involves streaming CSC-based features alongside CSC-based weights. In this approach, for each non-zero feature, an *exhaustive search* is conducted to identify all valid weights, and these valid pairs are subsequently buffered for processing by the MAC units. However, it is worth noting that due to the diversity of strides in various convolutional layers and the relatively large size of the input feature maps, the circuit responsible for searching for valid weights incurs a significant power and hardware overhead. Consequently, when compared to the original Eyeriss architecture [35], Eyeriss v2 occupies approximately 93% more hardware space.

Sparse CNN (SCNN) [22] is specifically designed to address two-sided sparsity in CNNs by implementing a *Cartesian product* dataflow on a data format based on RLE. Similar to the input-stationary dataflow, this unique dataflow architecture eliminates the need for a costly input-pair searching circuit. Nevertheless, SCNN encounters several significant challenges. Firstly, the Cartesian product dataflow often experiences data congestion in its output-scatter network. This arises because products of the same output can be computed concurrently at multiple locations, requiring additional buffers and intricate data routing. Secondly, the Cartesian product approach does not account for convolutional stride, as it involves multiplying every input feature with all non-zero weights. This approach is not suitable for strides other than one or when the input features are located at the boundary of the input feature map.

In contrast to SCNN, SparTen [27] employs an inner-product approach, which involves intersection operations to identify non-zero matching pairs of inputs before executing multiplications. Like other methods that rely on input matching, SparTen utilizes the ZVC format for both input features and weights, simplifying the process of searching for valid input pairs. To ensure an ample supply of valid inputs, SparTen utilizes a relatively *large search window* with a size of 128 elements. However, this strategy results in significant hardware and power costs. Specifically, the analysis of area and power consumption has revealed that the prefix sum components account for 54.6% and 40.6% of the total,

respectively, making substantial contributions to SparTen's overall hardware and power overhead.

GoSPA [15] treats input features as dynamic data streams and weights as stable ones, employing a *global search* approach to compute all possible operations for every non-zero feature. Subsequently, only the required PEs receive the features, as opposed to broadcasting them to all PEs, a common practice in many other architectures. This approach minimizes unnecessary data movement and allows the architecture to adapt to variations in convolutional strides through its global search algorithm. However, it is important to acknowledge that this circuit adds significant hardware and power costs. Furthermore, GoSPA's performance is limited by its input buffering stage, where only necessary input features are buffered one at a time, every clock cycle. The buffers remain idle when unnecessary features arrive, potentially causing the PEs to lack sufficient input data for processing, resulting in PE underutilization.

Sparse-PE [13] utilizes *look-ahead windows* to identify valid pairs within inputs that have been compressed using the ZVC format. These valid pairs are subsequently forwarded to multi-threaded, versatile PEs to execute sparse matrix multiplication. Nonetheless, akin to other methods grounded in input matching, Sparse-PE's look-ahead windows exhibit limitations in generating adequate data in scenarios characterized by high sparsity levels, resulting in suboptimal PE utilization. Conversely, the expansion of look-ahead windows can engender a disproportionate increase in input matching circuit size and complexity. To circumvent the necessity of enlarging the look-ahead windows, S^2 Engine [44] adopts a *fast-clock speed* to seek out valid input pairs, thereby substantially reducing the dimensions of the input matching circuit. Reportedly, only 16 bytes of buffer space are necessary for accommodating valid input features and weights. However, this design choice imposes a constraint on the overall operational speed of the accelerator, causing the maximum attainable operational speed to consistently remain four times slower than the available clock speed.

CSSpa [45] improves the performance of SparTen by reducing the dimensions of the *input search window* while maintaining high processing unit utilization. Additionally, it utilizes a *channel-stacking dataflow*, which maximizes the reuse of internally buffered data, effectively reducing the overall number of memory accesses. However, it is essential to highlight that the input matching circuit still introduces a noticeable overhead. Furthermore, CSSpa employs relatively large input buffers to ensure a balanced load distribution among its PEs.

III. PROPOSED STRIDE-AWARE SPARSITY COMPRESSION

To facilitate comprehension, we examine a one-dimensional convolution operation involving a row of input features and a row of filter weights. FIGURE 3 illustrates an example of one-dimensional convolution in three scenarios: (a) with a stride of 1, (b) with a stride of 2, and (c) with a stride of 2 while utilizing the proposed compression format for the inputs. In FIGURE 3(a), the input data is compressed using either COO or ZVC methods. It is observed that the weight-stationary dataflow and input-stationary

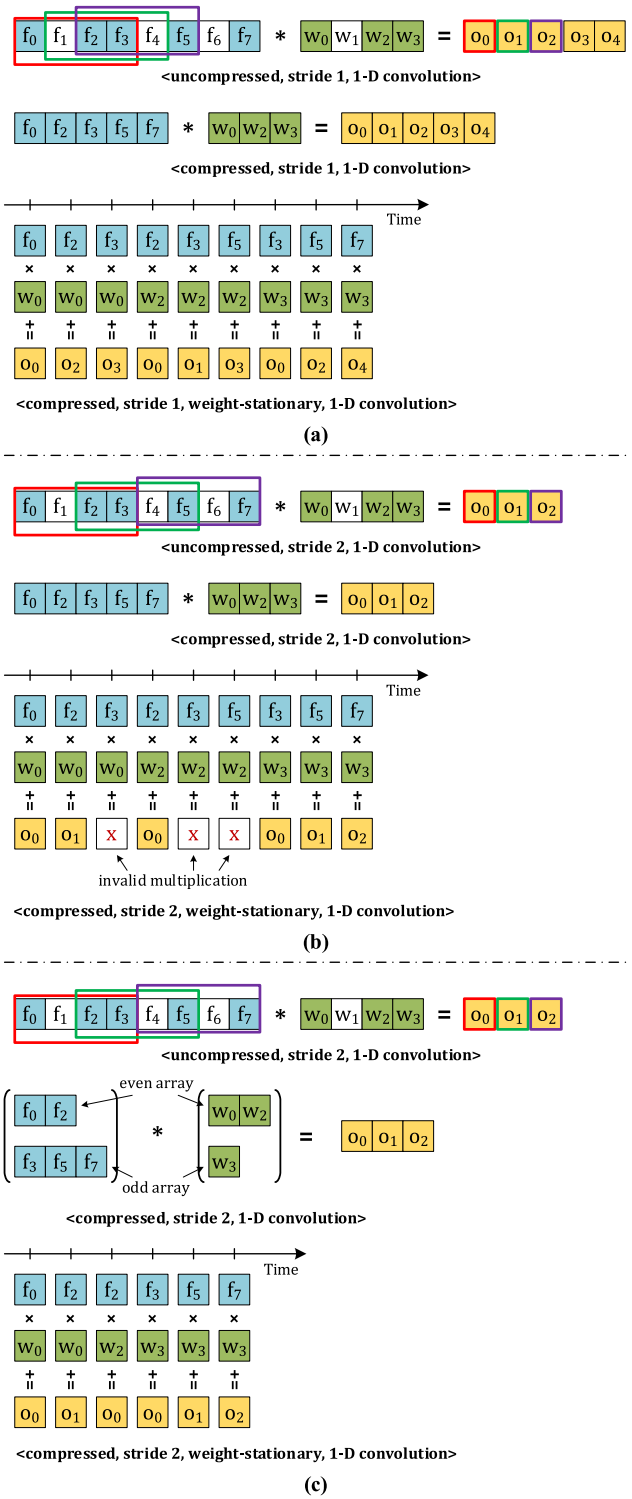


FIGURE 3. Weight-stationary one-dimensional convolutions occur between a row of input features and a row of filter weights in three scenarios: (a) with a stride of 1, (b) with a stride of 2, and (c) with a stride of 2 while employing the proposed input compression technique.

dataflow are more efficient than the output-stationary dataflow when considering the complexity of input matching. FIGURE 3(a) specifically showcases the weight-stationary dataflow. By avoiding invalid multiplications involving input features at the boundary (e.g., $f_0 \times w_2, f_0 \times w_3, f_7 \times w_0$, etc.), the multiplications are executed continuously by a MAC unit. This combination of the input compression format and the dataflow benefits the PE utilization and consequently enhances inference speed.

