Multiply-and-Fire (MnF): An Event-driven Sparse Neural Network Accelerator

MIAO YU∗ and TINGTING XIANG†, National University of Singapore, Singapore
VENKATA PAVAN KUMAR MIRIYALA†, National University of Singapore, Singapore
TREVOR E. CARLSON, National University of Singapore, Singapore

Deep neural network inference has become a vital workload for many systems, from edge-based computing to data centers. To reduce the performance and power requirements for DNNs running on these systems, pruning is commonly used as a way to maintain most of the accuracy of the system while significantly reducing the workload requirements. Unfortunately, accelerators designed for unstructured pruning typically employ expensive methods to either determine non-zero activation-weight pairings or reorder computation. These methods require additional storage and memory accesses compared to the more regular data access patterns seen in structurally pruned models. However, even existing works that focus on the more regular access patterns seen in structured pruning continue to suffer from inefficient designs, which either ignore or expensively handle activation sparsity leading to low performance.

To address these inefficiencies, we leverage structured pruning and propose the multiply-and-fire (MnF) technique, which aims to solve these problems in three ways: (a) the use of a novel event-driven dataflow that naturally exploits activation sparsity without complex, high-overhead logic; (b) an optimized dataflow takes an activation-centric approach, which aims to maximize the reuse of activation data in computation and ensures the data are only fetched once from off-chip global and on-chip local memory; (c) Based on the proposed event-driven dataflow, we develop an energy-efficient, high-performance sparsity-aware DNN accelerator. Our results show that our MnF accelerator achieves a significant improvement across a number of modern benchmarks and presents a new direction to enable highly efficient AI inference for both CNN and MLP workloads. Overall, this work achieves a geometric mean of $11.2 \times$ higher energy efficiency and $1.41 \times$ speedup compared to a state-of-the-art sparsity-aware accelerator.

CCS Concepts: • Hardware → Hardware accelerators; • Computer systems organization → Neural networks.

Additional Key Words and Phrases: Deep neural networks, hardware accelerator, energy-efficient accelerator, architecture, sparse processing, dataflow, event-driven

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have seen significant improvement in accuracy and an increase in applications on edge systems like mobile phones and Internet of Things (IoT) devices. They have become the key approach to solving tasks such as object detection, image classification and analysis, and semantic segmentation [30]. However, it is common for these deep networks to have multiple layers, millions of parameters, billions of operations and requires tremendous storage and intense computation resources [16, 27, 33, 43, 48], making it difficult to

∗Miao Yu and Tingting Xiang share equal contributions to the research.
New Paper, not an Extension of a Conference Paper.
†Venkata Pavan Kumar Miriyala completed this work at National University of Singapore and is currently employed at Advanced Micro Devices, Inc.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2023 Copyright held by the owner/author(s).
1544-3566/2023/10-ART
https://doi.org/10.1145/3632817

ACM Trans. Arch. Code Optim.
meet the energy-efficiency and performance requirements of resource-constrained devices. To address these issues, several model compression techniques [10, 19, 20, 25, 36, 52], efficient dataflow techniques [39, 46], and accelerators [1, 2, 4, 7–9, 17, 18, 23, 38, 54–57, 59] have been proposed and widely investigated in recent years.

In particular, exploiting sparsity by pruning DNNs has emerged as a promising approach for achieving energy-efficient and high-performance DNN solutions [1, 2, 4, 7–9, 17, 18, 23, 38, 54–57, 59]. Pruning methods can be broadly classified as either unstructured or structured, producing unstructured sparsity (zeros are distributed in an irregular pattern) or structured sparsity (zeros are in a regular pattern) in the model weights. A number of sparsity-aware DNN accelerators have been proposed to leverage unstructured pruning [1, 2, 4, 7, 8, 17, 18, 23, 38, 54–57] due to its ability in achieving a high compression ratio while maintaining model accuracy. However, exploiting unstructured sparsity tends to require complex hardware designs and current accelerators face two key challenges in handling this type of sparsity.

The first challenge is the irregularity seen when accessing the data. To support irregular activation and weight sparsity introduced by unstructured pruning, complex logic is required to pair non-zero values, which incurs additional power and area cost. For example, the prefix sum and priority encoder used to exploit sparsity consume 62.7% and 46% of the total area and power of Sparten [17], respectively. In addition, irregularity in activation and weight can cause the partial sums to be unevenly distributed. Previous works [50, 55] suffer from memory contention to update these sums, resulting in compute stalls and low utilization of the multiplier array. To achieve a better utilization rate with sparse workloads, STICKER [55] requires offline software optimization to rearrange the input activations.

Second, we see that workload imbalance can be a significant challenge for accelerators. Sparsity in weights and activations across filters, channels, and layers can vary greatly which results in imbalanced workload distribution across compute units. Units with smaller workloads must remain idle while waiting for other compute units with higher workloads to complete the computation, leading to a low utilization rate. For instance, in SCNN [38] the multiplier utilization rate falls below 60% when the overall weight and activation sparsity is more than 60% in a convolutional layer.

As an alternative, recent work [35] has shown that structured pruning can achieve comparable accuracy and be more competitive when computing CNNs, in terms of both computational efficiency and storage, compared to unstructured pruning. In addition, structured pruning produces regular data sparsity which simplifies memory access patterns and hardware complexity. To leverage the advantage of structured pruning, recent sparsity-aware DNN accelerators [21, 59] take a hardware-software co-design approach to reduce data irregularity and benefit from structured sparsity. These works propose structured pruning, which greatly reduces weight irregularity and processing complexity. However, a major problem, even when using structured pruning, remains: dynamic and irregular activation sparsity is still handled inefficiently. Recent work [59] continues to store and access zero-valued activations from off-chip memory, resulting in unnecessary access energy and data transfer. CSP [21] is another example of recent work that reduces weight irregularity to enable sequential access of activations, thus, reducing power-consuming off-chip activation accesses. Unfortunately, this previous work is unable to skip both the access and computation of zero-valued activation. This leads to low effectual PE utilization, ineffectual zero-valued multiplications and large overheads when accessing input data. Overall, prior works are unable to both reduce irregularity and eliminate unnecessary computation and memory accesses of zero-valued data while maintaining high PE utilization and efficiency. A solution that can accomplish each of these goals is needed to build a highly efficient sparsity-aware system.

To overcome the limitations in sparsity-aware DNN accelerators and to achieve energy-efficient DNN inference in edge devices, we leverage structured pruning and propose our novel Multiply-and-Fire (MnF) technique. MnF presents an event-driven dataflow that supports structured sparsity for convolution layers and both structured and unstructured sparsity for fully connected layers. In this event-driven dataflow, we take an activation-centric approach and view one convolution operation as **one activation (an event) multiplied by all relevant weights of**
the filters instead of the typical vector-vector multiplication or matrix-vector multiplication methods. Similarly, in fully connected layers, we view computation as one activation multiplied by all relevant weights of neurons. This technique allows sparsity to be handled efficiently in a number of ways. (1) As all the computations of a single non-zero activation can be performed in parallel, that activation is maximally reused and only needs to be read from off-chip (global DRAM) and on-chip (local SRAM) memory once. Moreover, weights and partial sums are highly reused from the low-cost local memory in the PE. Memory accesses are a significant source of energy consumption in prior works [15, 18, 21]; minimizing these accesses increases overall energy efficiency. (2) The proposed novel event-driven dataflow exploits irregular activation sparsity naturally and avoids ineffectual zero-valued multiplication without complex logic overhead like the prefix sum and priority encoder implemented in SparTen [17] or the content addressable memory (CAM) used by Extensor [23], translating to both energy and area gains. (3) In the convolution layer, both kernel level parallelism (two-dimensional weight matrix each assigned to a channel of input activations) and filter level parallelism (three-dimensional matrix consisting of multiple kernels) are exploited. In the fully connected layer, output neuron parallelism is exploited. With our proposed efficient hardware, multipliers can maintain >90% utilization rate across most sparsity levels.

Overall, the main contributions of this work are as follows:

- We propose an event-driven dataflow, the Multiply-and-Fire (MnF) methodology, which can support structured sparsity in convolution layers and both structured and unstructured sparsity in fully connected layers. It exploits activation sparsity and significantly improves activation reuse compared to the traditional dataflow techniques (See Section 3).
- We propose a sparsity-aware accelerator which provides high energy efficiency and high performance by leveraging the proposed event-driven dataflow to allow one-time memory access to activation data. It is designed to exploit both kernel-wise and filter-wise parallelism and enables a single-cycle processing time of activation data (See Section 4).
- We present a thorough study of modern dataflow techniques and use a variety of sparsity levels and convolutional layers to demonstrate the advantages of MnF (See Section 2 and 5).
- Finally, we perform a detailed study and performance comparison with related works. The proposed accelerator surpasses a recent sparsity-aware DNN accelerator, CSP [21], by a geometric mean of 11.2× on the evaluated models in terms of energy efficiency (inferences/J) and a geometric mean of 1.41× speedup (inferences/second) on most of the evaluated models (See Section 5).

