CANDLES: Channel-Aware Novel Dataflow-Microarchitecture Co-Design for Low Energy Sparse Neural Network Acceleration

Sumanth Gudaparthi
University of Utah
Utah, USA
sgudapar@cs.utah.edu

Sarabjeet Singh
University of Utah
Utah, USA
sarab@cs.utah.edu

Surya Narayanan
University of Utah
Utah, USA
surya@cs.utah.edu

Rajeev Balasubramonian
University of Utah
Utah, USA
rajeev@cs.utah.edu

Visvesh Sathe
University of Washington
Seattle, Washington, USA
sathe@uw.edu

***Abstract*—Several deep neural network (DNN) accelerators have been designed to exploit the sparsity exhibited by DNN activations and weights. State-of-the-art sparse accelerators can be described as either Pixel-first or Channel-first accelerators, each with its unique dataflow and compression format aiding its dataflow. The former expends significant energy updating neuron partial sums, while the latter expends significant energy in handling the index metadata. This work introduces a novel microarchitecture and dataflow that reconciles these trade-offs by adopting a Pixel-first compression and Channel-first dataflow. The proposed microarchitecture has a simpler index-generation logic combined with an accumulator buffer hierarchy and crossbar with low wiring overhead. The compression format and dataflow promote high temporal locality in neuron updates, further lowering energy. Finally, we introduce work partitions across processing elements that naturally lead to load balance without offline analysis. Compared to four state-of-the-art baselines, the proposed architecture, CANDLES, significantly outperforms three and matches the performance of the fourth. In terms of energy, CANDLES is between 2.5× and 5.6× more energy-efficient than these four baselines.**

***Keywords*-Convolutional neural networks, Sparse tensors, Microarchitecture-Dataflow co-design, Hardware Accelerators**

I. Introduction

Several deep neural network (DNN) accelerators [39], [25], [3], [2], [46], [34], [10], [18], [11], [9], [49], [38], [12], [15], [31], [41] have been introduced in recent years, including several commercial implementations [24], [48], [17], [8], [32], [52], [43], [5], [1]. One of the most promising opportunities to improve the energy efficiency of these accelerators is the high level of sparsity exhibited by weights [40] and activations [4]. However, exploiting sparsity in both activations and weights, referred to as *two-sided sparse* [16], has resulted in architectures that are complex and/or under-utilized.

A second key opportunity is to identify a loop ordering, tiling, and partitioning (referred to as *dataflow*) that maximizes data reuse and minimizes data movement. While some prior works [54], [30], [7] have developed compiler methodologies to discover the ideal ordering, tiling, partitioning for generic dense accelerators and sparse accelerators with only sparse weights, similar tools for two-sided sparse accelerators do not yet exist. Not only are sparse accelerators

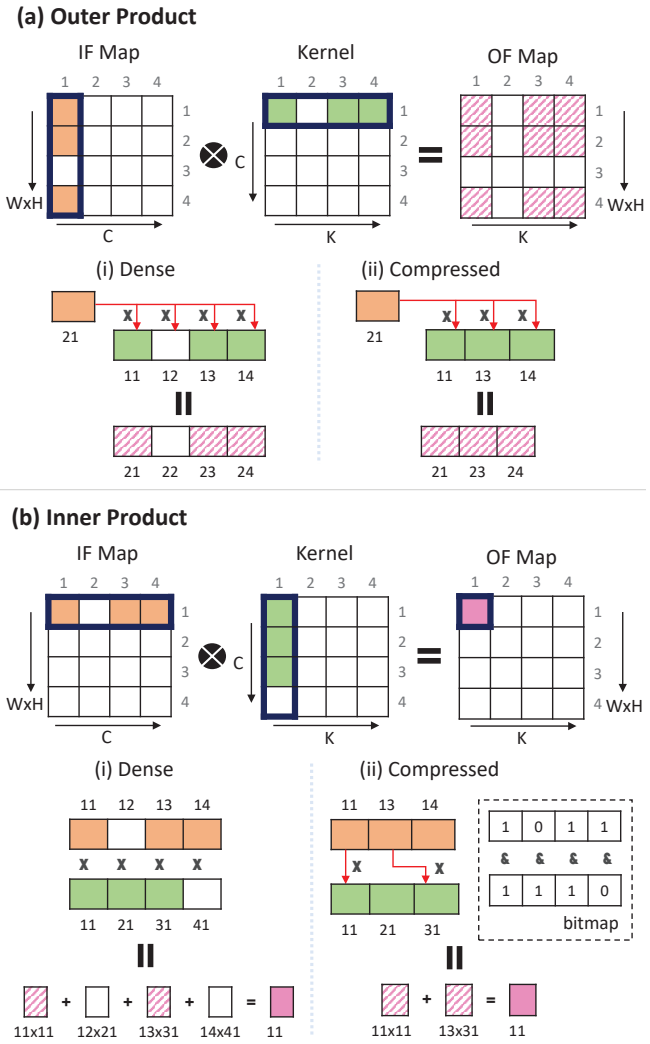

Figure 1: *Examples of (a) outer-product in Pixel-first architectures and (b) inner-product in Channel-first architectures. Strips: Partial sums; Solids: Fully accumulated neuron*

still evolving, they exhibit non-uniform sparsity behavior and load imbalance [16] at runtime that varies by layer and by input.

In this work, we design a new complexity-effective microarchitecture that captures the best elements of prior sparse accelerators, and define a dataflow that leads to high reuse and high utilization. We show that such *microarchitecture-dataflow co-design* can unearth new efficiency opportunities.

The opportunities stemming from weight and activation sparsity have spawned several DNN accelerators [4], [39], [26], [60], [34], [16], [57], [50], [58], [28], [35], [61], [14], [27], [20], [42], [3], [9], [53], [51]. These architectures improve power, throughput, and area by compressing the input activations and weights, followed by computation on this compressed data using different dataflow strategies. Depending on the compression format and dataflow choice, two-sided sparse accelerators can be split into two categories: Pixel-first architectures and Channel-first architectures.

Algorithm 1: Pixel-first Pseudo code

```
for c = 0 to C − 1 do
    for k = 0 to K − 1 do
        for a = 0 to (W ∗ H)_non_zero do
            out[a][k] += in[a][c] * wt[c][k]
```

Pixel-first architectures employ an outer-product strategy for computations (see Figure 1a[1]). It compresses the sparse data such that the non-zero activations and kernels are ordered in pixel dimension[2] for each channel (observe kernel representation in (ii) of Figure 1a). Algorithm 1 shows a simplified pseudo code used by Pixel-first architectures. A vector of non-zero activations and a vector of non-zero weights corresponding to a channel are read to perform a cartesian product. Here, any activation can be multiplied with any weight resulting in partial sums corresponding to several output neurons. The addresses of partial sums are obtained simply by replacing the row index with the row index of activations and column index with the column index of the kernel. Hence, Pixel-first architectures lead to high activation/kernel reuse and simple indexing schemes. However, such dataflows result in little to no partial sum reduction/reuse before writeback leading to high energy consumption. Besides, a cartesian product results in partial sums destined to unrelated output neurons requiring the need for large accumulator buffers and routing logic. For example, SCNN [34] and STICKER [56] are Pixel-first architectures and dissipate over 80% of total on-chip energy in accessing the crossbars and/or the multi-banked accumulator buffers. Most of this high energy is because of frequent traversal over the long wires connecting the crossbar to each bank and accessing those respective banks. Also, intra-PE and inter-PE underutilization are prevalent in Pixel-first architectures. Intra-PE underutilization is caused at feature map and kernel boundaries when the weight or activation vector are not fully populated. Inter-PE underutilization is caused by load imbalance stemming from variance in sparsity levels and work assigned to each PE. In addition, the outer-product model artificially induces nonexistent multiplications at feature map boundaries that cannot be evaded even with padding. These architecturally wasted computations can contribute upto 6.5% of the total computations for two-sided sparse models.

[1]Pixels in a 2D-fmap are linearized and shown as one of the dimensions (similar to a GeMM representation).

[2]Pixel dimension: all dimensions orthogonal to the channel dimension.