However, the integration of the conventional data compression format with weight-stationary or input-stationary dataflows is primarily optimized for unit-stride configuration. In cases involving different strides, the MAC operation often encounters interruptions due to the generation of invalid input pairs, leading to unnecessary output products. As depicted in FIGURE 3(b), when the stride is 2, while keeping other setups consistent with the previous example, a multitude of invalid multiplications occur, including instances such as $f_3 \times w_0, f_3 \times w_2$, and $f_5 \times w_2$. To circumvent these unnecessary computations, a typical approach involves the utilization of an input preparation circuit alongside the necessity for additional buffers to store valid input pairs. This practice is prevalent in most accelerators available today. However, when considering the unpredictability of non-zero input values and the variability in sparsity rates, the implementation of input matching circuitry can prove to be ineffective in several aspects. It may lead to a bulky and power-intensive solution in the case of Spartan architecture [27] or result in input shortages in the case of GoSPA [15].

The aforementioned drawbacks of the input preparation circuit can be effectively mitigated by adopting a two-array compression approach for the input features and weights, as proposed and illustrated in FIGURE 3(c). When convoluting with a stride of 2, both the input features and filter weights are organized into an *even array* and an *odd array*, as demonstrated in the figure. In this scenario, the even array of the features is convolved with the even array of the weights, while the odd array of the features is convolved with the odd array of the weights. Consequently, convolution can be efficiently executed without the necessity for input matching logic. For strides other than 2, the number of compressed arrays for both inputs should match the specified stride value.

The key distinction between the proposed compression technique and previous methods lies in the preparation of input features for a given layer (layer n^{th}) while they are still considered as output features from the preceding layer (layer $(n-1)^{th}$). FIGURE 1 illustrates this contrast between conventional architectures and the proposed architecture. In all these architectures, the inference process unfolds sequentially from one layer to the next. In the case of the proposed architecture, during the processing of the $(n-1)^{th}$ layer, the compression unit considers parameters such as the stride of the n^{th} layer to determine the optimal division and compression for the output features. These compressed output features are then stored in a global feature memory, ready to serve as inputs for the n^{th} layer. Notably, since compression units are commonly integrated into most accelerators, the proposed technique does not necessitate additional hardware, such as an input preparation stage. Additionally, the filter weights can be compressed offline before the inference process takes place. Given that the proposed data compression method considers the stride information of the subsequent layer to optimize data storage and facilitate the processing of that subsequent layer, it is aptly named the "*Stride-Aware Compressed Sparse Row" (SCSR*). The proposed accelerator that uses the *SCSR* data format is named *"StarSPA"*, an abbreviation for Stride-Aware Sparsity CNN Accelerator.

The proposed compression method effectively mitigates challenges associated with irregular sparsity and varying convolutional strides. However, it is important to note that the issue of potential invalid multiplications involving boundary input features still persists, and this matter will be explored further in the subsequent section.

IV. OPERATION ON SCSR FORMAT

In the previous section, a 1-D convolution example was employed for ease of introducing the proposed technique. This section provides a more comprehensive elucidation of the SCSR representation and its application in the context of 2-D convolution using the weight-stationary dataflow.

A. SCSR FORMAT

FIGURE 4(a) illustrates a specific instance of 2-D convolution with a stride of 2, involving a 16×16 sparse feature matrix and a 4×4 sparse weight matrix. To simplify the presentation, only the initial four rows of the feature matrix are displayed. Given the stride of 2, the weight is selectively multiplied with features with a distance of 2, rather than with all the features. The illustration identifies the features and weights potentially engaging in mutual multiplications by color-coding. It is noteworthy that a weight is not multiplied with all features sharing the same color, as certain features at the matrix boundary do not facilitate valid computations with the weight. For example, in FIGURE 4(a), the third row of the weight matrix is not multiplied with the first row of the feature matrix. The intricacies of computation addressing with features at the boundary will be further explored shortly.

Given that features and weights with the same color are more likely to undergo mutual multiplication, while those of different colors are not paired, there is a distinct advantage in consecutively compressing features or weights of the same color. This sequential compression facilitates the multiplication process in the PE, as the MAC unit can read consecutive data without relying on an input matching circuit. This forms the crux of the proposed *SCSR* format. The compression of the feature matrix can be formally expressed through a mapping function $\mathbb{Z}^{r_f^o \times c_f^o} \rightarrow \mathbb{Z}^{t_f \times r_f \times c_f}$ with notations and descriptions outlined in TABLE 2. The outputs of the function are determined as follows:

$$t_f = c_f^o \; mod \; s \tag{1}$$

$$r_f = r_f^o \tag{2}$$

$$c_f = c_f^o \tag{3}$$

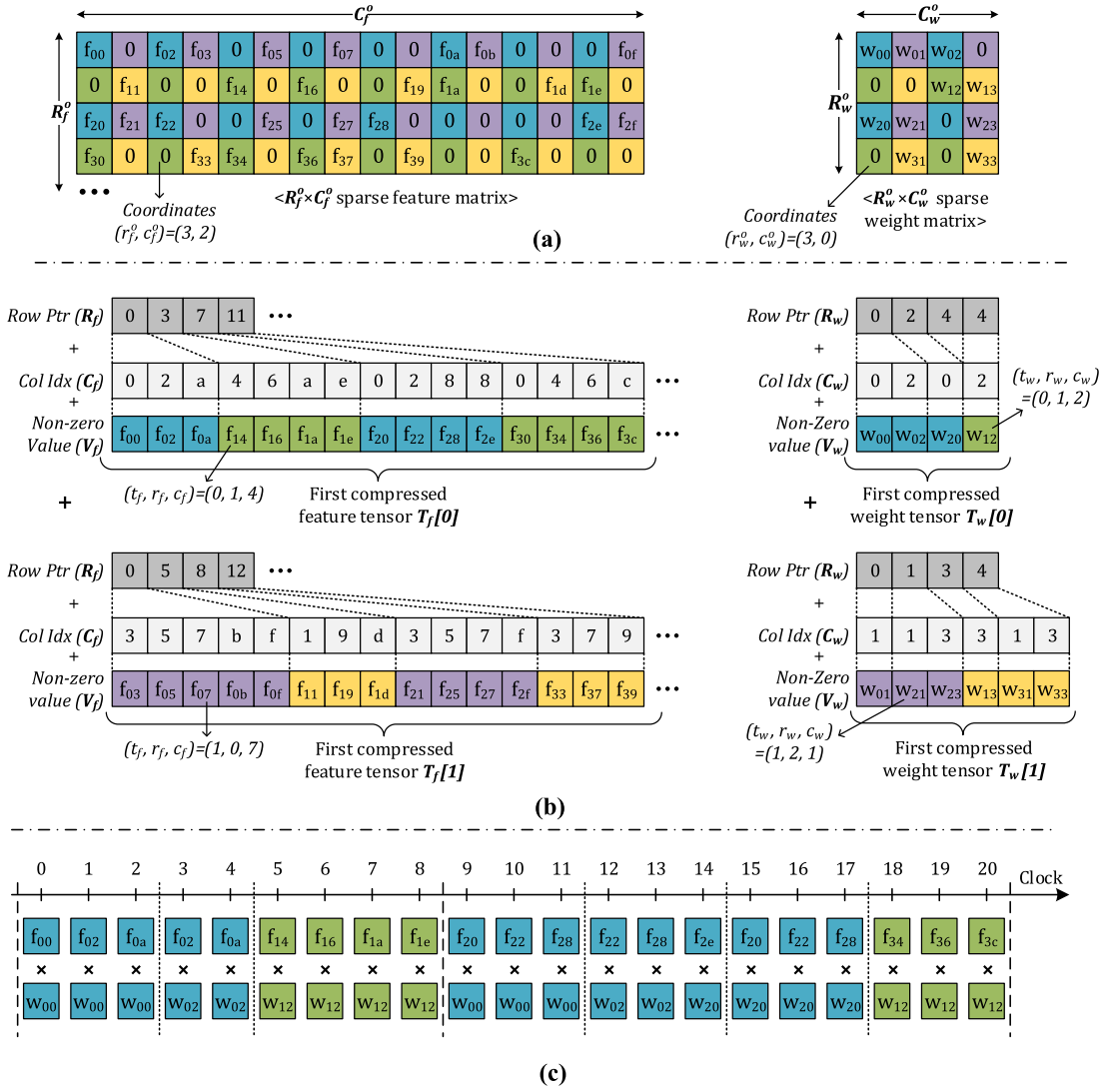


FIGURE 4. Illustration of stride-aware data compression together with its utilization by weight-stationary dataflow: (a) uncompressed sparse input features and weights, (b) stride-aware data compression format when stride is two, (c) the convolution of the first compressed feature and weight tensors on weight-stationary dataflow.

The outcomes reveal that the count of compressed tensors is denoted by $Max\left(t_f\right) = s$, corresponding to the stride. In each tensor, every row exclusively contains features of the same colors. Importantly, the row and column indices of the feature elements remain unchanged from their original matrix configuration. This facilitates a straightforward reconstruction of the original matrix from the CSR format.

The compression of the weight matrix can be mathematically represented by a mapping function $\mathbb{Z}^{r_w^o \times c_w^o} \rightarrow \mathbb{Z}^{t_w \times r_w \times c_w}$. The outputs are determined as follows:

$$t_w = c_w^o \; mod \; s \tag{4}$$

$$r_w = \left(r_w^o \; mod \; s\right) \times s + floor(r_w^o / s) \tag{5}$$

$$c_w = c_w^o \tag{6}$$

It is evident that the number of compressed tensors is $Max\left(t_w\right) = s$, corresponding to the count of compressed feature tensors. In contrast to the compression of features, the order of rows is modified to ensure that all weights with the same color within the original matrix are compressed in close

TABLE 2. Nomenclature.

Notations	Descriptions
s	stride
R_f^o, C_f^o	height and width of original sparse feature matrix
r_f^o, c_f^o	row and column indices of original sparse features
R_w^o, C_w^o	height and width of original sparse weight matrix
r_w^o, c_w^o	row and column indices of original sparse weights
T_f, R_f, C_f	tensors, rows and columns of compressed features
T_w, R_w, C_w	tensors, rows and columns of compressed weights
t_f, r_f, c_f	tensor, row and column indices of compressed features
t_w, r_w, c_w	tensor, row and column indices of compressed weights

proximity to each other. The results of applying the equations to the original sparse matrices are presented in FIGURE 4(b). In this specific example, with the stride of 2, both feature and weight matrices are compressed into two distinct tensors. Given that the order of rows in the weight tensors is altered, the weight compression is more intricate than the feature compression, which performs compression row-by-row. However, it is noteworthy that the weight compression can be conducted offline, eliminating the need for any hardware overhead during inference. Meanwhile, the compression of features is performed online and can be achieved using any straightforward CSR compression circuit without imposing significant hardware demands.

B. CONVOLUTION LAYER

The 2-D convolution between the original feature matrix and weight matrix has been restructured into multiple 2-D convolutions, each involving a feature tensor and its corresponding weight tensor. In this configuration, these convolution tasks are assigned exclusively to a single PE, accommodating only one MAC unit. The dedicated PE designed for these convolution tasks is illustrated in FIGURE 7. One primary objective is to fully leverage the capabilities of the MAC unit, ensuring it performs one multiplication and accumulation operation per clock cycle with minimal idle time. Simultaneously, another goal is to optimize the utilization of the relatively small buffers within the PE, mitigating bulky hardware overhead.

To achieve this, the compressed feature tensors are partitioned into distinct rows before being broadcast to the PE. In contrast, the compressed weight tensors remain intact and are stored within the weight buffer of the PE. Consequently, the initial 2-D convolutions between the feature tensors and weight tensors are transformed into multiple 2-D convolutions between the feature rows and the weight tensors. Specifically, each row of features within a feature tensor is sequentially loaded for convolution with the corresponding weight tensor. For example, every feature row in the feature tensor T_f [0] is processed consecutively in conjunction with the weight tensor T_w [0]. This process is then iteratively repeated for subsequent feature tensors and their corresponding weight tensors.

The timing process, representing a weight-stationary dataflow on the *SCSR*-based input data, is depicted in FIGURE 4(c). This process involves a 2-D convolution between the feature tensor T_f [0] and the weight tensor T_w [0]. The feature tensor is initially subdivided into multiple rows, and each row sequentially participates in a 2-D convolution with the weight tensor T_w [0]. In the context of the stride-2 convolution with the weight tensor, the feature rows only engage with specific weight rows of the same color. During the computation, a weight is selected and remains constant while the input feature counter progresses from a starting point to an end point within the feature row, advancing one step per clock cycle, as illustrated in FIGURE 4(c). The weight undergoes alteration only when the next feature address fails to form a *valid input pair* with the current weight. This transition occurs at specific clock cycles, such as at clocks 2, 4, 8, 11, and so forth.

As demonstrated in the provided example, the integration of the proposed data compression and the weight-stationary dataflow presents an effective solution for mitigating the input irregularity issue commonly encountered in sparsity-aware CNN accelerators. Nevertheless, a challenge persists with the presence of invalid convolutional pairs at the boundaries of the input feature matrix. This challenge manifests at both *row level* and *column level* of the feature tensors. For instance, in the case of row-level invalid pairs, the first row of the feature tensor R_f [0] exclusively participates in convolution with the first row of the weight tensor R_w [0] and does not form valid pairs with the second row R_w [1]. Similarly, for column-level invalid pairs, examples include pairs like (f_{20}, w_{02}) and (f_{2f}, w_{01}), which do not constitute valid operand pairs for convolutional operations.

To address the issue of row-level invalid computations, when a feature row is broadcast to PEs containing weight tensors, two additional signals are concurrently broadcast. These signals serve to inform the PEs about the starting and ending rows of weights that constitute valid computations. The PEs utilize this information to selectively choose weight rows within this valid range. The valid range of rows is denoted as $\left[r_w^{start}, r_w^{end}\right]$, and its calculation is determined by Algorithm 1.

The computation of the $\left[r_w^{start}, r_w^{end}\right]$ pair does not impose a significant time overhead; however, it necessitates specific hardware resources and consumes power. During the inference phase, this pair is generated by a control block and broadcasted to the PEs along with the row of compressed features. Instead of being recalculated locally by each PE, the precomputed pair is shared among multiple PEs. This spatial reuse of the pair across multiple PEs helps to amortize the cost associated with its computation.