2 BACKGROUND AND MOTIVATION

Deep neural networks (DNN) typically consist of numerous layers and a vast number of parameters, leading to considerable computational and storage requirements. These demands pose significant challenges in meeting high-performance and energy-efficiency requirements of devices intended for deployment in resource-constrained environments. In [14], it was demonstrated that these DNNs are often over-parameterized, meaning that they have more parameters than necessary to perform their tasks. As a result, there is considerable redundancy within these models. To address this issue, a promising compression technique called pruning [3, 19, 34, 36, 52] has emerged, which involves removing less important weights from the DNNs. The resulting sparse DNN retains a similar accuracy as the dense model but has a significantly reduced model size with much lower processing requirements [3, 20]. A number of prior works are designed to benefit from sparse DNNs to achieve better performance and energy efficiency [1, 2, 4, 7–9, 17, 18, 23, 38, 54–57, 59]. In the following subsections, we first present the basic concept of sparsity in DNNs then discuss the sparsity-aware accelerator, and finally, the motivation of this work.

ACM Trans. Arch. Code Optim.
2.1 Sparsity in Deep Neural Networks
Pruning introduces regular or irregular sparsity in weight parameters by applying structured or unstructured techniques, respectively. Unstructured pruning is able to achieve a high compression ratio while maintaining model accuracy. However, it introduces irregular weight sparsity as weights are randomly removed across the model [20], which can pose challenges in hardware accelerator design. Structured pruning, on the other hand, has a lower compression ratio but produces regular weight sparsity which is more hardware-friendly. In addition to pruning, the rectified linear activation function (ReLU) used in DNN layers also introduces irregular activation sparsity [28]. Weight-level sparsity is static, as neural network parameters are known before beginning the computation. But activation-level sparsity is dynamic and can only be known during the execution of specific input data. Due to such irregular and dynamic sparsity, it is difficult to reap the benefits of sparse models through traditional hardware solutions such as CPUs, GPUs, TPUs, and ASIC DNN accelerators. As a result, unique sparsity-aware DNN accelerators are required to handle sparsity efficiently and avoid unnecessary zero-valued multiplications between activations and weights [1, 5–8, 12, 15, 21, 59].

2.2 Sparsity-Aware Accelerator
Many prior accelerators aim to benefit from the high compression ratio of unstructured pruning by designing specific dataflow and hardware to handle irregular weight and activation sparsity [1, 7, 8, 12]. However, they cannot fully benefit from the high compression ratio due to the irregularity in the sparse data. More recent works shift the focus to structured pruning to leverage the regular sparsity in weights [21, 32, 45, 59]. Although the compression ratio of structured pruning can be lower than unstructured pruning, the storage requirement of the structurally pruned model can be smaller or comparable as additional indices are not required to identify sparse data in the compressed format [35]. Both prior structured and unstructured sparsity-aware accelerators manage to improve the performance of sparse processing, but they suffer from a number of drawbacks, such as ineffectual computations and costly pairing logic. In the following, we discuss the designs and drawbacks of two representative works.

2.2.1 Unstructured Sparsity-Aware Accelerator. Eyeriss v2 [8] is one representative work that handles both irregular weight and activation sparsity using the row stationary dataflow proposed in Eyeriss [7]. It stores and processes sparse data in a compressed sparse column (CSC) format with a hierarchical mesh topology network-on-chip (NoC). Although Eyeriss v2 is able to avoid the storage and access of zero-valued data and skip zero-valued multiplications, compressed data processing introduces several challenges. To represent the sparse data in the CSC format, the count and address value of each piece of data has to be stored, which adds memory overhead in this compression scheme. As shown in Fig. 1, in order to identify the data, the CSC format requires 24 extra values to be stored with the 11 non-zero activations and weights. To process CSC data, read dependencies are introduced in the dataflow, where the address of the data has to be read before the actual value for both activations and weights. The input activations have to be read before weights in order to find the right matching pairs. This dependency introduces unnecessary input access as non-zero activations must still be read from memory even if all corresponding weights are zero. In addition, to support sparsity, the control logic, and extra memories consume an additional 73% area cost compared to the original Eyeriss implementation with limited sparsity support. The control logic in the processing element alone (without multiply and accumulate logic) consumes an average of 25% of the power.

2.2.2 Structured Sparsity-Aware Accelerator. Since one of the key challenges of sparsity-aware DNN accelerators is to efficiently handle irregular sparsity, Cascading Structured Pruning (CSP) [21] employs a hardware-software co-design methodology to regularise weight sparsity. Although CSP avoids complex logic to handle irregular weight sparsity in its implementation, activation sparsity is not exploited. Since activation sparsity resulting from the ReLU function is typically around 50% [28], not exploiting activation sparsity can lead to around 50% more than necessary off-chip and on-chip data accesses. In addition, zero-valued activations are computed in CSP, and these
Fig. 1. A one-channel convolution example of the Eyeriss v2 dataflow. One input feature map (IFM) is multiplied with two kernels with unstructured sparsity to generate two output feature maps (OFMs). In order to represent the 11 non-zero activations and weights in CSC format, at least 24 extra values (highlighted in red) need to be stored and accessed. The input activations have to be read before the weights in order to correctly match the non-zero values. Therefore, when all corresponding weights are zero, non-zero activations (in the gray boxes) are still required to be read from memory. It should be noted that the zero values shown on the right of the figure are for illustration purposes and Eyeriss v2 does not access and calculate zero value activations and weights.

computations are ineffectual since they are multiplications with zeros. Fig. 2 shows how CSP accesses zero-valued activations (highlighted in red and 9 gray boxes in the first step) and calculates each accessed value using the corresponding weights. From a theoretical perspective, these ineffectual computations are unnecessary and incur unnecessary power and energy overheads.

CSP makes an observation that activation data re-fetch from the off-chip and on-chip global memory consumes a significant amount of energy. It is therefore designed to eliminate the re-fetch of activation data from the off-chip memory. By preserving more data reuse opportunities, CSP has achieved better energy efficiency than the prior work, Cambricon-S [59]. However, the dataflow still requires activation data to be accessed multiple times from the on-chip memory. Because the pruning algorithm of CSP prunes weights in the same position from one kernel to all later kernels, the positions of non-zero weights in kernel2 (as seen in Fig. 2) can only be the subset of kernel1. Then, the subset of activations (12 gray boxes in the second step) accessed with kernel1 must be re-accessed from on-chip memory to process with kernel2. This leads to a high number of activation accesses and reduces energy efficiency.

2.3 Motivation

To leverage the benefits of pruning, the sparsity-aware accelerators must be able to process sparsity efficiently. However, prior unstructured sparsity-aware accelerators face multiple challenges brought about by the irregularity in both weight and activation data. The key challenges include memory overhead and the complex and expensive hardware logic to find matching pairs of non-zero weight and activation as seen in Section 2.2.1. Recent structured sparsity-aware accelerators target a hardware-software co-design to reduce irregularity in weights, but they are unable to fully exploit activation sparsity and skip all ineffectual multiplications. This work aims to overcome these limitations with a novel event-driven dataflow and energy-efficient accelerator.
In convolution layers, the proposed event-driven dataflow targets structured pruning instead of unstructured pruning. As shown in [35], although structured pruning has a lower compression ratio, it can be more competitive than unstructured pruning in terms of storage requirement and computational efficiency. With the same accuracy and quantization method, the compression ratio of structured pruning is about 1.14× to 2.56× lower than unstructured pruning. But the storage requirement of the structurally pruned model is in most cases 1.70× to 3.00× smaller than the unstructurally pruned model [35]. In terms of computation efficiency, recent work [35] has found that the weight reduction to speed up ratio is approximately 1.0 for structurally pruned models, which means an x times weight reduction can result in the same x times speedup. While in unstructurally pruned models, the ratio is between 2.7 to 3.5 which means the high compression ratio does not translate to high speedup in hardware. Since structured pruning results in regular sparsity and is comparable to unstructurally pruned models in terms of accuracy and storage, it is leveraged in this work. In fully connected layers, the proposed dataflow is capable of supporting both structured and unstructured pruning.

Overall, the proposed event-driven dataflow can handle activation sparsity efficiently without complex, high-overhead logic. It allows all ineffectual multiplications to be skipped and significantly improves the reuse of activation data to achieve low overall energy consumption. The implemented accelerator leverages the dataflow and enables a highly parallel computation of each activation with high energy efficiency and performance.