Algorithm 2: Channel-first Pseudo code

```
for k = 0 to K − 1 do
    for a = 0 to (W ∗ H) do
        for c = 0 to C − 1 do
            /* Check for channel-index matching
               */
            if (in[a][c] ≠ 0) ∧ (wt[c][k] ≠ 0) then
                out[a][k] += in[a][c] * wt[c][k]
```

Channel-first architectures employ an inner-product strategy for computations (see Figure 1b). It compresses the sparse data structure such that the non-zero activations and weights are ordered in channel dimension for each pixel (see activations (orange) and weights (green) in (ii) of Figure 1b). Algorithm 2 shows a simplified pseudo code used by Channel-first architectures. A vector of non-zero activations and a vector of non-zero kernels corresponding to a pixel are read to perform an inner product operation. Examples of this approach include SparTen [16], SNAP [58], [59], and StitchX [29]. Within each processing element (PE), Channel-first architectures employ output-stationary dataflow to aid the inner product operation. Hence, a significant number of partial sums corresponding to an output neuron can be reduced locally before writing it back to the accumulator buffer, thus avoiding the overheads of crossbars and multi-banked accumulator buffers prevalent in Pixel-first architectures. Since the data structures are compressed in channel dimension, an auxiliary condition is required to find matching activation and kernel index pairs corresponding to the same channel (*if-condition* in Algorithm 2). Failure of the *if-condition* adds extra cycles to the execution time leading to intra-PE underutilization. To prevent these wasted cycles and improve intra-PE utilization, typical Channel-first architectures use auxiliary index-matching logic to prefetch only the operands that satisfy the *if-condition*.This *if-condition* makes Channel-first index generation/matching logic more complex than for Pixel-first architectures. For example, the index-matching logic in SparTen [16] consumes nearly 46% of on-chip power and 63% of on-chip area. Hence, while Channel-first architectures improve buffer energy consumption and

throughput over Pixel-first architectures, the channel index matching logic's power and area introduce a non-trivial overhead. Additionally, inter-PE underutilization due to load imbalance continues to be problematic. While techniques have been proposed to improve load balance [16], they require offline preprocessing techniques to rearrange kernels. Since computations are only performed on matching non-zero values, Channel-first architectures do not suffer from architecturally wasted computations.

We thus observe a significant trade-off between Pixel-first and Channel-first architectures, with the former enabling simpler index-matching logic and the latter enabling efficient aggregation. We propose CANDLES, a microarchitecture and dataflow co-design that combines the best of these two approaches. Specifically, it makes the following contributions.

1) CANDLES employs a Pixel-first compression and Channel-first dataflow to achieve efficient inner join using simple crossbars while circumventing the auxiliary index-matching logic.
2) We propose a 2-level organization for the accumulation buffer with a small set of low energy register files in the first level (L1) and a 6 KB multibanked accumulator buffer in the second level (L2).
3) We introduce a Tiled Pixel-first (TP) compression policy to promote high temporal locality in partial sum updates and, consequently, a higher L1 hit rate.
4) We experiment with different work partitions across PEs and identify regular partitions that achieve a high level of load balance with no offline preprocessing.
5) We explore the design space to identify the network and buffer hierarchy that best matches the capacity/reuse needs of the new microarchitecture and dataflow.

We evaluate the architecture with a synthesized implementation and by simulating the execution of a diverse set of image-based DNNs. We show that CANDLES is up to 5.6× more energy-efficient than state-of-the-art architectures while simultaneously performing at 86-99% of the peak throughput.

II. Background

In this section, we describe details of our baselines: Pixel-first architectures SCNN [34], STICKER [56] and Channel-first architectures, SparTen [16], SNAP [58], [59].

A. Pixel-First Architectures

SCNN: SCNN [34] has 64 PEs with connections to neighbors. Each PE has a 4×4 grid of multiplier units. An input activation buffer and a weight buffer each provide four non-zero activations at a time to perform a Cartesian product. Accordingly, these products are routed through a 16×32 crossbar to 16 of 32 banks that form the Accumulation Buffer. SCNN employs activation stationary dataflow on a subset tile of input activations per PE at a time. The accumulation buffer handles reads and writes to 16 partial sums at a time, each destined to a separate 384-byte bank, thus having a large footprint of engaged circuits. The accumulation buffer is a dominant energy contributor, accounting for over 80% of total accelerator energy. The crossbar and the MAC operations are other non-trivial and roughly equal contributors. Additionally, SCNN exhibits high PE load imbalance stemming from its choice of dataflow and parallelization by assigning different Planar Tiles to each PE. Because each PE may display different activation sparsities in their Planar Tiles, the load assigned to each PE varies significantly. This load imbalance leads to a high level of PE under-utilization and higher latency.

STICKER: STICKER [56] employs nine different modes of operation to handle varying sparsities of activations and kernels across layers. Second, due to the Pixel-first nature of STICKER, the short-term reuse of partial sums is not exploited. Instead, all the partial sums are directed to a large accumulator buffer. Instead of using a multi-banked accumulator buffer like SCNN, STICKER uses a 2-way set-associative PE to handle irregular data. It preprocesses and reorganizes input activations to reduce the conflict for accumulator buffer resources. STICKER saves significant storage area by avoiding the multi-banked accumulator buffer in SCNN. However, the large accumulator buffer remains a dominant energy contributor. Further, due to the conflict for accumulator buffer resources, there is an 8% drop in performance compared to SCNN.

B. Channel-First Architectures

SparTen: SparTen [16] is composed of several PEs, each of which performs an Inner Join operation. Kernels are partitioned and pre-assigned to PEs, while activations are broadcast to all PEs. The inner join performed within a PE corresponds to a single output neuron, thus avoiding a crossbar and multiple partial sum updates within the PE. However, A non-trivial circuit is required to identify matching non-zero entries for the inner join. SparTen's primary benefit is that it outperforms SCNN by roughly 4× with better load balancing. SparTen relies on an offline analysis to sort kernels by sparsity and map them to PEs with a greedy algorithm that balances the load per PE. Because kernels are permuted across PEs, the output neurons undergo a shuffle before they can be represented as compressed output feature maps.

SNAP: SNAP [58], [59] has four cores, a 7×3 PE array per core, and each PE has 3 MAC units. SNAP processes activation and kernels in bundles. An associative index matching (AIM) circuit processes bundles of activations and kernels to find matching non-zero activation kernel pairs. Unlike SparTen, the computations performed by a PE can correspond to more than a single output neuron. SNAP employs a two-level partial sum reduction (PE- and Core-level) to process all the output neurons. The first is PE

level (or intra-PE) channel dimension reduction. The second level of reduction is core-level (or inter-PE) pixel-dimension reduction by moving data over the interconnect network. This two-level reduction technique reduces the write-back traffic. AIM unit is the tradeoff – a large comparator size in the AIM unit negatively impacts the area and power but results in efficient index-matching thereby improving the intra-PE utilization. SNAP does not, however, solve the inter-PE underutilization overhead like SparTen.

III. CANDLES

A. Motivation

There are three main challenges in designing an efficient sparse accelerator:

1) Efficient PSUM aggregation
2) Simple indexing logic
3) Load balancing

State-of-the-art sparse CNN accelerators fall short in addressing one or more of these challenges. Load balancing has been addressed by using a combination of software and hardware techniques in Channel-first architectures. As discussed in Section I, Pixel-first architectures facilitate simple index-matching logic at the cost of inefficient PSUM aggregation, with the opposite being true for Channel-first architectures. PSUM aggregation efficiency is attributed to the presence of temporal locality in partial sums (outlined in the next paragraph). CANDLES uses microarchitecture-dataflow co-design to adopt the best of both architecture styles and address all three challenges.

Role of Temporal Locality: To explain the locality effect in various strategies, consider the illustrative example in Figure 2. The colored dots show the cycles when each entry in the accumulation buffer is updated. Each block in

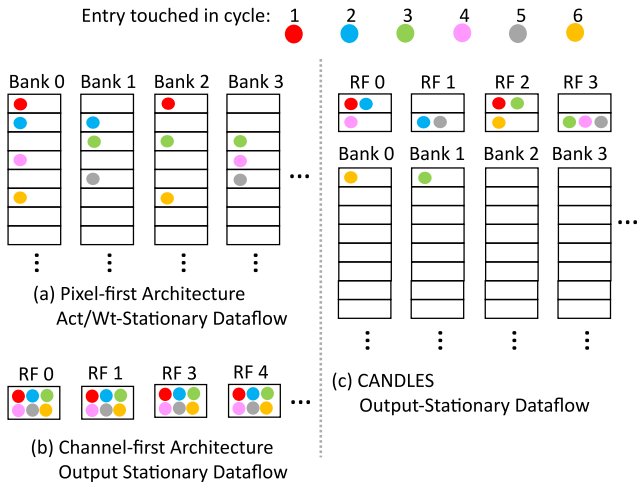

Figure 2: *PSUM access pattern in consecutive cycles for (a) Accumulation Buffer banks in Pixel-first architecture, (b) RF in Channel-first architectures, and (c) CANDLES.*

bank and register file (RF) represents a partial sum entry. In Pixel-first approach, within a cycle, updates are scattered to multiple accumulation banks (or buffers). As a result, the partial sums updated in consecutive cycles are often different requiring large accumulator buffers. Larger buffers lead to higher energy per access. This partial sum update pattern with little temporal locality is shown in Figure 2a and is a key factor in the accumulation buffer's dominant energy contribution. In Channel-first approach, since we first traverse through the channel dimension, partial sums in consecutive cycles correspond to the same output neuron, requiring only a small entry accumulator (a register file) to capture this pattern (see Figure 2b). In CANDLES, we retain the Pixel-first compression strategy. However, the dataflow is modified to be closer to output-stationary similar to Channel-first architectures, i.e., we traverse the activations and weights such that partial sum updates in consecutive cycles exhibit much higher temporal locality. This allows us to decompose the accumulation buffer into a 2-level structure with a high hit rate in the L1. As shown in Figure 2c, most of the updates in the first six cycles are localized to each bank/buffer's few entry L1 register file.