Algorithm 1 Select Valid Weight Row

input: r_f – compressed feature row index
s – stride
R_w^o – weight matrix height
R_f^o – feature matrix height
output: r_w^{start}, r_w^{end}

if $\left(R_f^o - r_f < R_w^o\right)$
  $r_w^{start} = \left(r_f mods\right) \times s + floor\left(\left(R_w^o - R_f^o + r_f\right)/s\right)$
else
  $r_w^{start} = \left(r_f mods\right) \times s$
end if
if $\left(r_f < R_w^o - 1\right)$
  $r_w^{end} = \left(r_f mods\right) \times s + floor\left(r_f/s\right)$
else
  $r_w^{end} = \left(\left(r_f mods\right) + 1\right) \times s - 1$
end if

To effectively tackle the challenge of column-level invalid computations, a cost-effective solution involves integrating a specialized address verification block seamlessly into the feature buffer. This block is designed to compute a $\left[c_f^{start}, c_f^{end}\right]$ range of column indices within the feature row for each specific weight position. Valid pairs can only be formed between the weight and features with column indices lying within this range. The formulas to calculate these values are outlined below:

$$c_f^{start} = c_w \tag{7}$$

$$c_f^{end} = C_f^o - C_w^o + c_w \tag{8}$$

To facilitate the selection of valid features within the column index range $\left[c_f^{start}, c_f^{end}\right]$ a counter cnt_f is initialized and incremented by one unit every clock cycle. The condition $c_f^{start} \leq C_f\left[cnt_f\right] \leq c_f^{end}$ is checked to ensure the selection of valid features. Determining the last value of cnt_f is straightforward by comparing the column index of the next feature with c_f^{end}. However, selecting the initial value of cnt_f requires more effort due to the irregular nature of the compressed feature row. To simplify this process, when broadcasting the compressed feature row, a small array $Sel\left[0 : C_w^o - 1\right]$ is concurrently broadcasted. This array aids the PE in promptly identifying the initial value of cnt_f. The search process can be illustrated by the following equation:

$$cnt_f = Sel\left[c_w\right] \tag{9}$$

The computation of Sel is executed in the control block and is illustrated by Algorithm 2.

FIGURE 5 provides a detailed illustration of the 2-D convolution between a compressed feature row and a compressed weight tensor. The compressed feature row corresponds to the third row of the first feature tensor, and the compressed weight tensor is the first weight tensor shown in FIGURE 4(b). The tuple $\left[c_f^{start}, c_f^{end}\right] = [0, 1]$ indicates the column indices of the weight rows participating in the convolution. When a weight is selected, an initial value of cnt_f is calculated by Eq. (9) is calculated by Eq. (9) to select the first valid feature. For example, when the weight w_{02} is selected, $cnt_f = Sel\left[2\right] = 1$, pointing to f_{22}. The counter cnt_f continues to increase until the last valid feature is selected, at which point the selected weight is swapped with the subsequent value.

C. FULLY CONNECTED LAYER

In contrast to convolution layers, fully connected layers do not exhibit weight reuse; each weight is multiplied by only one feature. Consequently, weight-stationary dataflow is not employed for these layers; instead, it is replaced by an output-stationary dataflow. In this configuration, each PE utilizes a dot-product operation on a weight tensor and the input feature tensor to produce a single output. To streamline the dot-product operation on the PE, one-sided sparsity exploitation is applied to reduce the complexity of the input matching circuit. Specifically, only compressed weights are divided into chunks and loaded onto the PEs, with each PE containing a chunk of weights from a distinct filter. Prior to broadcasting chunks of compressed features to these PEs, the compressed features undergo a zero-insertion process to uncompress the chunks. The zero-insertion circuit, illustrated in FIGURE 6(a), employs a single 1:16 multiplexer over multiple clocks to produce an uncompressed chunk of features. This decoded chunk is then broadcasted to the PE, while a new chunk is being processed. This approach ensures that input dependency latency is not incurred. The dot-product engine, depicted in FIGURE 6(b), utilizes a 16:1 multiplexer to select corresponding features based on the index information of the compressed weights, enabling efficient computation of the dot-product in the fully connected layers.

Algorithm 2 Select Valid Feature Column

input: C_f – feature column indices
C_w^o – weight matrix length
output: $Sel\left[0 : C_w^o - 1\right]$

```
int cnt = 0
for (i = 0; i < C_w^o − 1; i++)
    Sel[i] = cnt
    if (C_f[cnt] == i)
        cnt += 1
    end if
end for
```

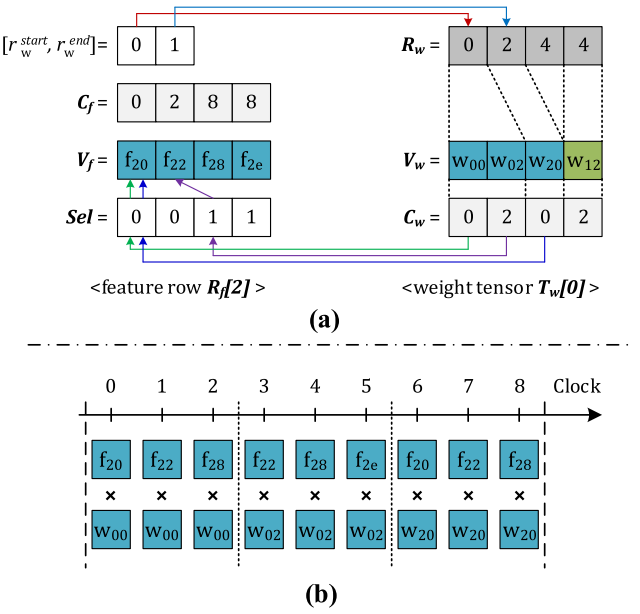


FIGURE 5. Illustration of 2-D convolution between a SCSR-based feature row and weight tensor: (a) input operands, (b) temporal operation.

V. OVERALL ARCHITECTURE

A. PE ARCHITECTURE

The PE specifically designed for the 2-D convolution of a *SCSR*-formated feature row with a *SCSR*-formated weight

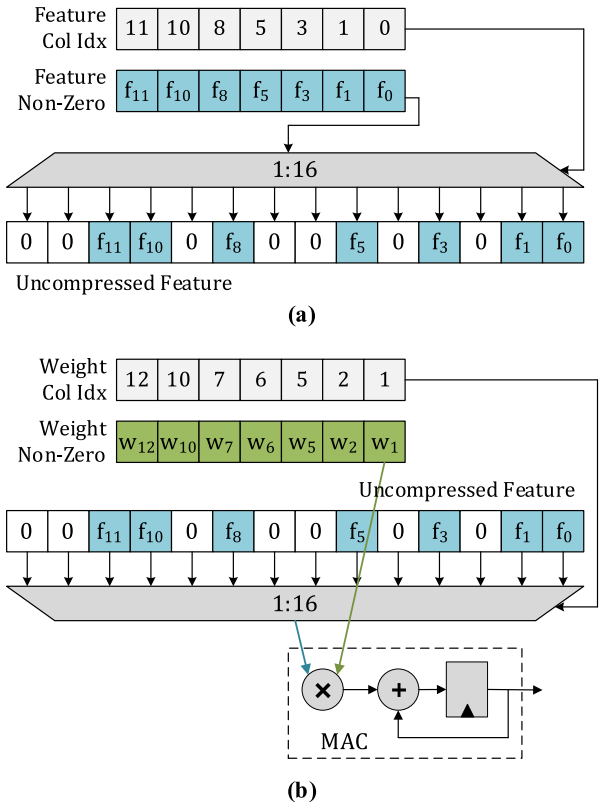


FIGURE 6. Operations on fully connected layers: (a) zero insertions of a compressed feature vector, (b) dot product of one-sided sparsity.

tensor, as described in Section IV, is illustrated in FIGURE 7. It consists of several key components: an input feature buffer, a weight buffer, an output buffer, a MAC unit, an address calculator, and a PE controller. The primary function of the PE is to compute the final sums of an output feature array within an output channel. As a result, the output buffer continually stores partial sums of the assigned output features during the convolution process across all input channels. Simultaneously, the input feature and weight buffers are periodically updated with the required input data. To minimize input data latency, the input buffers are doubled in size. In this architectural design, the input feature buffer size is of 2×16 elements, the weight buffer has dimensions of 2×25 elements, and the output buffer is configured at 14×14 elements.