3 MULTIPLY-AND-FIRE (MNF) : DATAFLOW

In this section, we present our novel event-driven dataflow for MnF and outline its two distinct phases: the multiplication phase and the firing phase. A simple example is used as an overview of the MnF event-driven dataflow. This example demonstrates how the proposed design aims to solve the two major problems mentioned in Section 2: (1) the complex and high-overhead logic in finding matching pairs of activations and weights; and (2)
the inability to fully exploit activation sparsity and skip all ineffectual computations. Meanwhile, to remove data irregularity and avoid additional memory overhead in unstructurally pruned models, we leverage the structured pruning techniques from previous works [31, 34]. As a result, pruned weights are in a regular shape and can be stored and accessed in a dense format without incurring any memory overhead.

As shown on the left of Fig. 3, each non-zero activation is seen as an event. The events are numbered according to the order in which they are sent. All zero-valued activations are not seen as events, and therefore not involved in the computation. The multiply phase is illustrated in the middle of Fig. 3. The dataflow takes an activation-centric approach in which we focus on processing one non-zero activation (event) at a time. Parallelism is exploited in the kernel (two-dimensional weight matrix, each assigned to a channel of input activations) and filter (three-dimensional matrix consisting of multiple kernels) levels in convolutional layers and the output neuron level in fully connected layers. Starting from the first event, all the corresponding weights are retrieved and sums are computed to update relevant output neurons. After processing the first activation, the activation can be discarded. Then the second non-zero activation is fetched and processed in a similar way. This mechanism reuses activations to a great extent because all computations requiring the activation are done at once and each activation only needs to be read once from the memory to the computing unit. Following the multiply phase, the right of Fig. 3 shows the fire phase. The output feature maps are traversed to find values that exceed the ReLU threshold (value zero). All values greater than the threshold will be fired to the next layer. Since zero-valued activations from the output feature map are discarded at the fire phase, they are never stored and accessed in the subsequent computation. Note that Fig. 3 only provides one example. The fire phase can perform more than the ReLU function, which is discussed in the following subsection. Together, the multiply and fire phases allow dynamic and irregular activation sparsity to be exploited naturally without any extra logic and enable all ineffectual computations to be skipped.

3.1 Multiply Phase

We consider two types of input events to support the acceleration of convolutional layers and fully connected layers.

3.1.1 Convolutional layers. Each non-zero activation (i.e., IFM[I_ch][I_i][I_j] in Fig. 4) is processed sequentially by the PE array as represented by the first 5 lines in the algorithm. I_c, I_w, and I_h represent the number of input channels, width, and height of input feature maps, respectively. In this layer, both kernel and filter-wise parallelism are exploited. We define the kernel as the 2-dimensional matrix with width and height and filter as a 3-dimensional matrix consisting of width, height, and channel size. Kernel-wise parallelism means that MAC computations of a single activation on a single kernel are processed in parallel. Filter-wise parallelism means that MAC computations of a single activation on multiple filters are processed in parallel. In the PE array, to exploit filter-wise parallelism between PEs, each PE handles \( \frac{O_c}{pe_{total}} \) output feature maps, where \( O_c \) is the total number of output feature maps and \( pe_{total} \) is the total number of PEs. Since the number of total output feature maps can be more than PEs, each PE is designed with the capability to exploit filter-wise parallelism. In addition, to exploit kernel-wise parallelism within the PE, all corresponding weights from a single kernel required to be multiplied with the activation are read and the results are updated to relevant output neurons in parallel. As each PE is uniquely mapped to specific filters and generates the corresponding output feature maps, weights are highly reused within the PE and partial sums are immediately accumulated in the PE.

As an example, consider a convolutional layer with an IFM size of 4×4×3 and four 3×3×3 weight filters shown in Fig. 4. Assuming that the kernel’s stride is 1 along the x- and y-axes and the activation value in the IFM at coordinates IFM[0][1][1] is non-zero, such as 100, as shown in the top half of Fig. 4, each PE is mapped with a different filter and processes different output feature maps in parallel. When a processing element (PE) in event-driven dataflow receives one event, it performs the steps outlined in the following algorithm (assuming padding is 0 and input/output channel is 1 and indexed with 0):

ACM Trans. Arch. Code Optim.
This work does not suffer from unnecessary data access and computation as seen in prior works [21] (see Fig. 2 for more details). Only non-zero activations are accessed and processed. All non-zero activations are processed sequentially without any re-access from off-chip and on-chip local memory.

(i) Read \( w[0][0][1][1] \) and multiply it with activation \( IFM[0][1][1][1] \). Then add the result with the previous result of \( OFM[0][0][0] \) to update \( OFM[0][0][0] \) as shown in Fig. 4 MAC0. (This operation can be represented as \( OFM[0][0][0] := w[0][0][1][1] * IFM[0][1][1][1] \))

(ii) Shift right the kernel and perform \( OFM[0][0][1][1] := w[0][0][0][0] * IFM[0][1][1][1] \) as shown in Fig. 4 MAC1.

(iii) Shift down the kernel and perform \( OFM[0][1][0][1] := w[0][0][0][1] * IFM[0][1][1][1] \) as shown in Fig. 4 MAC2.

(iv) Shifts right the kernel and perform \( OFM[0][1][1][1] := w[0][0][0][0] * IFM[0][1][1][1] \) as shown in Fig. 4 MAC3.

(v) All of the above operations can be handled by separate MAC units to allow for kernel-wise parallelism.

In this way, event-driven dataflow fully utilizes a single input activation event, performs all necessary multiplication operations, and updates all the required output neuron values in a convolutional layer’s OFM. As we aim to demonstrate in our hardware design later, multiplication operations for a single input event can be processed at once in parallel in the hardware.

In terms of data storage format, this work compacts activation information with activation value together and follows a specific format: \text{input\_value} + \text{channel\_id} + \text{neuron\_address}. The \text{channel\_id} indicates the corresponding channel number to which the input activation belongs. The \text{neuron\_address} indicates the first OFM neuron address that is required to be updated. The format is similar to COO and requires a similar storage. It allows both activation and its coordinate information to be read in one cycle instead of the two cycles required in

ACM Trans. Arch. Code Optim.
Filter-wise** parallelism

Kernel-wise** parallelism

(a) Event-driven dataflow on convolutional layers

(b) Algorithm for processing input events in convolutional layers. The if statement is only present for clarity and does not represent how this work skips the zero-valued activations.

CSC/CSR, which is more hardware-friendly. Since weights are structurally pruned, they can be stored and accessed in a dense format.

3.1.2 Fully connected layers. Although this work focuses on structured sparsity achieved by structured pruning, unstructured sparsity in fully connected is supported. The computation process is similar to the convolutional layer, each non-zero activation (i.e., \( I[I_i] \)) is processed sequentially by the PE array as represented by the first 3 lines in the algorithm. \( L_t \) represents the number of input neurons. Since there are no kernels and filters in a fully connected layer, the MAC units can be shared among all output feature maps in parallel.

ACM Trans. Arch. Code Optim.
Fig. 5. (a) The method used by event-driven dataflow to evaluate the outputs of neurons in fully connected layers. Given neuron 1 has a non-zero input value, the computation starts by broadcasting neuron 1 to all the PEs. In PE0, all relevant non-zero weights (light green) are multiplied by the input activation. All output neurons mapped to PE0 and connected with the non-zero weights (light blue) are updated. At the same time, PE1 multiplies relevant weights (deep green) with the same activation to update relevant output neurons (deep blue). In this process, output neuron level parallelism is exploited between PE and within each PE. After computations of neuron 1 are done, the process repeats to compute the next non-zero input activation. (b) Algorithm for processing input events in fully-connected layers. The if statement is only present for clarity and does not represent how this work skips the zero-valued activations and weights.

connected layer, we exploit output neuron-wise parallelism across the PE array and within the PE. In the PE array, each PE handles \((O_t / \text{pe\_total})\) output neurons, where \(O_t\) is the total number of output neurons and \(\text{pe\_total}\) is the total number of PEs. Inside each PE, multiple weights from each neuron are read to multiply with the input activation to update the relevant output neurons in parallel.

Take a two-layer fully connected network with unstructured sparsity as an example in which there are 5 and 10 neurons in the first and second layer, respectively. If neuron 1 in the first layer has a non-zero output neuron value such as 100, the input event delivered to the multiply module of the second layer comprises the following information: a) activation value of 100, and b) neuron address of 1. Suppose the 10 output neurons are assigned to 2 PEs. When PE1 receives the input event, it performs the steps outlined in Fig. 5b. It reads all 4 non-zero weights linked with neuron address 1, skips neurons linked with zero-valued weights, and multiplies non-zero weights by input activation (in this example, 100). The multiplication results will subsequently be added to the first four neurons (with non-zero weights) of the second layer. PE2 will simultaneously process the same activation value (i.e., 100) with the rest of the output neurons in the same way. In this way, event-driven dataflow performs all the required multiplication and updates the output neuron values in fully connected layers. For structured sparsity, the same process is done and may require less PE to process since neurons are pruned.