B. High-Level Overview

We introduce a synergistic combination of four key innovations. First, a Pixel-first compression and Channel-first dataflow architecture (PFCF) is implemented to achieve efficient inner join without the need for complex index matching logic. Second, a two-level accumulator buffer captures the reuse of partial sums; and third, a novel compression algorithm ensures high locality among consecutive partial sums. Fourth, a memory partitioning scheme ensures load balance without the need for software optimizations.

Figure 3 shows the microarchitecture of CANDLES. It consists of a central buffer and an 8x8 grid of PEs connected via the mesh network. The central buffer is responsible for distributing activations of each layer to individual PEs over the mesh network. The central buffer is also equipped with pool and ReLU modules.

Each PE consists of 3 buffers to store activations, weights, partial sums, a 4x4 multiplier array, a PSUM filter, a simple crossbar structure, and an index-generation logic. The heart of CANDLES PE is the PSUM filter that captures the reuse of partial sums for an energy-efficient accumulation. To reduce write-back traffic, we also support cross-PE reduction.

In a cycle, the activation and weight buffers provide input data structures to the 4x4 multiplier generating 16 partial sums. The index-generation logic computes the output neurons' addresses for these partial sums in parallel with the cartesian product. The resultant partial sums are stored in either the PSUM filter or the accumulator buffer. Individual PEs are populated with weights from off-chip DRAM. Once loaded, a set of weights are fully exhausted with all the available activations before fetching the next set (weight-

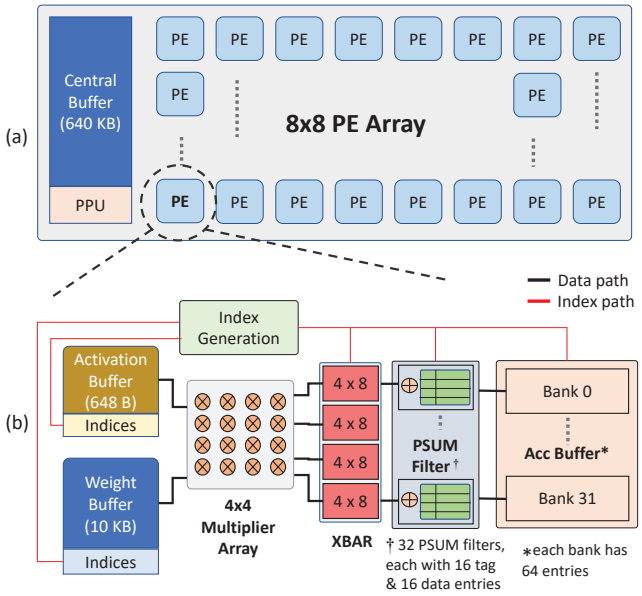

Figure 3: *CANDLES Microarchitecture.*

stationary dataflow at a high level). Activations, in contrast, are accessed to/from the central buffer.

C. Pixel-first Compression and Channel-first Dataflow

We now discuss the impact of dataflow on temporal locality. Please note that the proposed dataflow is tailored specifically for a microarchitecture with hierarchical accumulator buffers. To reduce the common-case reuse distance, we introduce the following dataflow in CANDLES, with code and examples shown in Figure 4. Each PE processes four non-zero activations corresponding to C_t channels in consecutive cycles. Hence PE0 is allocated four non-zero activations from the first C_t channels (green and violet), while PE1 is allocated four non-zero activations from the next C_t channels (orange and light-brown). In the first cycle (Figure 4), the multiplier array (say PE0) is fed with the first four (green) activations and the (red) first weights of the first four kernels (again, all from one input channel). These products correspond to four partial sums for each of four different output channels. In the next cycle, we switch to a different input channel and similarly fetch the first four (purple) activations and first (light green) weights from the first four kernels. Thus, the partial sums touched in the first two cycles belong to the same four output channels.

Further, as we rotate through several channels in consecutive cycles, the generated products all pertain to a localized region of four output channels, thus concentrating most updates to a small set of elements in the accumulation buffer. After rotating through all (Ct) input channels assigned to this PE, we rotate back to (green) activations from the first channel and move to (blue) weights from the next set of four kernels, thus producing partial sums for a localized region in the next four output channels. Thus, a new set of activations and weights are fetched from their buffers every cycle, increasing the (low) energy expended in the activation and weight buffers.

Once all the PEs finish performing local reduction of Ct-channels, every alternate PE transfers its partial sums to the neighboring PE via the grid network for inter-PE reduction. The receiving PE uses the adders to aggregate the received partial sums with the ones in its accumulation buffer. Note that the receiving PE does not perform multiplication operations during inter-PE reduction.

Cacheability: With the CANDLES dataflow, we observe a significant overlap between the partial sums touched in consecutive cycles. Therefore, even a small cache of partial sums can yield a very high hit rate with a single entry per bank. Figure 2c provides a specific example. In practice, the positions of non-zero activations in each channel will not line up perfectly, thus generating more misses or requiring more entries per bank to yield a high hit rate. Further, once weight sparsity is included, the partial sum updates are more scattered, again requiring multiple entries per bank to yield a high hit rate. We propose tiled-compression techniques to limit the scattering of partial sums to a small set (Section III-E).

D. The PSUM Filter

By revisiting the partial sums for the same output neurons in consecutive cycles, the above dataflow is most similar to an output-stationary dataflow. It therefore presents an opportunity to partition the accumulation buffer into two levels. The most recently accessed partial sums are moved into a small tagged cache, the PSUM Filter, to service the expected high temporal locality while other partial sums with a longer reuse distance are placed in a 6 KB second-level buffer similar to the accumulation buffer in baseline SCNN (see Figure 3). As we show in Section V, the PSUM Filter yields high hit rates even with 16 or fewer entries per bank. It is implemented as a set of registers along with accompanying tags. We layout the PSUM Filter adjacent to the crossbar's output ports (Figure 3). This reduces long interconnect traversal for PSUM Filter access.

Implementation Details: The PSUM Filter for each bank is fully associative. Each entry is associated with a 6-bit tag that points to one of the 64 entries in the L2 bank. The index generation logic produces a 11-bit tag for each generated product – five of these bits identify the bank, and six identify the entry within the bank. The tag check is performed along with output neuron index generation. Recall that index-generation is performed in parallel with the longer latency Cartesian Product. Therefore, by the time the product emerges from the crossbar, the hit/miss information is available. The partial sum proceeds with either accessing the Filter or the L2. Both structures are accessible in a single cycle, so a Filter miss does not impose a performance

PE loop-nesting code:

```
# Overview: K/Kt →R → S → N → W → H → Kt→ C
BUFFER wt_buf [R*S*K/Kt][Kt/4][C][4]
BUFFER in_buf [N][W*H/4][C][4]
BUFFER acc_buf [N][Kt][W][H]
BUFFER central_buf [N][Kt][W*H]

for k' = 0 to K/Kt -1: # Iterate through non-zero values of the kernels
 for w = 0 to R*S -1:  # Step over all weight values
  for n = 0 to N-1:  # Iterate over a batch of images
   for a = 0 to W*H/4 -1:  # reuse weights over all activations
    for k = Kt - 1:  # Across all kernels within the weight buffer
     for c = 0 to C-1:  #Accumulate across all channels
     {
       in[0:3] = in_buf [a][n][c][0:3]  # Get 4 non-zero activations
       wt[0:3] = wt_buf [w][k'+k*K/Kt][c][0:3]  # Get 4 non-zero weights,
                                                # 1 from each kernel
       parallel (i= 0 to 3) * (f= 0 to 3):  # in each multiplier
         k = Ksparse_coord(c,f)  # get output coordinates of k
         x = Xsparse_coord(a,i,w,f)  # get output coordinates of x
         y = Ysparse_coord(a,i,w,f)  # get output coordinates of y
         acc_buf[n][k][x][y] += in[i]*wt[f]  # multiply & accumulate to
                                  # respective output neurons
     }
     central_buf [n][a][k'+k*K/Kt][0:W*H-1] =
     acc_buf[n][k'+k*K/Kt][0:W-1][0:H-1]  # push acc_buf to out_buf
```

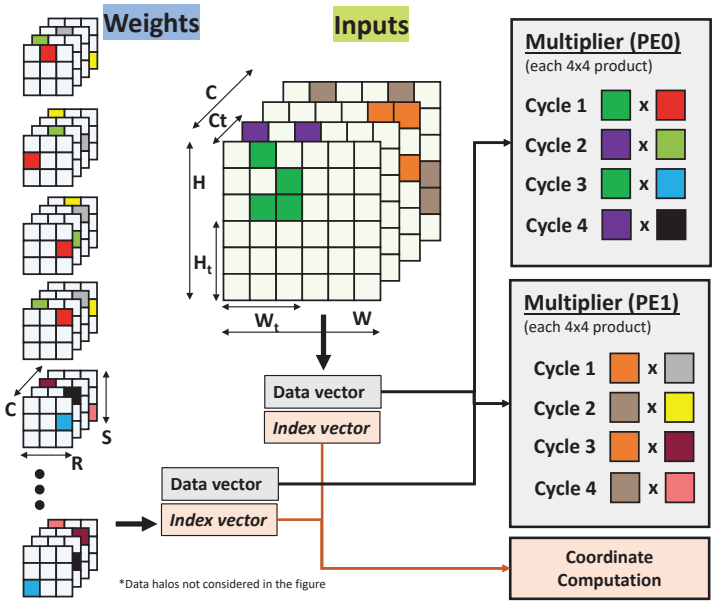

Figure 4: *Code and example of proposed dataflow with higher temporal locality. In each cycle for multipliers PE0, PE1, operands in left denote values from input activations, and operands in right denote values from weights.*

penalty. On a Filter miss, the Filter and L2 both perform parallel read-modify-writes while swapping entries in the Filter and L2. As shown later, hit rates are not sensitive to replacement policy parameters.