The Address Calculator is responsible for computing addresses for output partial sums, relying on the coordinate information of the input data. Consequently, the output feature value, specified by the output address, is accumulated with the product generated by the MAC unit. Meanwhile, the PE Controller serves as a control unit, responsible for receiving *event signals*, *address signals* from the input buffers, and *config signals* from the main controller of the accelerator. These inputs are used to determine the necessary input address selections for the input buffers.

The configuration data, provided to the PE Controller prior to inference, contains essential information regarding convolution parameters (such as stride and filter size) and mapping algorithm parameters (such as dimensions of the assigned input and output). The function of the PE controller is to calculate counters (cnt_w, cnt_f) to select valid weights and features in their respective buffers. he operation of the PE Controller is elucidated by Algorithm 3, which takes $\left[r_w^{start}, r_w^{end}\right]$ and *Sel* calculated in Algorithm 1 and 2 as inputs. The functions *find_index*() and *find_column*() can be simply implemented using *SCSR* format.

Algorithm 3 2-D Convolution Between a Feature Row and a Weight Tensor

input: r_w^{start}, r_w^{end}
$Sel\,[0 : l_w - 1]$
C_f^o, C_w^o
T_w
output: cnt_w, cnt_f

```
cnt_w = find_index(T_w, r_w^start)
cnt_w = find_index(T_w, r_w^stop)
while cnt_w ≤ find_index(T_w, r_w^end)
    r_w = find_column(T_w, cnt_w)
    cnt_f = Sel[r_w]
    while cnt_f ≤ C_f^o − C_w^o + r_w
        cnt_f += 1
    end while
    cnt_w += 1
end while
```

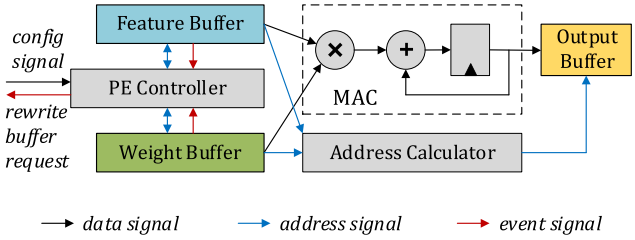


FIGURE 7. PE architecture.

Designating the row and column indices of an output feature as r_o, c_o, the subsequent equations serve to determine the address of the output feature.

$$r_o = \left(r_f - r_w\right) / s \tag{10}$$

$$c_o = \left(c_f - c_w\right) / s \tag{11}$$

B. WORKLOAD ALLOCATION

In FIGURE 8, we present the workload distribution for PEs within our designed architecture, featuring a 16×16 PE array. Each PE is tasked with computing the final sums of a subsection of output features within a channel. Consequently, it is necessary for the PE to read the weights of an entire filter and a subtensor of input features. These inputs are segmented into smaller units and then transferred to the PE. To minimize the number of memory accesses and take full advantage of

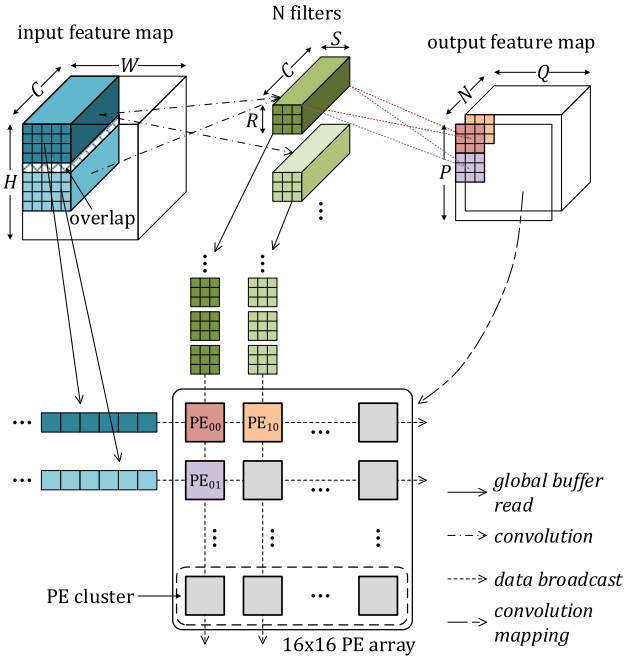


FIGURE 8. Workload assignment.

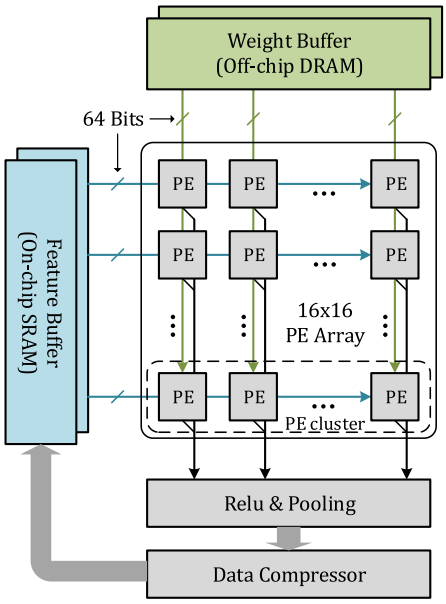


FIGURE 9. StarSPA accelerator's architecture.

the convolution process, we employ a broadcast approach for the input data, allowing multiple PEs within a row to access the same input feature rows and PEs within a column to access the same weight channels. This means that PEs located within the same row are responsible for generating output features for consecutive output channels, while PEs within the same column focus on producing output features for the same output channel. Given the variability in workload dimensions across layers within the same model and among different models, we have devised a mapping algorithm to ensure an equitable distribution of the workload among the PEs.

After receiving the requisite input data, the PEs independently perform 2-D convolutions, as elaborated in Section IV. Upon completing their computations, PEs initiate buffer swapping to continue their operations with fresh data. Simultaneously, the released buffers from PEs in the cluster are refilled with new data, effectively reducing data dependencies. Since each PE is responsible for generating complete sums of distinct output feature subsections, input feature tensors read by adjacent PEs exhibit data overlap, which in turn increases the number of required memory accesses. To mitigate this data overlap, we have structured the workload distribution among PEs in a preferably square configuration.

C. OVERALL ARCHITECTURE OF STARSPA

The overall architecture of the proposed *StarSPA* accelerator is illustrated in FIGURE 9. It comprises the PE array, an external Weight Buffer, an internal Feature Buffer, a ReLU & Pooling unit, and a Data Compressor. For simplicity, the control block is not depicted in the diagram. The sparse pretrained weights are compressed in the *SCSR* format and stored within the Weight Buffer, organized into banks. Each bank within the Weight Buffer broadcasts weights to a column of PEs via a 64-bit data bus. The output features generated by the PE array pass through the ReLU & Pooling block before undergoing compression by the Data Compressor. The Data Compressor utilizes mapping and stride information from the subsequent layer to apply appropriate compression to the output features. These compressed output features are then stored within the internal Feature Buffer, making them accessible as input features for PEs involved in computing the next layer. Similar to the Weight Buffer, the Feature Buffer is comprised of multiple banks that efficiently broadcast features to PEs situated in different rows. The size of the Feature Buffer, set at 1MB, is determined through simulations involving actual datasets and various widely-used CNN models, such as AlexNet [2], VGG16 [4], GoogLeNet [3], ResNet34 [32], MobileNetV1 [46], and EfficientNetV2 [47], all employing 8-bit data precision.