In terms of storage format, activations are stored similarly to the convolutional layer while weights follow a different format based on the pruning technique applied. In structured pruning, neurons of less importance are entirely removed from the layer, resulting in a smaller dense layer with only neurons of high importance. In this case, we store weights in a dense format without additional index information. In contrast, unstructured pruning randomly removes weights of low importance from each neuron, resulting in a sparse layer with the same number of neurons but fewer weights per neuron. We store the sparse weights in a compressed format similar to CSR but instead of using row and column pointers, we use a single neuron index to identify which output neurons each weight is connected to and minimize memory overhead.
3.2 Fire Phase

The fire phase can be viewed as a compression phase which is commonly not considered by the recent works [8, 12, 21]. This phase performs four tasks: (1) ReLU; (2) max/average pooling; (3) formatting and compacting the output activation (output neuron) value; and (4) ordering sequence of output activations. After completing the multiply phase, the output values in the convolutional or fully connected layer undergo a comparison with the ReLU threshold, typically set at zero. Output values below the threshold are filtered out by the fire module and are not transmitted to the next layer. On the other hand, if the value of the output neuron exceeds the threshold, it is either transmitted to the pooling module for further processing or directly formatted to input activation to the next layer. For convolutional layers followed with max/average pooling, the pooling operation is fused with ReLU and done on-the-fly. Each output value goes through ReLU first and is collected based on the size of the pooling operation. When all values from a pool are collected, the maximal (max-pooling) or average (average-pooling) value will be selected as the final value and then formatted and “fired” as input activation to the next layer. This means only non-zero output activations are fired to the next layer instead of all activations including zeros. This process naturally compressed the sparse output activations. In addition, the fire phase can control the firing sequence of activation values by traversing the OFMs in different orders. This represents a fundamental characteristic of event-driven dataflow. The ordering capability of the fire phase provides additional possibility to dataflow, which we consider as future work to improve energy efficiency. This is further discussed in Section 7.

The fire phase together with the multiply phase enables the accelerator to compute data in an event-driven manner and significantly reduces the number of computations in the network and the need to re-compress the activations after ReLU to the corresponding compression format, leading to lower inference latency and better energy efficiency.

4 MULTIPLY-AND-FIRE (MnF) : HARDWARE ARCHITECTURE

This section describes the hardware architecture of MnF to enable energy-efficient sparse DNN inference using the event-driven dataflow presented in Section 3.

Based on the studies and observations made by previous work [15, 18, 21], data movement, especially access to off-chip memory, consumes the highest amount of energy. While it may be necessary to employ off-chip memory in the design, modern pruning, and quantization techniques have made it possible for large networks such as AlexNet and VGG-16 to fit on chip. Therefore, the architecture of this work is designed with the goal of eliminating off-chip memory accesses and fitting pruned and quantized modern networks fully on chip. In cases where the on-chip memory is insufficient to store the entire model, an off-chip DRAM can be added as the main storage. Memory streaming and double-buffering techniques can be used to hide off-chip memory access latency and maintain high throughput.

4.1 Hardware Overview

The architecture consists of the OpenSMART network-on-chip (NoC) [29] with a mesh topology that connects multiple processing elements (PEs) and memory units through the X-Y routing mechanism. Each non-zero input activation will be sent as an event to the first PE (0,0) for processing and, in the meantime, forwarded to the second PE (0,1). The second PE will start the same process and forward the same activation to the third PE (0,2) and so on. The design exploits kernel-wise and filter-wise parallelism within a PE and filter-wise parallelism between PEs. Following our dataflow described in Section 3, all multiplication of a non-zero input activation can be processed in parallel. Activation sparsity can also be exploited without hardware overhead. As our design fully exploits the activation sparsity and only non-zero activation values are stored or accessed. We refer to non-zero value activation as activation in the following subsections. The design of the processing element is detailed in the
4.2 Processing Elements

Fig. 6 shows the PE architecture consisting of a router interface, core, memory interface, FIFOs, and two types of SRAM to store and process the data locally. The PE processes two types of events, input activation events and end-of-data events. With our dataflow, each input activation is only required to be accessed and sent from the memory unit to PE once. Input activations are received by the router interface in PE and sent to the core where the MAC operators reside. Partial sums from the computations are accumulated and stored locally in the accumulated SRAM. After all the computations are completed for a layer, the sums are sparsified by ReLU in the activation module and the final results are sent back to the memory unit through the router interface. End-of-data events indicate that all input activations have been processed and PEs are ready to send the results to the next layer.

4.2.1 Router interface. Input activations are received by the router interface, the gateway into the PE. It primarily performs three types of operations: (1) delivering the activation to the core for processing; (2) forwarding the activation to other PEs; and (3) transferring the output results from the core to the memory unit. In our design, operations one and two are performed at the same time to enable parallel processing of the activation.

4.2.2 Storage and Memory Interface. As seen from Fig. 6, each PE contains two types of SRAM: one for weight and one for accumulated SRAM. As we aim to eliminate off-chip memory accesses from our design during inference, neural network weights are spread across PEs and stored in the weight SRAM, whereas results of MAC operations are stored in the accumulated SRAM. In addition, both of the SRAMs are clock gated to improve energy efficiency.

4.2.3 Core. Fig. 6 depicts the design of the core. It comprises a load module, a dispatcher module, a MAC cluster, and an activation module. The architecture is built on a decoupled access/execute approach [44], which allows the modules to operate independently as long as the necessary data is available.
When the MAC module is stalled the load module can continue loading data to serve the dispatcher modules, and the dispatcher module can still generate new results to be processed by the MAC module. The decoupled access/execute micro-architecture results in increased compute and memory access overlap, which reduces computational delay. Instead of connecting the modules directly, circular FIFOs are used as an interface between them to enable this result.

**Load Module:** The load module is responsible for reading the data required to execute the multiply and fire phases introduced in Section 3.

The load module begins by decoding the input events received from the router interface. Based on the decoded channel index and start weight address, it generates the weight address for the weight SRAM and then sends a read request to the weight SRAM interface. In addition, it generates a list of output neuron addresses that the current input will update. The load module packs the addresses in the correct order and sends them with the input value to the dispatcher module.

When an end-of-data event is received after all input data have been received, the load module does not compute any additional addresses or send any new read requests; it only forwards the event to the dispatcher module. In both operations, the load module determines the type of input received (actual input event or end-of-data event), decodes the event, and forwards the information to the dispatcher module.

**Dispatcher Module:** The dispatcher module has two main tasks. First, it groups an input activation value with its corresponding output neuron address and weight retrieved from the weight SRAM. In our design, the number of weights retrieved in a single read equals the number of multipliers used. This allows multiplication between weights retrieved and inputs to occur in parallel in a single cycle. Second, based on the output neuron address, it sends the group of data to the respective MAC module to be processed. When the end-of-data event is received, the dispatcher simply forwards the information to all MAC modules.

**MAC Cluster:** The MAC cluster is where the multiply phase happens and is responsible for executing the multiply and accumulation computation of 8-bit fixed point input activations and weights. In the cluster, there are multiple MAC modules and each MAC module is designed with multiple multipliers and accumulators to perform multiple computations in parallel. It is also connected with the accumulated SRAM directly through the memory interface module, as seen in Figure 6, to retrieve and store the 25-bit partial sum (through our hardware simulation studies, the partial sum can be reduced to 25-bit without degrading the accuracy).

The MAC module receives two types of data from the dispatcher module: (1) input activation value and the corresponding weight and output neuron address, it performs the MAC computations and stores the newly accumulated partial sum back to the local SRAM. Following our dataflow, partial sums are immediately accumulated in the designated output neurons in local SRAM. Therefore, our architecture does not produce a large number of partial sums and does not cause stalls and data transfer congestion to write back these sums; (2) End-of-data event, it quantizes the accumulated partial sums from SRAM to 8-bit as proposed in [25] and transfers the quantized sum to the activation module.

**Activation Module:** The activation module is where the fire phase happens and is responsible for performing max-pooling and ReLU operations. To perform max-pooling, a bit mask is applied to retrieve relevant results from the MAC modules. Then, the max value is calculated by comparing the retrieved results. In the ReLU operation, only output results that are larger than the ReLU threshold (quantized value of float 0) are transferred to the router interface. Results smaller than the threshold will be discarded.