E. Tiled Pixel-first Compression

We make the case that the conventional Pixel-first compression approach can significantly impact the PSUM Filter hit rate. In a typical kernel or feature map, the distribution of zeros is non-uniform. This non-uniformity can result in non-zero outlier values substantially impacting the PSUM Filter hit rate when using the CANDLES dataflow.

Consider an example feature map shown in Figure 5a using the conventional implementation of Pixel-first compression (Figure 5b) in state-of-the-art architectures. The non-zero values are stored in an array along with an index vector (not shown in the figure) that encodes metadata. The numbers in each cell indicate the coordinates in pixel dimension, whereas the color indicates different channels. Assume that we have a single dense 1×1 kernel with the same number of channels as the example input feature map. We walk through this sample benchmark using the CANDLES microarchitecture and dataflow to highlight the overheads incurred when compressed using the conventional implementation.

Recall that CANDLES dataflow traverses across the channel dimension four non-zero values at a time. In two cycles, four non-zero values from both the channels are processed. We can observe that as we traverse through the channel dimension, pixels 11 and 14 are touched in both cycles. Note that we assume that the pixels stored in the PSUM

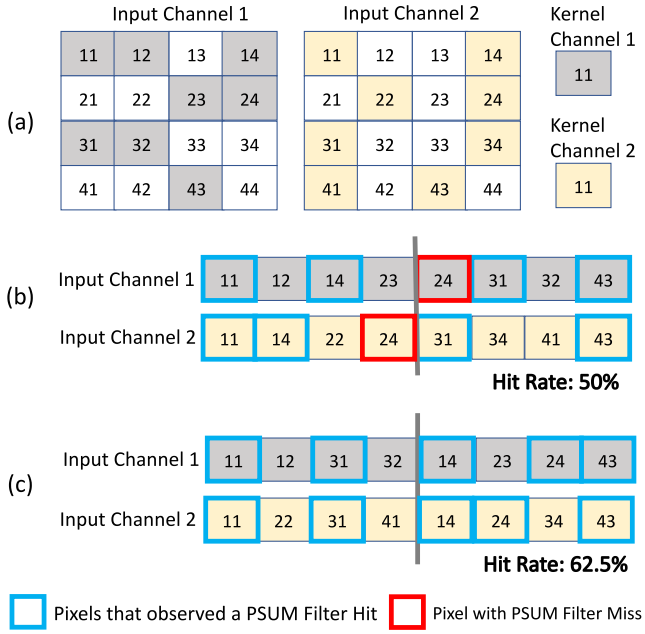

Figure 5: *A sample feature map and kernel (a) compressed using conventional Pixel-first compression (b), and the proposed Tiled Pixel-first compression strategies (c). Grey color denotes the non-zero values in channel-1, Yellow color denotes the non-zero values in channel-2, and white denotes the zero values in both the channels*

filter during the first iteration are evicted before the second iteration begins. This is because, in a typical deployment scenario, we traverse across 64 channels at once, and there is a very low probability for the value to remain in the cache due to the bimodal reuse distances we observed. During the second iteration, pixels 31 and 43 are touched, resulting in a total PSUM Filter hit rate of 50% to execute the feature map fully. Note that while pixel-24 also has a matching non-zero value in both channels, its locality is not being captured by this compression strategy. This is the result of the non-zero value in pixel-23 of the first channel. While both the channels have the same number of non-zero values, they are not evenly distributed across the pixel dimension. Since we only take four non-zero values at a time, the non-uniform density distribution results in the dataflow not capturing the locality of pixel 24. This gap is exacerbated in real applications, leading to under 40% hit rates.

We observe that grouping the pixels before compressing can significantly reduce the non-uniformity in distribution, thereby yielding a higher PSUM Filter hit rate ($> 85\%$). This is the motivation for our Tiled Pixel-first (TP) compression. We tile the feature map into multiple groups before compressing them. During compression, the non-zero values are stored one tile at a time. Consider tiling the previously discussed example feature map into two equal parts, with each part having only two of the four columns. Figure 5c shows the compressed data structure with only the non-zero values using the TP compression strategy. Implementing the CANDLES dataflow on this new data structure results in a higher (62.5%) PSUM Filter hit rate. This is because the placement of a bounding box on the pixels limits the scattering of non-zero values in the compressed data structure. We observe that the PSUM Filter hit rate is directly proportional to the number of tile partitions. However, an extremely small tile size can result in intra-PE underutilization (discussed later in Section V). Experimental analysis shows that a tile size of 7×4 ensures high hit rate without sacrificing much of intra-PE utilization.

Additionally, while traversing through the channel dimension, the computations can be skipped entirely for the respective channel if we encounter an empty feature map or kernel. This is acheived by allocating a valid bit for each channel of the feature map and kernel.

F. Load Balancing across PEs

We now discuss how work is partitioned across multiple PEs to promote load balance. CANDLES allocates the same number of non-zero activations (in the common case) and an $N \times N$ partition of weights to each PE. The load imbalance is primarily determined by the sparsity variation in kernel partitions across individual PEs. This is different from Pixel-first architectures like SCNN where each PE has a duplicate copy of the weights, and where load imbalance is determined by the sparsity variation in activation partitions. While

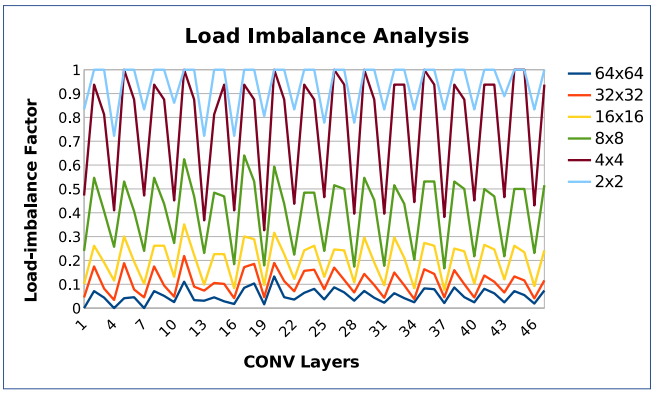

Figure 6: *Load-imbalance (between most and least busy PEs) across layers of ResNet50 as N is varied (lower is better).*

both approaches may seem equivalent, unlike weights, the sparsity of activations change dynamically across different layers of the network for each image. This makes it hard to determine the ideal distribution of activations across PEs during run time. On the other hand, the sparsity of weights does not change during inference, allowing us to perform offline analysis.

Partition Design Space: Returning to the example in Figure 4, we see that PE0 and PE1 are both assigned just 2 (input) channels each and 8 kernels each. We refer to this partition as "2 $\times$ 8". The computations required for a convolutional layer can be expressed as inputchannels$\times$ kernels$\times$ A $\times$W, where A is the set of non-zero activations in a 2D input channel and W is the set of non-zero weights in a 2D kernel channel. That total computation must be split across 64 PEs in our architecture. For now, we will assume that the weights in one channel of one kernel are not partitioned across PEs, i.e., we are not partitioning W. In a typical convolutional layer with many channels and kernels, adopting a "2 $\times$ 8" partition would imply that each PE receives a small share of channels and kernels but a large share of each input feature map channel. On the other hand, adopting a "64 $\times$ 64" partition would imply that each PE receives a large share of channels and kernels but a small share of each input feature map channel.

Empirical Analysis: We are trying to estimate the partition of work across PEs that minimizes load imbalance. To simplify the control logic and avoid any offline analysis, we are attempting a partition by drawing lines at regular intervals. Figure 6 quantifies this load imbalance for a number of "N $\times$ N" partitions. We see that it is clearly beneficial to use large N; for N = 64, the load imbalance is under 10%. This partition is consistently balanced across different layers, unlike SCNN that sees higher load imbalance when feature maps shrink in later layers. Multiple factors play a role in this empirical observation. There is indeed a large variation in sparsity across individual kernel channels. For example,

suppose we assume that a convolutional layer has 3 × 3 × 128 × 128 weights. In that case, the number of non-zero weights in each channel in each kernel will be a list of 16,384 integers ranging from 0-9 with high variation: 3, 7, 2, 4, 7, 0, 5, 9, ... If each PE is assigned a small consecutive subset of this list, the variation in load across PEs will be higher than if we assigned a large consecutive subset of this list. In other words, a large sample averages out the high variation across kernel channels, favoring a large N. Second, by using large N, each PE is assigned a smaller fraction of the input feature maps. It can be argued that smaller feature map samples may lead to higher variation in non-zero activation tuples per PE – however, this effect is alleviated because these activations are spread across N channels, making the sample more diverse than if those activations were from a few channels.