VI. EVALUATION

A. METHODOLOGY

The proposed design *StarSPA* has undergone comprehensive evaluation through a two-fold approach, encompassing an FPGA-based hardware implementation and a cycle-accurate simulator. The FPGA implementation provides insights into hardware dimensions, quantified in terms of Lookup Tables (LUTs) and Flip-Flops (FFs). Due to the time-intensive nature of developing the entire hardware accelerator and conducting simulations, the focus was directed towards implementing a PE cluster, comprising 16 PEs, implemented using SystemVerilog. The Register-Transfer Level (RTL) design underwent synthesis to ensure optimal

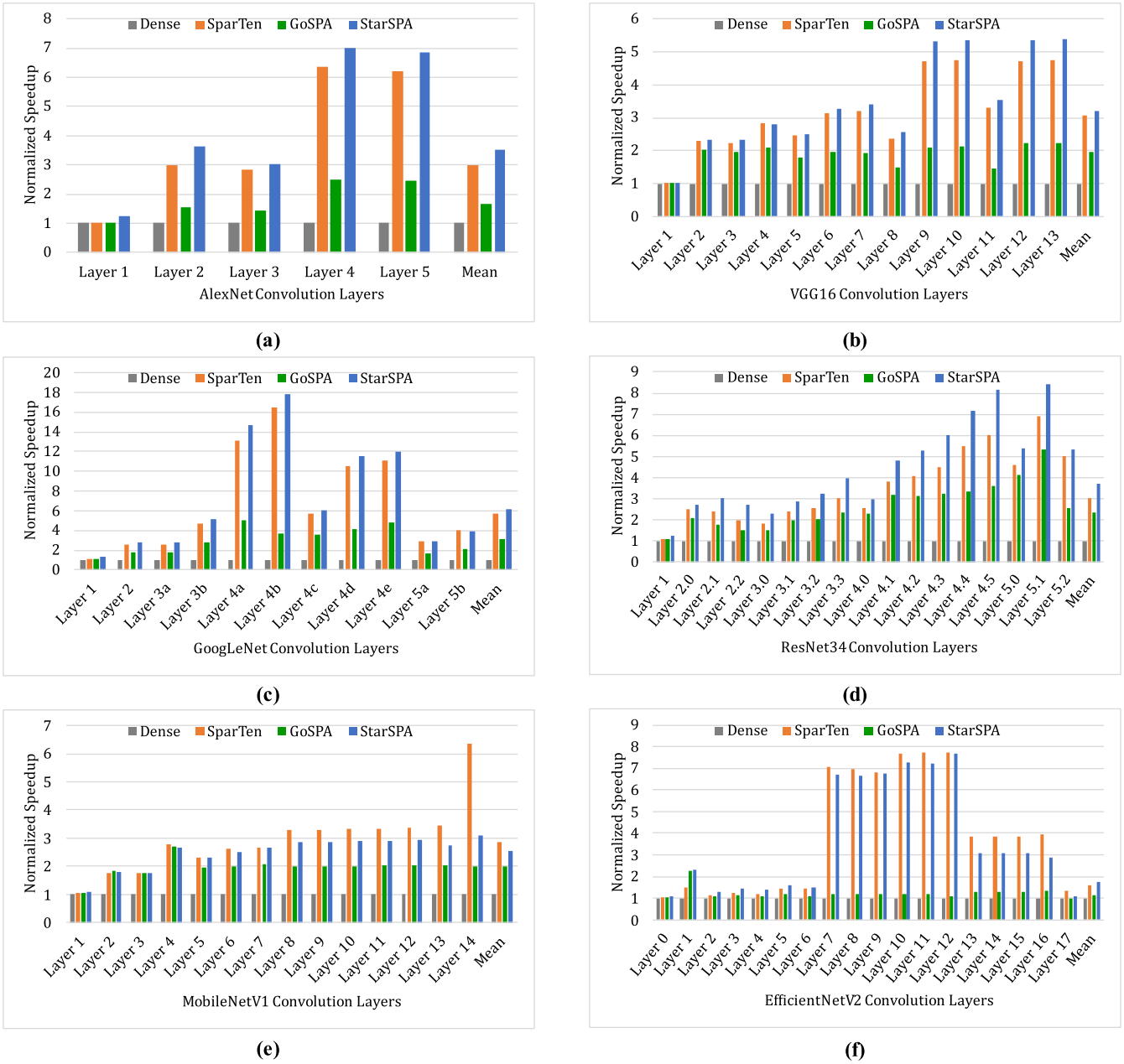


FIGURE 10. Comparisons on normalized speedup on convolution layers of (a) AlexNet, (b) VGG16, (c) GoogLeNet, (d) ResNet34, (e) MobileNetV1 and (f) EfficientNetV2 (baseline: Dense architecture).

performance, albeit at the potential increased expense of power consumption and hardware costs.

Moreover, the FPGA implementation allows for an in-depth examination of energy efficiency, which is computed as the product of the average power consumption and the time needed to complete a specified taskset. We generated various synthetic tasksets with diverse dimensions and sparsity levels to mirror the real dimensions of layers within CNN models. These tasksets span a range of sparsity rates from 0% to 90%. The reported energy efficiency figures represent the mean energy consumption required for performing inference on these generated tasksets.

The performance of *StarSPA* is empirically validated through inferences conducted on actual CNN models using the cycle-accurate simulator. This simulator, equipped with 256 MAC units, accurately quantifies the number of clock cycles essential for computations in each layer of the CNN models. The CNN models chosen for this experiment encompass well-known architectures, including AlexNet, VGG16, GoogLeNet, ResNet34, MobileNetV1, and EfficientNetV2. It is noteworthy that MobileNetV1 and EfficientNetV2 incorporate non-unit stride at both their initial and various intermediate layers, whereas AlexNet, VGG16, GoogLeNet, and ResNet34 exclusively utilize non-unit stride

in their initial layers. Notably, the models have undergone pruning and fine-tuning CIFAR10 dataset [48] to achieve commensurate accuracy levels with their dense counterparts. The average sparsity rates of the models used in this experiment are detailed in TABLE 3. Additionally, the simulator provides data on the number of memory accesses, contributing to the evaluation of the proposed architecture's energy efficiency.

TABLE 3. Benchmarks.

CNN Models	Weight sparsity	Feature sparsity	Unpruned accuracy	Pruned and quantized accuracy
AlexNet [2]	71.34%	24.68%	87.36%	85.62%
VGG16 [4]	64.05%	47.54%	90.38%	87.84%
GoogLeNet [3]	59.22%	33.17%	88.24%	85.54%
ResNet34 [32]	61.2%	42.02%	87.1%	86.16%
MobileNetV1 [46]	60.68%	48.82%	82.74%	80.76%
EfficientNetV2 [47]	51.3%	44.9%	84.52%	83.56%

B. SPEED UP ON ACTUAL CNN MODELS

The performance evaluation of *StarSPA* was carried out utilizing the cycle-accurate simulator. This evaluation encompassed the inference process on six distinct CNN models, as detailed in TABLE 3. The objective was to record the precise counts of clock cycles necessary for computations within each layer of these models. Typically, for the initial layers of these CNN models, a conventional practice involves the deployment of an external data processing unit to partition and store the input image into the feature buffer of the accelerator. In the case of *StarSPA*, the input image is divided into manageable subsections through the internal Data Compressor. These subsections were subsequently stored in the Feature Buffer. Given that the input images were inherently dense, the Data Compressor's operation involved selecting pixels at stride steps for concurrent storage in the buffer. Subsequently, the PEs carried out the actual convolution process on the preprocessed input image.

To offer a comprehensive perspective on the enhanced performance of the proposed architecture, we conducted implementations and simulations of other state-of-the-art architectures, specifically Sparten [27] and GoSPA [15]. Furthermore, we included a dense architecture in our comparative analysis, which was conceptually similar to *StarSPA* accelerator but lacked the sparsity-exploiting features. All of these architectures were uniformly configured with identical parameters, featuring 256 MAC units and 1MB of SRAM designated for feature storage. The recorded counts of clock cycles required for inference processing across the layers of each CNN model, employing these architectures, were subjected to normalization regarding the dense architecture.