5 EVALUATION

5.1 Experimental Methodology

We use Verilog to implement our accelerator and obtained cycle counts through behavioral simulation. The RTL model is synthesized using Synopsys Design Compiler version P-2019.03, targeting the 22nm technology
Table 1. Processing element specifications

|                | MAC Cluster Size | Multiplier per PE | Weight SRAM per PE | Acc SRAM per MAC Cluster | Frequency (MHz) | Bit Precision |
|----------------|------------------|-------------------|-------------------|--------------------------|----------------|--------------|
| MnF-D          | 9                | 27                | 10.1 KB           | 4.69 KB                  | 200            | Weight/Activation : 8 bits |
| MnF-S          | 9                | 27                | 648 KB            | 51.6 KB                  | 200            | Psum : 25 bits             |

MnF-S is a fully on-chip design, based on an 11 processing element design, the SRAMs are sized to store the entire pruned model weights and per-layer accumulated sum on the on-chip local memory.

to a fair comparison with the prior works that require off-chip memory accesses, we evaluated two versions of the hardware design. MnF-S represents our target design with only on-chip local memory access and MnF-D represents the implementation with off-chip memory. For MnF-D, we carefully sized the on-chip memory so that with our MnF technique, activation, and weight data will only be fetched once from the off-chip memory. Table 1 contains the full hardware specifications. To compare the proposed work with the prior works, we begin by closely comparing the performance between our design and two existing structured sparsity-aware DNN accelerators, namely Cambricon-S and CSP. Subsequently, we extend the comparison to include other sparsity-aware DNN accelerators.

For comparison with the two structured sparsity-aware DNN accelerators, we select the widely used networks which are evaluated by CSP and Cambricon-S as well. They are AlexNet [27], VGG-16 [43], ResNet-50 [22] and Transformer [49]. Structured pruning of ImageNet [41] trained AlexNet and VGG-16 is done with the ThiNet [34] technique. In the following, we refer to sparsity as the number of zero-valued data. Structured pruning of CIFAR-10 trained VGG-16 and ResNet-50 is done with Network Slimming [31]. On the CIFAR-10 [26] dataset, VGG-16 is pruned to a 70% weight sparsity and ResNet-50 has 78% weight sparsity. On ImageNet data [13], pruned VGG-16 and AlexNet achieved 36.8% and 50% filter sparsity, respectively. This translates to 50.8% weight sparsity in VGG-16 and 81.9% in AlexNet. The accuracy of all the pruned models is maintained at a similar level as respective dense models with less than a 1% drop in accuracy. It should be noted that both pruning techniques prune the entire filter of less importance. At the point of evaluating this work, a ready-to-apply pruning technique was not found to prune the original Transformer model [49] hence, we apply a synthetic sparsity of 85% (based on the pruning ratios provided by previous work [21]). As the ReLU function is only applied in the first layer of feed-forward layers, we assume input activation to be fully dense in the evaluation of Transformer.

For evaluation with other sparsity-aware DNN accelerators, we compared AlexNet, VGG16 and MobileNet 1 [24] since those accelerators only reported performance on the mentioned networks. Pruned AlexNet and VGG-16 are the same ImageNet models used in comparison with CSP and Cambricon-S. Similar to Transformer, we assume a synthetic sparsity of 70% on MobileNet based on the overall DRAM accesses of the model by GoSPA [12].

To analyze the effectiveness of our dataflow and architecture on convolution and fully connected layers separately, we target only the convolution layers in CNN models and fully connected layers in Transformer.

1The MobileNet evaluated has a width multiplier of 0.5 and an input size of 128×128, the same workload as Eyeriss v2 and GoSPA.
5.2 Comparison on Data Access Energy
We evaluate the off-chip and on-chip data access energy of MnF-D and show lower overall access energy when compared with Eyeriss v2 [8] and CSP [21]. The access energy is computed by multiplying the number of accesses with the access energy per byte. To make a fair comparison, we assume the same access energy per byte for each evaluated model. The details of the evaluation are elaborated in the following subsections.

5.2.1 Comparison with Eyeriss v2. To simulate the data access pattern of Eyeriss v2, we use Sparseloop [51] which has a ready-to-run Eyeriss v2 setup to evaluate some of the point-wise convolutional layers in MobileNet [24]. We use the results of those layers from MnF-D and compared them with Eyeriss v2. Since Sparseloop does not provide off-chip data access energy, we only compare the on-chip accessing energy. To allow for a fair comparison, we calculate the energy consumption of this work by multiplying the energy per access proposed by Sparseloop (0.13 pJ for activations, 0.33 pJ for partial sums, and 0.5 pJ for weight) and the number of access generated by our dataflow. As shown in Fig. 7, the energy consumed by MnF-D in accessing activations, weights, and partial sums (psums) is all less than Eyeriss v2. Specifically, our event-driven dataflow only accesses activation once, the energy of accessing activation is $452 \times$ less than Eyeriss v2. Overall, when running these point-wise convolutional layers
Table 2. Layer-wise utilization and efficiency of AlexNet on ImageNet, running at 200MHz with 297 multipliers

| AlexNet Layer | Weight Sparsity | Input Act. Sparsity | Multiplier Utilization | Efficiency (Frames/J) |
|---------------|-----------------|---------------------|------------------------|-----------------------|
| Conv 1        | 56.3%           | 1.2%                | 93.3%                  | 3826.4                |
| Conv 2        | 81.1%           | 32.2%               | 91.5%                  | 7093.8                |
| Conv 3        | 85.9%           | 47.3%               | 94.7%                  | 17449.4               |
| Conv 4        | 89.4%           | 72.7%               | 98.8%                  | 25044.9               |
| Conv 5        | 67.6%           | 59.0%               | 97.0%                  | 11265.6               |
| Average       | 76.1%           | 52.8%*              | 95.1%                  | -                     |
| Total         | 81.9%           | 52.9%*              | -                      | 1699.8                |

*Computation excludes the first convolution layer where input is almost dense.

In MobileNet, this work can save $11.3 \times$ on-chip access energy compared with Eyeriss v2. For off-chip access, this work requires a total of 1.05 MB access, which is $3.7 \times$ smaller than the off-chip access mentioned in the Eyeriss v2 paper (3.9 MB).

5.2.2 Comparison with CSP. To compare data access energy, we estimate the number of accesses from the description of CSP’s dataflow. Based on the number of accesses when running one inference of ImageNet on VGG-16, we calculate the energy consumption of both CSP and this work by multiplying the number of accesses and the energy per access proposed by CSP (0.84 pJ for on-chip activations access, 1.76 pJ for on-chip weights, 2.83pJ for on-chip psums, 766pJ off-chip read, and 780 pJ off-chip write). As shown in Fig. 7b, the off-chip accessing energy of activation and psums of this work is $5.93 \times$ and $8.36 \times$ less than CSP, respectively. The main reason is that our event-driven dataflow naturally filters out zero-valued activations. Therefore, this work only requires reading and writing non-zero values from off-chip memory. Meanwhile, for on-chip activation access, this work is $14.0 \times$ better than CSP since this work fully reuses the accessed activation and, unlike CSP, we do not need to re-access the same activation. In Fig. 7c, we show an overall $26.9 \times$ less number of activation accesses than CSP. However, the total number of weight accesses is more than that of CSP. One reason is that the sparsity of our VGG-16 is less than that of CSP. This inherently results in more weight fetches. Another reason is our dataflow, as we aim to maximize activation reuse, we trade off weight reuse. Therefore, the energy of weight accesses of this work is $1.69 \times$ more than CSP on off-chip access and $46.25 \times$ more than CSP on on-chip access. Despite generating a higher number of on-chip weight accesses compared to CSP, it is worth noting that these on-chip weight accesses only constitute 15% of the total accesses. As a result, even with lower sparsity, we manage to significantly reduce the number of total accesses and achieve access energy that is $2.06 \times$ lower than CSP.

In addition, we evaluate processing element (PE) utilization. We define PE utilization as (#Active PEs) / (#PEs) and an active PE as one that is processing a non-zero activation (effectual computations). Since CSP cannot handle activation sparsity, the PEs have to process unnecessary zero operand multiplications (ineffectual computations) which ineffectively occupy PE resources. As shown in the secondary axis of Fig. 7c, the utilization rate remains stable (>90%) for MnF, but varies significantly across layers for CSP (which sees values from 15% to 70%). Disregarding activation sparsity can lead to the underutilization of PEs, which in turn can have a detrimental effect on the overall performance of the system.