G. Microarchitecture Design Choices

We now discuss the impact of our new PFCF dataflow and new work partition on the proposed microarchitecture.

Weight and Activation Buffers: CANDLES reads a new tuple of weights and activations every cycle and exhibits activation and weight reuse with varying reuse distances. We therefore size the weight and activation buffers to capture the resulting reuse pattern. Given our choice of a "64×64" partition for most layers, a PE is assigned 4K weights per layer at a time. Including the index metadata, we allocate a 10 KB buffer to store these weights. These weights are fetched from DRAM, reused completely, then evicted to make room for the subsequent 4K weights from DRAM. An activation is re-visited after cycling through 64 different channels assigned to the PE; the activation buffer is therefore large enough to store 256 activations (648 B, including index metadata).

Accumulator Buffer: The design choice for accumulator buffer size captures the worst-case partial sum scattering scenario. Since we use a 7x4 (=28) tile size for activations and 64 unique kernels in the weight buffer, the worst-case scenario accommodates a maximum scattering of 1792 partial sums (28x64). With 24-bit PSUMs, a 5.25 KB accumulator buffer is required, which we round up to 6 KB because of limitations in our memory compiler.

Central Buffer: Once a set of weights is brought into the weight buffer, it has to be consumed by all the activations assigned to that PE. Since the activation buffer only handles 256 entries at a time, it has to be re-filled periodically. To accommodate this reuse pattern for activations, the activation buffer is organized as a two-level hierarchy. The 640 B first level captures most of the reuse. The second level is a 640 KB central buffer that all the PEs share; it is responsible for the periodic re-fill of the first level, and it captures the longer-distance reuse pattern in the activations. The central buffer is preceded by a pre-processing unit (PPU) responsible for applying the activation function and creating the compressed output feature map. While aggregation across channels take place at PE-level, aggregation across convolution filters (ex: 3×3) is usually performed at Central-Buffer. Since the final aggregated PSUM is only present in the central buffer, we just place pool/ReLU units next to it. Further, support for much larger batch sizes can be accomplished by simply increasing the central buffer size with no modifications to PE micro-architecture.

Simpler Crossbar: A natural consequence of our dataflow is that the 16 partial sums generated in a cycle are split into four parts, each corresponding to a different output channel. The four partial sums in each part are split across 8 PSUM filters using a small 4×8 crossbar. This is significantly smaller than the 16×32 crossbar implemented by SCNN. The four PSUMs entering each of the 4×8 Xbar correspond to a multiplication between four different input pixels and a single kernel entry. This results in PSUMs corresponding to four unique indices. Hence no two PSUMs computed in the same cycle will have the same output index.

Activation Metadata: Since the feature maps of initial layers are large, the metadata overhead can be non-trivial with a naive approach that stores w and h indices. For activations, we adopt a slightly different indexing mechanism than prior works. We use a hybrid RLE approach where for every four non-zero activations, we use a combination of absolute indices and RLE style zero indices. The index of the first activation stores its w and h indices, while the remaining three store the number of zero occurrences since the last non-zero activation. Since each tile is only 7x4, a 5-bit value is used to store the absolute indices for one of every four non-zero activations in the tile. The rest of the non-zero activations in the tile use a 4-bit zero index similar to RLE. Since PEs process one tile at a time, we have to store a 2-byte tile index in the index-generation logic to account for the tile offset.

Kernel Metadata: Since typical kernels are usually small (1×1 or 3×3), we store the absolute indices of all the non-zero weights. 4-bit metadata for each non-zero weight is sufficient to store the absolute indices for all our benchmarks.

Wasted Computations: When performing outer-product computations, some multiplications involving feature map boundary elements do not contribute to output neurons and are therefore wasted. This reduces effective throughput and wastes energy for all Pixel-first architectures, including CANDLES. As we show later, this impact is relatively minor, especially given recent trends towards small kernel dimensions.

IV. Methodology

We compare the CANDLES architecture against four state-of-the-art sparse neural network accelerators: SCNN, STICKER, SparTen, and SNAP. We primarily report iso-resource (same number of MAC units) comparisons.

Energy and area modeling: To get accurate estimates of energy and area, we modeled CANDLES and other baseline Pixel-first architectures in Verilog, implemented them using industry-standard synthesis, place-and-route tools in a 65 nm CMOS process. SRAM memories with the targeted dimensions were compiled using a vendor-provided memory compiler. The energy dissipation numbers obtained from the place-and-route tool's power report are combined with memory access energy (read and write) to get the average power dissipation. To accurately estimate the multi-banked accumulator buffers' overheads in both CANDLES and SCNN, we first modeled a single accumulator bank using the memory compiler. We later placed 32 instances of the bank in a grid structure during layout, with each column having the same number of banks required to match the crossbar's height. We placed 64 instances of the modeled PE next to the central buffer during layout. To model the mesh interconnect, we have estimated the wire length required to move data across PEs from the obtained layout. We used a conservative estimate of 0.1 fF/micrometer wire capacitance from the technology library and estimated the wire energy and delay based on the wire length. We did not model the synthesized implementation of Channel-first architectures as the index-matching logic complexities are hard to model in enough detail to get meaningful energy and area numbers. Instead, we have directly used the power and area numbers reported in those respective works. Note that each baseline architecture uses different datawidths for computation and storage. To have a fair comparison against other accelerators, we have modeled three CANDLES variations based on the datawidth – an 8-bit MAC with 24-bit partial sums, a 16-bit MAC with 24-bit partial sums, and an 8-bit MAC with 8-bit partial sums.

Simulator modeling: We built a combination of a cycle-accurate simulator and analytical simulator to accurately estimate performance. For SCNN, we explored a range of feature map partitioning schemes. We observed that while SCNN's proposed partitioning scheme is the most energy-efficient version, it is not ideal for performance. For that reason, we have considered two variations of SCNN: SCNN-E and SCNN-EP as baselines. SCNN-E is the most energy-efficient variation, while SCNN-EP obtains the best energy-delay product. STICKER uses different compression formats depending on the level of sparsity. To ensure an apples-to-apples comparison and isolate the impact of CANDLES, we assume that all layers are compressed using CSR for STICKER. For all the architectures, the simulator accurately captures both intra-PE and inter-PE underutilization. We have configured CANDLES to handle two-sided sparsity and scenarios where only one of the two data structures (activations or weights) is sparse.

Benchmarks: We executed four CNN workloads: VGG16 [44], ResNet-50 [19] (ResNet50-A), Inception-v1 [47], and MobileNet-v1 [23]. We use VGG-16 as a proxy for large input data. While experiments on the above four workloads were carried out with dense kernels, we also consider a fifth workload with sparse kernels: a publicly available pruned checkpoint of ResNet-50 (ResNet50-AW) trained on ImageNet [22]. Since we do not have more pruned networks at our disposal, we have synthetically pruned the top 50% weights closer to zero of MobileNet (MobileNet-v1-AW*) for our evaluation of two-sided sparsity. Note that we only pruned weights extremely close to zero ($-0.03 \leq 0 \leq 0.03$). Note that prior works [39], [40] have used iterative pruning and training to achieve a range of sparsity and accuracy levels.

We execute the above workloads on 2000 images from the Imagenet [22] dataset, feeding the dynamically generated activations to simulated models of SCNN-E, SCNN-EP, STICKER, SNAP, SparTen, and CANDLES. These sample images were collected from diverse image classes.

V. Results

A. Energy

We first quantify the energy per inference. We use an LRU replacement policy, a 16 entry PSUM Filter per bank, and a tile size of 7×4 for most of our experiments. Table I summarizes the energy consumed by individual components in all three variants of CANDLES.

Component	16/24-b Energy per access	8/24-b Energy Energy per access	8/8-b Energy per access
Weight buffer	24.5	17.1	17.1
Activation buffer	19.6	13.1	13.1
MAC	1.94	0.24	0.24
Crossbar	8.09	1.62	1.62
Accumulator buffer energy / bank access	8.7	8.7	5.85
PSUM Filter	1	1	0.33
Tag lookup	0.114	0.114	0.114
Central Buffer (80-bit datawidth)	41.6	41.6	41.6
PPU (80-bit datawidth)	0.285	0.285	0.285
Interconnect-Energy/nanometer/bit	0.0216	0.0216	0.0216

Table I: *Energy per access for each component in all 3 variations of CANDLES in pJ at 65 nm CMOS technology.*

Importance of Microarchitecture-Dataflow Codesign

To isolate the impact of each contribution and highlight the importance of microarchitecture-dataflow codesign, we consider several variants of CANDLES with one or more primitives – dataflow, PSUM Filter, and TP-Compression. Figure 7a plots the impact of each variation on energy consumption normalized to SparTen's energy.