FIGURE 10 provides a comparative analysis of speedup factors for convolution layers across various accelerators applied to the CNN models. In most cases, our work outperformed other architectures. The ranking of performance,

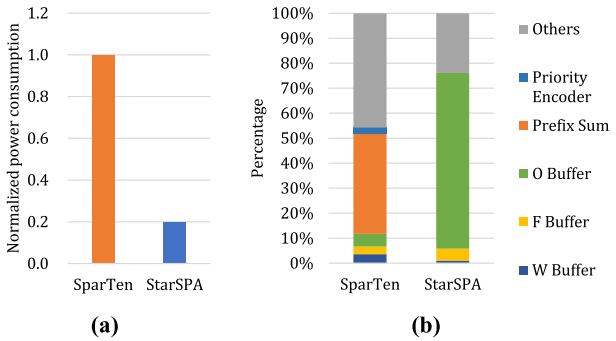


FIGURE 11. Energy consumption on Zynq-Ultrascale+ FPGA (a) Normalized energy efficiency and (b) Energy breakdown.

from highest to lowest, was as follows: *StarSPA*, SparTen, GoSPA, and the dense architecture. To delve into specifics, our work achieved speedup factors of 1.17×, 1.05×, 1.09×, 1.23×, and 1.12× in comparison to SparTen for AlexNet, VGG16, GoogLeNet, ResNet34, and EfficientNetV2, respectively.

It is important to note that, in the case of MobileNetV1, *StarSPA* exhibited slightly lower performance compared to SparTen. This deviation can be attributed to the architectural specifics of MobileNetV1, which incorporates separable depthwise convolutions. These separable depthwise convolutions consist of a depthwise convolution layer followed by a pointwise convolution layer. *StarSPA* demonstrated efficient hardware utilization in depthwise convolution layers, where the dataflow fully leveraged buffered inputs. However, in the pointwise convolution layers, *StarSPA* buffered only a single weight representing an entire weight channel in each cycle, which could lead to underutilization when the buffered weight value was zero. In contrast, the pointwise dataflow within SparTen proved more effective in these scenarios.

C. AREA EFFICIENCY

To evaluate the area efficiency of *StarSPA*, we implemented a Register-Transfer Level (RTL) design for a single PE cluster using the SystemVerilog programming language. This cluster consisted of 16 PEs arranged in the same row within the PE array. Each individual PE was equipped with an 8-bit multiplier and a 24-bit accumulator. To facilitate a meaningful comparison, we developed an RTL design for a SparTen's compute cluster, ensuring it featured an equivalent number of multipliers. Specific details regarding buffer parameters for both architectures are provided in TABLE 4. The synthesis of the RTL designs for both accelerators was conducted on a Zynq-Ultrascale+ FPGA [49]. The synthesis options were optimized with a preference for performance. It is noteworthy that the MAC units were synthesized using the FPGA's LUTs and FFs, without the need to utilize Digital Signal Processing (DSP) units.

TABLE 5 presents a comprehensive breakdown of the FPGA resources used in our evaluation. Notably, our proposed architecture's elimination of the need for an input

FIGURE 12. Comparisons on normalized SRAM accesses on convolution layers of (a) AlexNet, (b) VGG16, (c) GoogLeNet, (d) ResNet34, (e) MobileNetV1 and (f) EfficientNetV2 (baseline: Dense architecture).

preparation stage resulted in a significant reduction in the total number of employed LUTs, showing an impressive 4.55× decrease. This figure decreased from 175270 LUTs in SparTen to only 38550 in *StarSPA*. Furthermore, the total number of employed FFs experienced a notable reduction of 1.74×, dropping from 162664 in SparTen to 93749 in our work. The decrease in LUTs can be attributed primarily to the removal of an input searching circuit within *StarSPA*. Simultaneously, the reduction in FFs in *StarSPA* can be traced back to the decreased sizes of the PE buffers, as detailed in TABLE 4.

D. ENERGY EFFICIENCY

To assess the energy efficiency of *StarSPA*, synthetic task sets, comprising 3-D input feature maps and 3-D filters, serve as inputs for running simulations on the post-synthesis netlists. Switching Activity Interchange Format (SAIF) files were generated to create power reports. The energy required to complete a taskset is computed by multiplying the average power consumption by the duration needed to finish the taskset. Various tasksets were generated, each with varying dimensions and variable sparsity levels ranging from 0% to 90%. The reported energy consumption

TABLE 4. PE's buffer parameters.

Parameters	SparTen	StarSPA
Input feature buffer size (bytes)	128 × 2* = 256	16 × 2* = 32
Weight buffer size (bytes)	128 × 2* × 2§ = 512	25 × 2* = 50
Output buffer size (bytes)	32 × 2* × 2§ × 3¶= 384	196 × 3¶ = 558
Total buffer size per PE (bytes)	1,152	640

*Double buffering, §Collocated buffer, ¶Size of a partial sum in bytes.

TABLE 5. Numbers of FPGA primitives in synthesized PE clusters.

FPGA primitives	Breakdown of FPGA primitives	SparTen	StarSPA
# of LUTs	Prefix Sum	50080	0
	Priority Encoder	48352	0
	W Buffer	30464	5644
	F Buffer	15232	3424
	O Buffer	14921	27804
	Others	16221	1678
	Total	175270	38550
# of FF	Prefix Sum	288	0
	Priority Encoder	0	0
	W Buffer	73856	9289
	F Buffer	36928	6560
	O Buffer	49160	72444
	Others	2432	5456
	Total	162664	93749

represents the average value derived from the results obtained by simulating these tasksets. The energy efficiency of the proposed design was compared to that of Sparten. Both architectures were synthesized and verified to operate correctly at a clock speed of 100 MHz.

In FIGURE 11(a), there was a significant 5.0× reduction in energy consumption observed in *StarSPA* when compared to Sparten. This reduction can be attributed to the absence of an input matching circuit in *StarSPA*. FIGURE 11(b) provides a detailed breakdown of the influence of various components on the overall energy consumption within a single PE cluster. Notably, the most significant energy-consuming component in SparTen was the prefix sum, whereas in *StarSPA*, the primary contributor to energy consumption was associated with its output buffer. This distinction can be explained by the fact that *StarSPA* operates under a weight-stationary dataflow, which necessitates frequent queries to the output buffer on each clock cycle.

In addition to the advancements in terms of hardware size and energy efficiency, *StarSPA*'s dataflow resulted in a significantly reduced number of memory accesses compared to SparTen. FIGURE 12 illustrates the normalized counts of memory accesses caused by *StarSPA*, SparTen, GoSPA in comparison to the dense architecture. It is evident that SparTen incurred a substantial volume of SRAM accesses, primarily due to the limited reuse of locally buffered data. In contrast, GoSPA and *StarSPA* showed similar quantities of SRAM accesses, as both accelerators efficiently managed data access, reading and flushing data only when necessary for all potential computations. The average counts of SRAM accesses attributed to SparTen during simulations of AlexNet, VGG16, GoogLeNet, ResNet34, MobileNetV1, and EfficientNetV2 were 9.4×, 7.3×, 2×, 5.6×, 1.8×, and 2.23×, respectively, when compared to *StarSPA*. The significant reductions in SRAM accesses observed for AlexNet, VGG16, and ResNet34 can be attributed to their relatively wide filters, which facilitate more effective feature reuse in our architecture. In contrast, GoogLeNet and MobileNetV1, which incorporate numerous pointwise convolution layers with 1 × 1 filters, experienced the least reduction in SRAM accesses.