5.3 Sensitivity to Sparsity

A sparsity-aware accelerator should be able to efficiently process models with different sparsity. We study how activation sparsity can affect the latency and efficiency of MnF. From Fig. 8a, it is shown that the access, latency, and energy efficiency improvement scales with activation sparsity. With $10 \times$ reduction (90% sparsity) in activation, the total number of accesses is reduced by 10 times, and both latency and energy efficiency are improved by
Multiply-and-Fire (MnF): An Event-driven Sparse Neural Network Accelerator

Fig. 8. (a) Access reduction, latency, and energy efficiency improvement over dense workload with increasing activation sparsity on convolution layer 2 of AlexNet. In the evaluation, the dense workload is used as the baseline. Only the input activation sparsity is changed while all other design choices that can affect performance are kept the same (e.g. weight sparsity and number of processing elements. (b) Multiplier utilization rate across different weights and activation densities on ResNet-50, MobileNet, and GoogleNet. ResNet-50 has a large number of repeating blocks of $1 \times 1$, $3 \times 3$, and $1 \times 1$ kernels with different channel sizes, in the utilization rate evaluation only these blocks are evaluated. Similarly, in GoogleNet, all the inception blocks are evaluated.

around 9 times. The proposed event-driven dataflow processes non-zero activations sequentially and focuses on maximizing the parallelism of a single activation at a time. This allows our hardware to be insusceptible to irregular activation sparsity. The same evaluation is done on weight sparsity and a similar result is observed.

Low utilization of multipliers caused by the imbalance workload of sparse DNNs is a common problem faced by sparsity-aware accelerators. Based on the mapping of the convolution layer on MnF described in Section 3, the utilization of the multiplier, seen as the main computation resource, depends solely on the number of output channels mapped to a processing element. As shown in Fig. 8b, MnF can achieve a high multiplier utilization rate across workloads from dense to 90% synthetic sparse. The multiplier utilization rate is computed based on active processing elements (PEs) that are involved in the computation process. PEs that are not activated are power-gated and not accounted for in the evaluation. The evaluation is performed with the hardware specification shown in Fig. 1. On ResNet-50 and MobileNet, MnF is able to maintain a high utilization rate of above 90% across dense to 80% sparse workloads and is able to achieve a utilization rate of above 80% in extremely sparse workloads. On GoogleNet [47], the utilization rate is lower as our event-driven dataflow exploits filter-wise parallelism and the number of multipliers per PE (27) becomes more than the number of computations required with increasing sparsity in some of the point-wise layers. However, MnF is still able to maintain high utilization of 85% and above on GoogleNet across most of the sparsity levels.

The layer-wise sparsity can vary across layers in the same network. In Table 2, the layer-wise sparsity breakdown of AlexNet and the corresponding multiplier utilization and energy efficiency are shown. With our activation-centric dataflow and architecture design, MnF is able to maintain a high multiplier utilization rate (>90%) across layers of different sparsity. MnF is also able to utilize the yield of the high sparsity as with higher sparsity, MnF reaches a higher efficiency.
5.4 Comparison with Structured Sparsity-Aware DNN Accelerators

Cambricon-S and CSP are two of the state-of-the-art sparsity-aware DNN accelerators that are designed to handle DNN models with structured sparsity efficiently. As both works do not provide the actual performance numbers but the normalized results against the baseline accelerator, DianNao [5]. To compare against the accelerators, we first use Timeloop[37] to simulate DianNao and obtain the latency results on target networks. The energy results from DianNao are estimated based on the method described in CSP: the total inference energy is a sum of off-chip access memory energy, on-chip memory energy, and MAC computation energy. Off-chip memory energy is computed by multiplying the number of accesses by the per-byte energy cost of reads (766pJ/read) or writes (780pJ/write). On-chip memory energy is also computed by multiplying the number of accesses by the unit-energy cost of reads (1.15pJ/read) or writes (2.98pJ/write). MAC energy is computed by multiplying the number of MACs by a conservatively estimated dynamic power of 0.081pJ. After getting the performance results, we then normalize our results to DianNao and compared them directly with the normalized results reported by CSP. As CSP normalized and scaled the results of CambriconS in its evaluation, the results of CambriionS are taken from CSP.

We would like to note that performance results from both Cambricon-S and CSP are based on 1024 multipliers, while we only scaled our accelerator to 972 multipliers. We did not scale to 1024 since each PE of our design is implemented with 27 multipliers which is not a factor of 1024 and 972 multipliers are sufficient to achieve comparable latency performance. For MnF-D design, we used the same unit energy cost of reads and writes as described in CSP to estimate the off-chip access energy and assume a data rate of 3200MT/s. Fig. 9 shows both energy efficiency and speedup of all the accelerators normalized to the results of DianNao. Energy efficiency is the energy consumed per inference and the speedup is the latency of running a single inference. For a fair comparison with the accelerators, we scale the energy of our accelerator to 65nm technology node with the factor provided in recent work [42]. We compare speedup in terms of cycles per inference since MnF is running at 200MHz and CSP and CambriconS are running at 300MHz.

5.4.1 Energy Efficiency. As shown in Fig. 9, on all of the evaluated models except for AlexNet, our accelerator shows significant energy improvement over Cambricon-S and CSP. Overall, MnF-S is 40.7× and 11.2× more energy efficient than Cambricon-S and CSP, respectively. MnF-D is 7.8× and 2.2× more energy efficient than Cambricon-S and CSP, respectively. The efficiency improvement mainly comes from (1) the ability of MnF in handling activation sparsity efficiently. Unlike CSP and Cambricon-S, the unnecessary data storage, access, and
ineffectual MAC computations are entirely eliminated from this work. (2) Our event-driven dataflow enables maximum reuse of activation, thus it requires just a single activation access. In contrast, CSP only improves activation reuse and still requires multiple on-chip memory activation access. Even in Transformer where we assume no sparsity in the activation, MnF requires a lower number of memory access and achieves $4.8 \times$ higher energy efficiency than CSP. All these are translated to energy savings and allow MnF to achieve higher energy efficiency over Cambricon-S and CSP.

However, as shown in Fig. 9, MnF does not show a significant improvement in energy efficiency on AlexNet and this is due to the hardware design. As our PE is designed with 27 multipliers to work with the $3 \times 3$ kernel size, handling the $11 \times 11$ and $5 \times 5$ kernels presented in the first two layers of AlexNet becomes less efficient. In spite of that, MnF-S is still $1.41 \times$ better and MnF-D is $30\%$ less efficient than CSP. The limitation in supporting various kernel sizes is discussed in Section 7.

5.4.2 Speedup. In terms of speedup, MnF achieves comparable or better performance on most of the network models even with fewer multipliers. Excluding Transformer, MnF-S is overall $2.19 \times$ and $1.41 \times$ better than Cambricon-S and CSP, respectively. Without the limitation of NoC data width (currently, set to one output result per transfer per processing element) to transfer data between the processing elements (PEs) and memory units, MnF-D achieves a $2.38 \times$ and $1.53 \times$ faster speed than Cambricon-S and CSP, respectively. Our accelerator achieves this speedup by avoiding accessing zero-valued activations, therefore, skips the ineffectual multiplications with zero. Furthermore, our dataflow is able to process all computations of a single activation in one cycle and process multiple activations in parallel. On Transformer, MnF-D is slightly slower and achieves $86\%$ the speed of CSP. One reason is that the activation of the Transformer is generally dense and we are assuming a fully dense activation input. With fewer multipliers used and running a dense workload, we are bound by computing resources. In addition, our activation module can only access one result each from the 9 MAC clusters (designed to efficiently support convolutional layers), this limits parallelism in transferring the dense results out of the processing PE. To achieve higher throughput, the number of multipliers and the transferring parallelism can be increased. Although we are $1.16 \times$ slower in running Transformer, we are at least $4.8 \times$ more energy efficient than CSP.

5.5 Comparison with Other Sparsity-Aware DNN Accelerators

In the comparison with other sparsity-aware DNN Accelerators, the power of all the accelerators is scaled to the 28nm technology node for a fair comparison. We synthesize our approach with the same setup as described in Table 1, running at 200 MHz operating frequency based on the 8-bit neural network architecture. To match the peak computation speed (number of MAC × frequency) to most of the evaluated works, we use 11 processing elements designed with 297 multipliers. We assume a data rate of 3200MT/s for MnF-D and a 168pJ per-byte off-chip data access energy (computed from GoSPA) for MnF-D and Eyeriss series. The results are shown in Table 3.