The first variant only considers CANDLES with the proposed Pixel-first compression and Channel-first (PFCF) dataflow. CANDLES is up to 57% more energy-consuming than SparTen. This is because of two reasons. First, as

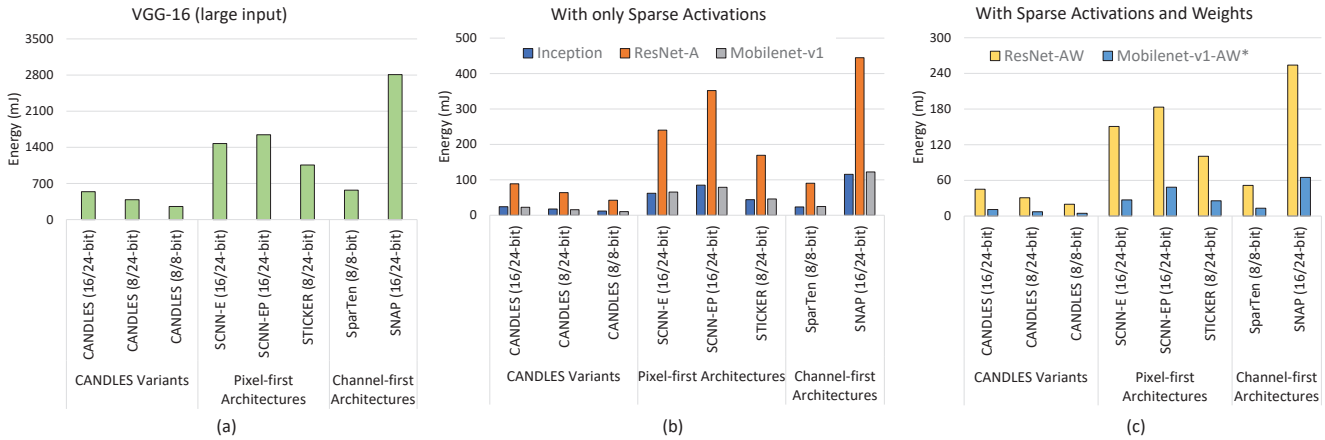

Figure 8: *Energy consumption for CANDLES and baseline SCNN-E, SCNN-EP, STICKER, SparTen, and SNAP.*

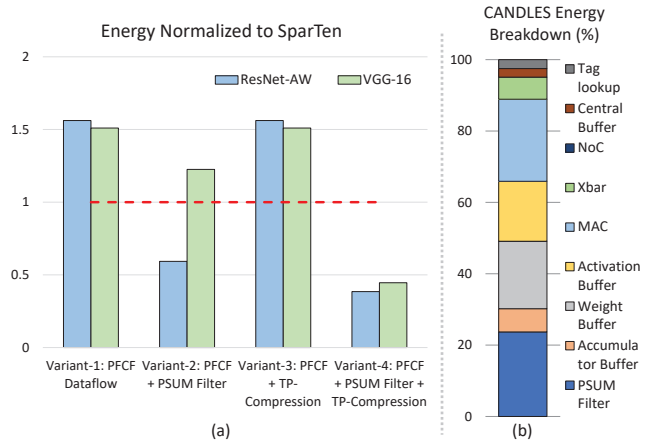

Figure 7: *Energy breakdown in CANDLES.*

discussed previously, the PSUM reuse is under 40% for most layers without the TP-compression. Second, since there is no PSUM Filter to capture the available reuse, all the partial sums are redirected to the large accumulator buffer resulting in high energy per access. However, this variant is 1.2× more energy-efficient than SCNN-E, and 1.45× more energy-efficient than SCNN-EP when executing the benchmarks. This is because of better crossbar structures, higher MAC utilization, and efficient dataflow of CANDLES.

The second variant considers CANDLES with the PFCF dataflow and the PSUM Filter but without TP-compression. This limits the reuse captured by the PSUM Filter as the initial layers suffer with lower hit-rates (see Figure 10b). Variant-2 is between 1.3 – 2.6× more energy-efficient than variant-1.

The third variant considers CANDLES with the PFCF dataflow and TP-compression but without the PSUM Filter. While the PSUM reuse is increased to >85% with TP-compression, the energy consumed is similar to variant-1 because of the lack of a PSUM filter to capture this reuse.

The final variant considers all three primitives. We see that CANDLES is 2.6× more energy-efficient than SparTen. This is because the PSUM Filter now captures all the partial sum reuse enabled by TP-compression. As the PSUM-Filter energy is 8.7× smaller than accessing the accumulation buffer, high reuse leads to reduced energy consumption.

CANDLES Energy Analysis

Figure 7b shows a breakdown of energy dissipation for each component in the proposed architecture for Resnet50 benchmark with two-sided sparsity. The rest of the benchmarks also observe a similar breakdown of energy. Because of the new dataflow, CANDLES dissipates more energy in its activation buffer despite its smaller size. However, this energy consumption increase is offset by the much lower energy in the accumulation buffer and crossbar. The more compact crossbar in CANDLES consumes nearly 3× less energy compared to the baseline SCNN crossbar. Both interconnect and PPU consume less than 1% of the total energy (NoC in Figure 7b). This is because, except to execute depthwise convolutions, the only purpose of PPU is to compress the output neurons before processing the next layer. Each neuron is only read once. Since the number of computations is orders of magnitude higher than the number of activations, the PPU's share of energy is low. The same argument is applied for interconnect to the central buffer; it is only used for a single exchange of data between the PE and central buffer, whereas the number of PE operations initiated by that exchange are orders of magnitude higher.

Wasted Computations: CANDLES due to its Pixel-first compression incurs architecturally wasted computations like other Pixel first architectures. However, these wasted computations contribute to less than 6.5% of the total energy consumed by CANDLES across all the benchmarks. Modern benchmarks with kernels of dimension 1x1 incur no wasted computations.

Figure 8 shows the energy consumed by CANDLES and baseline architectures (SCNN-E, SCNN-EP, STICKER, SparTen, and SNAP) when executing the benchmark applications. We denote the datawidth for MAC and partial

sums next to the respective architecture to understand the energy benefits of CANDLES better. For example, an 8/24-bit denotes an architecture with 8-bit MAC units and 24-bit partial sums. In SCNN, every new partial sum generated should access the crossbar and the accumulator buffer. This frequent access to these large structures is a significant contributor to SCNN's energy. SCNN-EP ensures better parallelism by choosing the appropriate tile size to distribute the load. This results in increased writes of data structures and hence more energy compared to SCNN-E. STICKER benefits from the reduced area by replacing the crossbar with a set-associative PE and using smaller accumulator buffers. However, this does not aid with saving significant energy compared to SCNN. The partial sums are still written to an accumulator buffer with a similar size as a single bank in SCNN's accumulator buffer. Additionally, similar to SCNN, the partial sums are scattered, and no reuse is captured locally next to the MAC units. All these factors contribute to the energy in STICKER.

SparTen, on the other hand, uses Channel-first dataflow and hence completely captures the reuse of partial sums into a small register near the MAC units. Additionally, it replaces the large crossbar in SCNN with a simple permuter, saving energy. However, this benefit is offset by the use of complex index-matching logic. In SparTen, nearly 46% of on-chip power is consumed by the priority encoder and the prefix-sum circuits. SNAP, similar to SparTen, is a Channel-first architecture with a high share of power and area consumed by the index-matching logic. Additionally, SNAP does not capture the reuse of partial sums as efficiently as SparTen. This is because the intra-PE utilization efficiency depends on the comparator's size in the index-matching logic (associative index matching unit). Increasing the size of the comparator increases the area and power quadratically, which is not desirable. Alternatively, not capturing the reuse of partial sums locally will result in accessing the larger buffer in the next level of hierarchy. All these factors contribute to high energy consumption in SNAP. Overall, CANDLES is up to 3.3×, 4×, 3.2×, 2.5×, and 5.6× more energy-efficient than SCNN-E, SCNN-EP, STICKER, SparTen, and SNAP architectures. Note that we assumed similar datawidths for CANDLES as its respective baseline for this comparison.

B. Performance

We next compare the performance for CANDLES and the baselines. Figure 9 shows the throughput (Tera Operations per Second) of CANDLES for all the benchmark applications, relative to SCNN-E, SCNN-EP, STICKER, SparTen, and SNAP. We consider two variants of CANDLES (CANDLES-A and CANDLES-U) for this analysis. CANDLES-A shows the absolute TOPS for all the computations performed by CANDLES, which includes the architecturally wasted computations, whereas CANDLES-U only considers the useful computations for measuring TOPS. The share of architecturally wasted computations is between 0-6.5% of the total computations. Both variants of CANDLES are over 4× faster than SCNN-EP on benchmark applications with only sparse activations and over 2.5× and 2× faster over ResNet-AW and MobileNet-AW*, which have both sparse activations and weights. A vital reason for this gap is the presence of intra-PE and inter-PE under-utilization in SCNN. SCNN-E is an additional 8% slower than SCNN-EP. On the other hand, CANDLES achieves a high load balance due to its efficient work partitioning and buffer size choices. STICKER uses a 2-way set-associative PE for partial sum accumulation. When there's a conflict, it takes two cycles to update the partial sums. While STICKER proposes shuffling of data to avoid conflicts, it does not fully solve the problem. Our STICKER analysis showed that the conflict rate can be between 1-15% across the layers of the benchmark applications. Overall, STICKER is up to 5× slower than CANDLES.