VII. CONCLUSION

This paper introduces a novel data compression method known as *Stride-Aware Compressed Sparse Row (SCSR)*, designed to alleviate the computational load of input searching in sparse CNN acceleration. What sets this approach apart from previous research is that it compresses the output features of a layer based on the parameters of its subsequent layer. Consequently, the input features are fully prepared for computation in the following layer, eliminating the need for expensive input searching circuits. The combination of a weight-stationary dataflow and asynchronous swapping of the doubled input buffers within the proposed architecture leads to exceptional PE utilization, resulting in enhanced inference speed. However, despite the significant advancements achieved by this architecture compared to state-of-the-art accelerators, it is acknowledged that the application of a weight-stationary dataflow results in a substantial number of queries to the output buffer. Therefore, further research is needed to mitigate the reading costs associated with the output buffer.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," presented at the 25th Int. Conf. Neural Inf. Process. Syst., vol. 1, 2012.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[5] X. Zhu, K. Guo, H. Fang, L. Chen, S. Ren, and B. Hu, "Cross view capture for stereo image super-resolution," *IEEE Trans. Multimedia*, vol. 24, pp. 3074–3086, 2022, doi: 10.1109/TMM.2021.3092571.

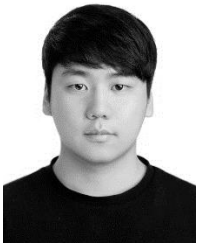
[6] X. Zhu, K. Guo, S. Ren, B. Hu, M. Hu, and H. Fang, "Lightweight image super-resolution with expectation-maximization attention mechanism," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 1273–1284, Mar. 2022, doi: 10.1109/TCSVT.2021.3078436.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12 pp. 2493–2537, Aug. 2011.

[8] W. Huang, Z. Wu, P. Mitra, and C. L. Giles, "RefSeer: A citation recommendation system," in *Proc. IEEE/ACM Joint Conf. Digit. Libraries*, Sep. 2014, pp. 371–374, doi: 10.1109/JCDL.2014.6970192.

[9] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 1, 2021, Art. no. 241.
[10] A. K. Jameil and H. Al-Raweshidy, "Efficient CNN architecture on FPGA using high level module for healthcare devices," *IEEE Access*, vol. 10, pp. 60486–60495, 2022, doi: 10.1109/ACCESS.2022.3180829.
[11] K. Fukushima, "Neocognitron," *Scholarpedia*, vol. 2, no. 1, p. 1717, 2007. [Online]. Available: http://www.scholarpedia.org/article/Neocognitron
[12] S. Kang, G. Park, S. Kim, S. Kim, D. Han, and H.-J. Yoo, "An overview of sparsity exploitation in CNNs for on-device intelligence with software-hardware cross-layer optimizations," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 634–648, Dec. 2021, doi: 10.1109/JETCAS.2021.3120417.
[13] M. A. Qureshi and A. Munir, "Sparse-PE: A performance-efficient processing engine core for sparse convolutional neural networks," *IEEE Access*, vol. 9, pp. 151458–151475, 2021, doi: 10.1109/ACCESS.2021.3126708.
[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, Oct. 2016, doi: 10.1145/3007787.3001163.
[15] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "GoSPA: An energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1110–1123, doi: 10.1109/ISCA52012.2021.00090.
[16] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13, doi: 10.1109/ISCA.2016.11.
[17] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 1059–1071.
[18] 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, doi: 10.1109/JETCAS.2019.2910232.
[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[20] 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.
[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," presented at the 27th Int. Conf. Int. Conf. Mach. Learn., 2010.
[22] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 27–40, doi: 10.1145/3079856.3080254.
[23] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.
[24] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, "Soft threshold weight reparameterization for learnable sparsity," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 5544–5555.
[25] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6071–6079.
[26] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2, 1989, pp. 598–605.
[27] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," presented at the 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture, Columbus, OH, USA, Oct. 2019, doi: 10.1145/3352460.3358291.
[28] S. Girish, K. Gupta, S. Singh, and A. Shrivastava, "LilNetX: Lightweight networks with extreme model compression and structured sparsification," 2022, *arXiv:2204.02965*.
[29] X. Xie, J. Lin, Z. Wang, and J. Wei, "An efficient and flexible accelerator design for sparse convolutional neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 7, pp. 2936–2949, Jul. 2021, doi: 10.1109/TCSI.2021.3074300.
[30] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," 2021, *arXiv:2104.08378*.
[31] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, "Learning N:M fine-grained structured sparse neural networks from scratch," 2021, *arXiv:2102.04010*.
[32] 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.
[33] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," 2020, *arXiv:2003.02389*.
[34] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.
[35] 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.
[36] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019, doi: 10.1109/TNNLS.2018.2852335.
[37] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12, doi: 10.1109/MICRO.2016.7783723.
[38] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, "Tensaurus: A versatile accelerator for mixed sparse-dense tensor computations," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 689–702, doi: 10.1109/HPCA47549.2020.00062.
[39] 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, doi: 10.1109/HPCA47549.2020.00015.
[40] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "ExTensor: An accelerator for sparse tensor algebra," presented at the 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture, Columbus, OH, USA, Oct. 2019, doi: 10.1145/3352460.3358275.
[41] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "SpArch: Efficient architecture for sparse matrix multiplication," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 261–274.
[42] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "OuterSPACE: An outer product based sparse matrix multiplication accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 724–736, doi: 10.1109/HPCA.2018.00067.
[43] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "MatRaptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 766–780, doi: 10.1109/MICRO50266.2020.00068.
[44] J. Yang, W. Fu, X. Cheng, X. Ye, P. Dai, and W. Zhao, "S^2 Engine: A novel systolic architecture for sparse convolutional neural networks," *IEEE Trans. Comput.*, vol. 71, no. 6, pp. 1440–1452, Jun. 2022, doi: 10.1109/TC.2021.3087946.
[45] N.-S. Pham and T. Suh, "Optimization of microarchitecture and dataflow for sparse tensor CNN acceleration," *IEEE Access*, vol. 11, pp. 108818–108832, 2023, doi: 10.1109/ACCESS.2023.3319727.
[46] 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, *arXiv:1704.04861*.
[47] M. Tan and Q. Le, "EfficientNetV2: Smaller models and faster training," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10096–10106.
[48] V. N. A. Krizhevsky and G. Hinton. *The CIFAR-10 Dataset*. Accessed: Aug. 14, 2023. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html
[49] Xilinx. (2021). *Vivado*. [Online]. Available: https://www.xilinx.com/support/download.html

NGOC-SON PHAM (Member, IEEE) received the B.S. degree from Vietnam National University, Hanoi, Vietnam, in 2011, and the M.S. and Ph.D. degrees in electrical and electronics engineering from Chung-Ang University, Seoul, South Korea, in 2019. He is currently a Research Professor with the Department of Computer Science and Engineering, Korea University. His research interests include mixed-signal integrated circuit design, embedded and real-time systems, machine learning accelerators, and security.



SANGWON SHIN (Student Member, IEEE) received the B.S. degree in computer science and engineering from Korea University, Seoul, South Korea, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering. His research interests include hardware architecture, security, and cryptography.

LEI XU (Member, IEEE) received the Ph.D. degree from the Chinese Academy of Sciences, China. He is currently an Assistant Professor with the Department of Computer Science, Kent State University. His research interests include applied cryptography, cloud/mobile security, and decentralized systems.

WEIDONG SHI (Senior Member, IEEE) is currently an Associate Professor with the University of Houston. His research interests include high-performance computer architecture for large-scale multimedia applications, applied security, real-time computer graphics, blockchain, and cloud computing.

TAEWEON SUH (Member, IEEE) received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea, in 1993, the M.S. degree in electronics engineering from Seoul National University, in 1995, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2006. He is currently a Professor with the Department of Computer Science and Engineering, Korea University.