Although MnF hardware is designed and optimized for a fully on-chip architecture, MnF-D shows comparable and superior results to previous works. Overall, MnF-D is $17.4 \times$, $2.28 \times$, $2.35 \times$, $2.53 \times$ and $1.12 \times$ more energy efficient than Eyeriss, PermCNN, Eyeriss v2, NullHop, and GoSPA, respectively. Our targeted energy-saving design, MnF-S achieves $28.2 \times$, $3.70 \times$, $4.81 \times$, $4.21 \times$ and $2.14 \times$ better energy efficiency than Eyeriss, PermCNN, Eyeriss v2, NullHop and GoSPA, respectively. In terms of speedup, compared to the Eyeriss series, SPOTS, NullHop, and the state-of-the-art design GoSPA, MnF (considering both MnF-S and MnF-D) is at least $1.39 \times$ faster on all the evaluated networks except for AlexNet. From the results, MnF achieves a small improvement in both the energy efficiency and speedup of AlexNet and this is due to the limitation in supporting various kernel sizes elaborated in Section 7. Overall, MnF has comparable or superior performance than other sparsity-aware DNN ASIC designs as we aim to eliminate the need for high-overhead pairing logic, skip all ineffectual computations and achieve lower memory access energy with our event-driven dataflow.
Table 3. Performance comparisons among different sparsity-aware DNN ASIC designs (Power and Frames/J are scaled to 28nm)

| Design       | Eyeriss [7] | PermCNN [11] | SPOTS [45] | Eyeriss v2 [8] | NullHop [1] | GoSPA [12] | MnF-S | MnF-D |
|--------------|-------------|--------------|------------|----------------|-------------|------------|-------|-------|
| Sparsity     | Unstructured | Structured   | Structured | Unstructured   | Unstructured | Unstructured | Structured | Structured |
| Bit Width    | 16          | 16           | 16         | 8              | 16          | 8          | 8     | 8     |
| Num. of MACs | 168         | 128          | 512        | 384            | 128         | 128        | 297   | 297   |
| Frequency (MHz) | 200       | 800          | 500        | 200            | 500         | 500        | 200   | 200   |
| Power (mW)   |             |              |            |                |             |             |       |       |
| AlexNet      | 205.9       | 654          | -          | 406            | -           | 290        | 181.2 | 299.9 |
| VGG-16       | 153.2       | 689          | -          | -              | 257         | 277        | 183.8 | 313.7 |
| MobileNet    | -           | -            | -          | 861.1          | -           | 418        | 250   | 675.1 |
| AlexNet      | 34.7        | 456.6        | 249.8      | 342.4          | -           | 460.3      | 472.6 | 473   |
| VGG-16       | 0.7         | 43.3         | 15.2       | -              | 13.7        | 29.7       | 41.5  | 42.2  |
| MobileNet    | -           | -            | -          | 1470.6         | -           | 1868       | 2893.5| 3179.7|
| Frames/S     |             |              |            |                |             |             |       |       |
| AlexNet      | 168.5       | 698.2        | -          | 843.3          | -           | 1587       | 2682.2| 1699.8|
| VGG-16       | 4.5         | 62.9         | -          | -              | 53.3        | 107.3      | 224.6 | 134.7 |
| MobileNet    | -           | -            | -          | 1707.8         | -           | 4473       | 12425.8| 4677.9|
| Frames/J     |             |              |            |                |             |             |       |       |
| AlexNet      | 168.5       | 698.2        | -          | 843.3          | -           | 1587       | 2682.2| 1699.8|
| VGG-16       | 4.5         | 62.9         | -          | -              | 53.3        | 107.3      | 224.6 | 134.7 |
| MobileNet    | -           | -            | -          | 1707.8         | -           | 4473       | 12425.8| 4677.9|

5.6 Energy Efficiency Breakdown

![Fig. 10. Comparison on energy efficiency, denoted in frames/J, between various versions of MnF and Eyeriss v2 on AlexNet. The MnF-D with "dense no parallelism" represents the version without kernel and filter wise parallelism running a dense workload without sparsity.](image)

We evaluate the factors influencing the energy efficiency of MnF, including kernel and filter-wise parallelism, weight and activation sparsity, and off-chip access. As shown in Fig. 10, by leveraging the kernel and filter-wise parallelism, MnF-D achieves 6.81x better energy efficiency than the implementation without parallelism. Since MnF targets sparse models, considering both activation and weight sparsity, MnF-D achieves 14.16x better energy efficiency than MnF-D with dense activation and the non-pruned model. Moreover, to further improve energy efficiency, we introduced MnF-S, the fully on-chip design, which improves 1.58x energy efficiency than the off-chip version and outperforms Eyeriss v2 by 3.18x.

ACM Trans. Arch. Code Optim.
5.7 Power and Area Breakdown

The power consumption breakdown of one processing element (PE) is shown in Fig. 11a. The per-PE power consumption is the same for both MnF-D (DRAM design) and MnF-S (fully on-chip) designs as we use clock-gating and multiple banks of memory, allowing only one small bank of memory to be accessed each time. From the figure, on-chip memory, FIFOs, and buffer consume 73.5% of the power and the core control and computation logic only consumes 22.8% of the total power.

The area results are based on the 22nm technology node. The total area of one PE without the on-chip memory (SRAMs) is 0.117 mm$^2$. For our MnF-D design, the total area of one PE including on-chip SRAMs is 0.44 mm$^2$. Based on the area breakdown shown in Fig. 11b, we can clearly see that the on-chip memory consumes 73.5% of the total area in the MnF-D PE design while all the control and computation logic only consumes a total of 19.8% of the area. Our targeted energy-saving design, MnF-S, which trades off area for power and energy efficiency has a total area of 4.02 mm$^2$ per PE. Unlike prior works such as SparTen and Eyeriss v2 which require specialized hardware logic and incur high power and area overhead in handling sparsity (62.7% and 46% of the total area and power in SparTen), MnF minimizes the area and power overheads and shows that the total control and computation logic only consumes around 20% of the PE’s area and power.

6 RELATED WORK

Multiple accelerators [1, 2, 4, 7–9, 17, 18, 21, 23, 32, 38, 40, 45, 53–59] are designed to leverage the benefits of pruned DNNs to achieve high energy efficiency and throughput. EIE [18] and Cnvlutin [2] are early efforts that exploit unstructured sparsity to enhance DNN throughput and energy efficiency. EIE exploits both irregular activation and weight sparsity, but it only supports the fully connected layers. Although Cnvlutin supports convolutional (Conv) layers, it only exploits irregular activation sparsity and does computation with zero-valued weights.

SCNN [38] is the first sparsity-aware accelerator that takes advantage of unstructured sparsity in both activations and weights. Although it is able to avoid all zero-valued computations, the input-stationary dataflow together with the Cartesian product technique introduces non-zero-valued multiplications that do not exist in the original convolution algorithm. Furthermore, the dataflow employed produces a large number of irregularly distributed partial sums which requires large accumulator buffer and complex hardware logic to update the results. Since the partial sums can update the same output, memory contention is possible which can lead to compute stalls and
a low multiplier utilization rate. STICKER [55] and [50] follow a similar dataflow as SCNN and try to reduce the memory overhead. Instead of the large multi-banked buffer used by SCNN, STICKER and [50] employ a set-associative processing element (PE) design to update the irregular partial sums. Although this significantly reduced storage area, memory contention still remains when updating the partial sums. Furthermore, a contention in a set of PEs can stall the whole computation process and lowers the multiplier utilization rate. This is because the design process data in batches and the PE array can only start to process the next batch when all operations are done in the current batch. SparTen [17] and GoSPA [12] are recent works that use an intersection-based method to exploit both the unstructured weight and activation sparsity and perform only the necessary computations. SparTen has demonstrated better energy efficiency and latency than SCNN, however, it uses an expensive operation, prefix sum, to identify the non-zero pairs of weight and activation. Unlike SparTen, GoSPA [12] took an on-the-fly intersection approach and employs a dedicated activation processing unit (APU) to identify the valid non-zero pairs of weight and activation, resulting in smaller energy overheads than SparTen.

In contrast to the aforementioned works, Cambricon-S [59], CSP [21], SPOTS [45] and S2TA [32] seek to exploit structured sparsity to accelerate the sparse DNNs. Cambricon-S and CSP use a hardware-software co-design approach that regularizes the weight sparsity but they cannot handle the irregular activation sparsity efficiently. SPOTS is a systolic-based design, it can skip ineffectual multiplications when both the input and weight blocks contain all zeros but is unable to do so when the block contains both non-zero and zero values. S2TA goes a step further, it exploits structured sparsity not only in the weight but also in the activation. To enable the structured activation sparsity, S2TA implements a specialized dynamic activation pruning technique and time-unrolled hardware architecture. Similarly, we leverage the structured pruning technique to regularize weight sparsity in this work, but we employ a straightforward Multiply-and-Fire (MnF) technique that naturally exploits unstructured sparsity in activations and skips all ineffectual computations. This technique eliminates the need for complex hardware overhead and achieves highly efficient event-driven computations.