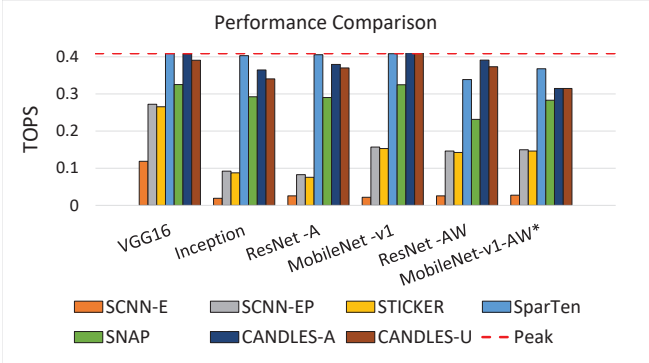

Figure 9: *Performance comparison. Performance is expressed as TOPS (higher is better).*

SNAP's channel-first dataflow ensures that partial sums are reduced before they are written back to the output activation buffer. This partial sum reduction results in a significant drop in congested writeback traffic and contention at the output activation buffers, thus improving performance. While SNAP eliminates a large fraction of intra-PE under-utilization, it does not address the load imbalance across PEs due to the implicit barriers imposed by the broadcast bus. This inter-PE underutilization is resolved by SparTen using greedy-balancing techniques and hardware co-optimizations. In contrast, CANDLES is not limited by the implicit barriers and achieves load balance by using sufficiently large weight buffers, as discussed before. CANDLES is up to 68% and 15% faster than SNAP and SparTen. However, when sparse activations with dense kernels are considered, SparTen can perform up to 1.1× faster than CANDLES. This is because SparTen broadcasts the activations allowing all the PEs to finish computations at the same time. Hence for sparse activations alone, SparTen's performance is very close to an ideal peak throughput. However, our analysis shows that CANDLES consumes 10% less area than SparTen.

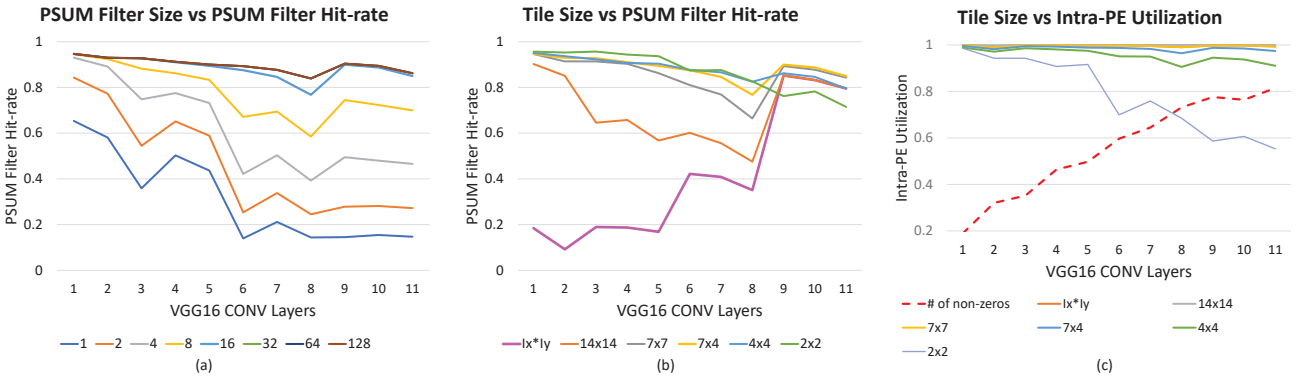

Figure 10: *For each CONV layer of VGG-16, (a) plots the variation of PSUM Filter Hit-rate w.r.t. PSUM Filter size, whereas (b),(c) plots the variation of PSUM Filter Hit-rate and intra-PE utilization w.r.t. Tile size. (c) also plots sparsity.*

CANDLES would therefore out-perform SparTen in an iso-area comparison (note that most reported results are for an iso-MAC comparison). Overall, CANDLES runs at 86-99% of peak throughput across all the benchmark applications.

The performance improvement observed is the result of both microarchitecture and tiling optimizations. Inter-PE utilization is improved due to better load balancing (which depends on weight buffer size), and intra-PE utilization or compute utilization is improved by efficient tiling (which depends on tile size). A larger weight buffer results in a large sample of weights per PE which averages out the high variation across kernels promoting inter-PE load balance (Section III-F). An ideal tile size ensures high MAC utilization in each PE promoting intra-PE load balance. Additionally, by using a grid network, CANDLES avoids implicit barriers imposed by the broadcast network in baselines.

We have also explored the impact of tile size on baseline SCNN. In Figure 9, SCNN-E represents the performance of baseline SCNN, and SCNN-EP represents SCNN with tile size obtained by our proposed approach. We observe that the performance of SCNN is increased by $2.5-7\times$ over the baseline SCNN. This is due to the increased PE-utilization from better tiling. While tiling can help improve intra-PE utilization in baselines, the choice of microarchitecture limits them from getting better load balance across the PEs. CANDLES, due to its microarchitecture and tiling, is at least $2\times$ faster than SCNN-EP.

C. PSUM-Filter Sensitivity Analysis

We next examine how PSUM Filter hit rates vary as a function of various parameters.

Replacement Policy: We explore many replacement policies, including LRU, Second chance, LRU Insertion policy (LIP), and Bimodal insertion policy (BIP with ε ranging from 1/2 to 1/64) [36]. We observe that the replacement policy negligibly impacts the hit rate because of the bimodal reuse distance nature of partial sums. The very short reuse distances are always captured, and the very long reuse distances are not captured by the PSUM Filter, regardless of replacement policy.

PSUM Filter Size: Figure 10(a) plots the average hit rate across our set of images while executing each CONV layer of VGG16 with various PSUM filter sizes. It shows that the hit rate drops as we transition to deeper layers. The improvement in hit rate saturates beyond 16 entries per bank.

Tile size for TP-Compression: Figure 10(b) plots the variation of PSUM Filter hit rate with varying tile sizes. A tile size of $I_x * I_y$ represents CANDLES without the tiled Pixel-first compression. There is an inverse correlation between the tile size of the TP-compression strategy and the PSUM Filter hit rate. Small tiles limit the scattering of partial sums to a small range, thereby ensuring better locality of partial sums. In addition, grouping the pixels before compression can reduce the non-uniformity in the distribution of output neurons further improving the locality of partial sums.

However, reducing the size of the tile leads to fewer non-zero values present in each tile. Reducing the tile size beyond a threshold will result in not having four non-zero activations to feed the cartesian product each cycle catalyzing intra-PE underutilization. Figure 10(c) shows the impact on intra-PE utilization with the variation in tile size. We observe that a 7×4 tile ensures higher PSUM Filter hit rates ($>85\%$) while simultaneously having minimal impact on intra-PE utilization.

Space and Complexity of Loop Tiling: Choosing the ideal tile size is straightforward. While there is an inverse correlation between tile size and PSUM Filter hit rate, extremely small tile sizes lead to intra-PE underutilization. We also observed that the PSUM filter hit rate in each layer has a direct correlation with the number of zeros in activations of that layer. From this, we deduced that tile size has a inverse correlation to the number of zeros in activations. Based on this observation, we define tile size as the ratio of the minimum number of non-zero values required for maximum intra-PE utilization and the fraction of non-zero values in the layer. Since we read four activations each cycle, we need at least four non-zero values per tile to ensure

Figure 11: *(a) Number of non-zero computations normalized to the total number of computations, (b) Mbs of data moved from buffers directly to compute unit (8/24-bit), (c) Mbs of data moved from buffers directly to compute unit (4/8-bit). V: VGG-16, I: Inception, R-A: ResNet-A, M-PW: MobileNet-v1, R-AW: ResNet-AW, M-PW-AW: MobileNet-v1-AW*

maximum intra-PE utilization. We observed that more than 99% of the images have a minimum of 14% non-zero values in each layer. Since we need at least four non-zero values per tile, the minimum tile size is $4/0.14 = 28$. Hence we choose a tile size of 7x4.

Note that a future design can have dynamic tile sizes for each layer depending on the sparsity distribution. For example, a layer with 50% non-zero values can probably get away with $4/0.5 = 8$ entries per tile.

D. Broader Context Discussion

Dense & Quantized Dense Accelerators:
While the metadata is a non-trivial overhead, the benefits from CANDLES far outweigh the cost of additional metadata. By only accessing non-zero operands and performing non-zero computations, CANDLES greatly reduces the amount of compute and data movement overhead compared to a dense accelerator. As shown in Figure 11-a, CANDLES performs as little as 26% of the total dense computations with just sparse activations and up to 20% of the total dense computations with both sparse activations and weights. This has two major benefits. First, skipping the cycles of processing MACs that have zero activations or weights helps improve throughput significantly. Second, in addition to saving the energy consumed in performing MAC, the large share of energy in moving data across the buffer is also significantly reduced.