7 DISCUSSIONS AND LIMITATIONS

MnF achieves better performance and energy efficiency than prior works but in the current hardware design, there is limited support for different kernel sizes. The processing element (PE) works best with kernel width that is multiple of 3 and pointwise 1x1 kernel. It is able to handle other kernel sizes like 11x11 kernels but the utilization rate can be lower and energy becomes less efficient as seen in Section 5. However, the 3x3 and 1x1 kernel sizes are commonly used and constitute a large number of layers in many recent CNNs [22, 24, 27, 43, 47] including the MobileNet and Resnet-50 evaluated in this work. Therefore, we did not extend the support for this work. Future work can look into supporting various kernel sizes to optimize performance and energy efficiency.

As seen from Fig. 7b, MnF has a large number of on-chip memory weight access. This can be optimized in future work by caching the 2-dimensional kernel in the MAC unit and leveraging the fire phase of the proposed dataflow to order and enforce the processing sequence of input activations. Since activation from an input channel will only be multiplied by the kernel from the same channel, by processing all activations from the same channel first, the corresponding kernel can be maximally reused.

In addition, the current design only exploits kernel and filter-wise parallelism in the PE. This means parallelism is limited to the same number of activations required by a single actialation. In future work, we can look into processing multiple activations in the same PE to exploit parallelism between activations and further improve the latency.

8 CONCLUSION

This work proposes MnF which presents a novel event-driven dataflow and an energy-efficient hardware accelerator for sparse DNN inference workloads. Our event-driven dataflow exploits activation sparsity naturally and skips all
ineffectual multiplications without the need for complex and high-overhead logic. It takes an activation-centric approach that aims to maximize the reuse of activation data, resulting in low overall energy consumption. Our accelerator leverages the dataflow and enables both kernel-wise and filter-wise parallelism with a high multiplier utilization, achieving high energy efficiency and performance. We demonstrate that MnF can achieve superior efficiency and performance than the state-of-the-art sparsity-aware DNN accelerators for most of the evaluated common DNN tasks where sparsity is now commonplace.

ACKNOWLEDGMENTS

We thank the reviewers who provided comments on the manuscripts that contributed significantly to the quality of this paper. This research is supported by A∗STAR under its RIE2020 IAFICP (Award I2001E0053). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the A∗STAR.

REFERENCES

[1] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Julia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbruck. 2019. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. IEEE Transactions on Neural Networks and Learning Systems (TNNLS) 30, 3 (2019), 644–656.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). 1–13.

[3] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured Pruning of Deep Convolutional Neural Networks. ACM Journal on Emerging Technologies in Computing Systems (JETC) 13, 3, Article 32 (2017).

[4] Kuo-Wei Chang and Tian-Sheuan Chang. 2019. VSCNN: Convolution Neural Network Accelerator with Vector Sparsity. In IEEE International Symposium on Circuits and Systems (ISCAS). 1–5.

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 269–284.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In 47th IEEE/ACM International Symposium on Microarchitecture (MICRO). 609–622.

[7] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. IEEE Journal of Solid-State Circuits (JSSC) 52, 1 (2017), 127–138.

[8] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9, 2 (2019), 292–308.

[9] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. IEEE Micro 41, 2 (2021), 29–35.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In 28th International Conference on Neural Information Processing Systems (NeurIPS), Vol. 2. 3123–3131.

[11] Chunhua Deng, Siyu Liao, and Bo Yuan. 2020. PermCNN: Energy-efficient Convolutional Neural Network Hardware Architecture with Permuted Diagonal Structure. IEEE Transactions on Computers (TC) 70, 2 (2020), 163–173.

[12] Chunhua Deng, Yang Sui, Siyu Liao, Xuehai Qian, and Bo Yuan. 2021. GoSPA: An Energy-efficient High-performance Globally Optimized SParse Convolutional Neural Network Accelerator. In ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). 1110–1123.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 248–255.

[14] Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. 2013. Predicting Parameters in Deep Learning. In 26th International Conference on Neural Information Processing Systems (NeurIPS). 2148–2156.

[15] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting Vision Processing Closer to the Sensor. In ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). 92–104.
[16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 580–587.

[17] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. In *52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 151–165.

[18] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *43rd International Symposium on Computer Architecture (ISCA)*, 243–254.

[19] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*, 1–14.

[20] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *28th International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 1, 1135–1143.

[21] Edward Hanson, Shiyu Li, Hai ‘Helen’ Li, and Yiran Chen. 2022. Cascading Structured Pruning: Enabling High Data Reuse for Sparse DNN Accelerators. In *49th Annual International Symposium on Computer Architecture (ISCA)*, 522–535.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

[23] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W. Fletcher. 2019. ExTensor: An Accelerator for Sparse Tensor Algebra. In *52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 319–333.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv* abs/1704.04861 (2017).

[25] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2704–2713.

[26] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report TR-2009. University of Toronto.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1106–1114.

[28] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. Inducing and Exploiting Activation Sparsity for Fast Inference on Deep Neural Networks. In *37th International Conference on Machine Learning (ICML)*, 5533–5543.

[29] Hyoukjun Kwon and Tushar Krishna. 2017. OpenSMART: Single-Cycle Multiple-Hop NoC Generator in BSV and Chisel. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 195–204.

[30] Yann LeCun, Y. Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (May 2015), 436–444.

[31] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *IEEE International Conference on Computer Vision (ICCV)*, 2736–2744.

[32] Zhi-Gang Liu, Paul N Whatmough, Yuhao Zhu, and Matthew Mattina. 2022. S2TA: Exploiting Structured Sparsity for Energy-efficient Mobile CNN Acceleration. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 573–586.

[33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully Convolutional Networks for Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440.

[34] Jian-Hao Luo, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* 33, 9 (2022), 4930–4944.

[35] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Huizi Li, Xuehai Qian, and Xue Lin. 2022. Non-Structured DNN Weight Pruning—Is It Beneficial in Any Platform? In *IEEE Conference on Neural Information Processing Systems (NeurIPS)*, 3045–3055.

[36] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and Trevor Darrell. 2017. SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks. In *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 27–40.

[37] Ngoc-Son Pham and Taeweon Sub. 2023. Optimization of Microarchitecture and Dataflow for Sparse Tensor CNN Acceleration. In *IEEE Access* 11 (2023), 108818–108832.
Multiply-and-Fire (MnF): An Event-driven Sparse Neural Network Accelerator

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115 (2015), 211–252.

[42] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5.

[43] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*. 1–14.

[44] James E. Smith. 1982. Decoupled Access/Execute Computer Architectures. In *9th Annual Symposium on Computer Architecture (ISCA)*. 112–119.

[45] Mohammadreza Soltaniyeh, Richard P. Martin, and Santosh Nagarakatte. 2022. An Accelerator for Sparse Convolutional Neural Networks Leveraging Systolic General Matrix-matrix Multiplication. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 3 (2022), 1–26.

[46] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9.

[48] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2818–2826.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017).

[50] Jingyu Wang, Zhe Yuan, Ruoyang Liu, Huazhong Yang, and Yongpan Liu. 2019. An N-Way Group Association Architecture and Sparse Data Group Association Load Balancing Algorithm for Sparse CNN Accelerators. In 24th Asia and South Pacific Design Automation Conference (ASP-DAC). 329–334.

[51] Ynan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2022. Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling. In 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). 1377–1395.

[52] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6071–6079.

[53] Weijie You and Chang Wu. 2019. An Efficient Accelerator for Sparse Convolutional Neural Networks. In *IEEE 13th International Conference on ASIC (ASICON)*. 1–4.

[54] Zhe Yuan, Yongpan Liu, Jinshan Yue, Yixiong Yang, Jingyu Wang, Xiaoyu Feng, Jian Zhao, Xueqin Li, and Huazhong Yang. 2020. STICKER: An Energy-Efficient Multi-Sparsity Compatible Accelerator for Convolutional Neural Networks in 65-nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)* 55, 2 (2020), 465–477.

[55] Jie-Fang Zhang, Ching-En Lee, Chester Liu, Yakuh Sophia Shao, Stephen W. Keckler, and Zhengya Zhang. 2021. SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference. *IEEE Journal of Solid-State Circuits (JSSC)* 56, 2 (2021), 636–647.

[56] Yue Zhang, Shuai Wang, and Yi Kang. 2023. MF-DSNN: An Energy-Efficient High-Performance Multiplication-Free Deep Spiking Neural Network Accelerator. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 1–4.

[57] Xiaodi Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. 2018. Cambricon-X: An Accelerator for Sparse Neural Networks. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 15–28.

ACM Trans. Arch. Code Optim.