Figure 11-b,c shows the MBs of data transferred from buffers directly to the MACs to execute a benchmark in both dense and sparse situations. For CANDLES, we also consider the additional index metadata in data movement. At 8/24-bit precision, CANDLES performs up to 4x less data movement. As the metadata size remains unchanged, the MBs of data movement for sparse models increases with reduced precision relative to a dense model. However, this is still less than using a dense model due to the reduction in the number of MACs. CANDLES performs up to 3x less data movement compared to a dense model at 4-bit quantized precision. While a 2-bit quantized dense architecture might further reduce this gap, a dense architecture will likely not match the sparse accelerator on other relevant metrics like throughput and accuracy. While the design space of dense, quantized, and sparse platforms continues to evolve, a sparse platform is proven enough to form the basis for commercial designs like Cerebras, and this work helps advance the state-of-the-art in sparse acceleration.

VI. Related Work

A. Similarities with the Baselines

The CANDLES PE microarchitecture is similar to SCNN given the use of cartesian products, similar total SRAM buffer size, and crossbars to route partial sums. The intra- and inter-PE reduction employed in CANDLES also shares similarities with the two-level PE reduction in SNAP, a Channel-first architecture. CANDLES exceeds the baselines with key changes, including the dataflow, the crossbar, buffer hierarchy and sizes, and work partitions. In addition, other microarchitecture components like the grid network, index-generation logic befitting our metadata format, and PSUM filter are introduced to further improve efficiency.

B. Other Related Work

OuterSPACE [33] is a Pixel-first architecture that uses an outer-product-based matrix multiplication technique with decoupled multiply and merge phases to eliminate redundant memory accesses to non-zero operands. Since PSUMs are not reduced and OuterSPACE uses comparatively large shared caches, the energy consumed is significantly higher. While OuterSPACE claims performance improvement over inner-product-based matrix multiplication due to channel index mismatch (if-condition in Algorithm 2), modern Channel-first architectures easily avoid this by implementing additional index-matching logic. Eyeriss-v2's row-stationary dataflow is another example of a Pixel-first architecture. While row-stationary dataflow performs compression differently from other Pixel-first architectures, Eyeriss-v2 implements an outer product strategy, and similar to SCNN, each activation is reused sequentially with multiple weights resulting in scattering of partial sums to a large 32 entry scratchpad. This results in significant energy consumption to access the partial sums like other Pixel-first architectures.

CANDLES efficiently reduces the partial sums before writing back, thereby reducing access to large buffers.

ExTensor [21] is another Channel-first architecture that finds the intersection of coordinates of non-zero elements. ExTensor uses parallel comparators to find matching intersections. Like other Channel-first architectures, this auxiliary index matching circuit has a non-trivial impact on on-chip power and area. CANDLES avoids this comparator overhead by using pixel-first compression and channel-first dataflow. That being said, the hierarchical elimination of ineffectuals proposed in ExTensor is orthogonal to our contributions and can further improve the benefits offered by CANDLES.

Stitch-X [29] is another Channel-first architecture similar to SNAP that employs a novel dataflow that leverages both spatial and temporal reduction to balance energy efficiency and dataflow control complexity. Bit-Tactical [28] aims to reduce bandwidth and energy costs of memory accesses in sparse DNN accelerators by utilizing a lightweight sparse interconnect, and a novel static scheduling scheme for weights. Cambricon-S [61], PermDNN [14], and Packed Systolic [27] aim to efficiently address the irregularity of sparse neural networks. Scalpel [55] proposes coarse-grained pruning to maintain regularity. Other designs like UCNN [20] exploit sparsity and weight repetition by reusing dot products. Laconic [42], Bit-Pragmatic [3], and Bit-Tactical [28] target bit sparsity in DNN networks by leveraging Booth encoding to elide zeroes. Eyeriss v2 [9] uses a specialized NoC to handle sparsity, but is optimized for small mobile models.

Meanwhile, Sparse ReRAM Engine [53] and SNrram [51] explore ReRAM-based DNN accelerators. While in-memory accelerators [6], [13], [38], [45] provide large benefits with analog logic, exploiting sparsity on them is difficult. Some efforts [62], [37] investigate techniques to accelerate sparse neural networks on GPUs.

VII. Conclusions

State-of-the-art sparse accelerators exhibit inherent trade-offs – Pixel-first architectures require onerous neuron updates while Channel-first architectures require complex indexing logic. We show that this trade-off can be reconciled by adoping a Pixel-first compression and Channel-first dataflow. This approach leads to simple indexing and high temporal locality in neuron updates, which can further be exploited with a 2-level accumulation buffer. We also introduce a work partition strategy that matches the performance of the fastest sparse accelerator (SparTen) without requiring offline analysis. CANDLES achieves low energy for indexing and neuron updates, thus consuming 2.5× to 5.6× lower energy than four state-of-the-art baselines.

VIII. Acknowledgments

We thank the anonymous reviewers for many helpful suggestions. This work was supported in parts by NSF grant CCF 2119677, NSF CAREER grant 1844791, and Google.

References

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

[2] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. Gupta, and H. Esmaeilzadeh, "SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks," in *Proceedings of ISCA*, 2018.

[3] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 382–394.

[4] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. Jerger, and A. Moshovos, "Cnvlutin: Zero-Neuron-Free Deep Convolutional Neural Network Computing," in *Proceedings of ISCA-43*, 2016.

[5] Amazon, "AWS re:Invent: Deliver High Performance ML Inference with AWS Inferentia," 2019, https://www.youtube.com/watch?v=17r1EapAxpk.

[6] A. Ankit, I. Hajj, S. Chalamalasetti, G. Ndu, M. Foltin, R. Williams, P. Faraboschi, W. Hwu, J. Strachan, K. Roy *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of ASPLOS*, 2019.

[7] R. Baghdadi, A. N. Debbagh, K. Abdous, F. Benhamida, A. Renda, J. E. Frankle, M. Carbin, and S. P. Amarasinghe, "TIRAMISU: A polyhedral compiler for dense and sparse deep learning," *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2005.04091

[8] Cerebras, "Cerebras Wafer Scale Engine: An Introduction," 2019, https://www.cerebras.net/wp-content/uploads/2019/08/Cerebras-Wafer-Scale-Engine-Whitepaper.pdf.

[9] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[10] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of ISCA-43*, 2016.

[11] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of MICRO-47*, 2014.

[12] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proceedings of ISCA-43*, 2016.

[13] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 27–39.

[14] C. Deng, S. Liao, Y. Xie, K. Parhi, X. Qian, and B. Yuan, "PermDNN: efficient compressed DNN architecture with permuted diagonal matrices," in *Proceedings of MICRO*, 2018.

[15] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," in *Proceedings of ISCA-45*, 2018.

[16] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks," in *Proceedings of MICRO*, 2019.

[17] Graphcore, "Intelligence Processing Unit," 2017, https://cdn2.hubspot.net/hubfs/729091/NIPS2017/NIPS\%2017\%20-\%20IPU.pdf.

[18] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Wire-Aware Architecture and Dataflow for CNN Accelerators," in *Proceedings of MICRO*, 2019.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[20] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition," in *Proceedings of ISCA*, 2018.

[21] 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," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019.

[22] S. Hooker, A. Courville, Y. Dauphin, and A. Frome, "Selective Brain Damage: Measuring the Disparate Impact of Model Pruning," *arXiv preprint arXiv:1911.05248*, 2019.

[23] 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," *arXiv preprint arXiv:1704.04861*, 2017.

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

[25] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *Proceedings of MICRO-49*, 2016.

[26] D. Kim, J. Ahn, and S. Yoo, "ZeNA: Zero-aware neural network accelerator," *IEEE Design & Test*, 2017.

[27] H. Kung, B. McDanel, and S. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of ASPLOS*, 2019.

[28] A. Lascorz, P. Judd, D. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proceedings of ASPLOS*, 2019.

[29] C.-E. Lee, Y. S. Shao, J.-F. Zhang, A. Parashar, J. Emer, S. W. Keckler, and Z. Zhang, "Stitch-x: An Accelerator Architecture for Exploiting Unstructured Sparsity in Deep Neural Networks," in *SysML Conference*, vol. 120, 2018.

[30] R. Li, Y. Xu, A. Sukumaran-Rajan, A. Rountev, and P. Sadayappan, "Analytical Characterization and Design Space Exploration for Optimization of CNNs," 2021.

[31] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of MICRO-50*, 2017.

[32] NVIDIA, "NVIDIA TESLA V100 GPU Architecture," retrieved 2018, white paper http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[33] 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 *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[34] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. Keckler, and W. Dally, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," 2017.

[35] 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 *Proceeding of HPCA*, 2020.

[36] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive Insertion Policies for High Performance Caching," in *Proceedings of ISCA*, 2007.

[37] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *Proceedings of HPCA*, 2018.

[38] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *Proceedings of ISCA*, 2016.

[39] S.Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of ISCA*, 2016.

[40] S.Han, H. Mao, and W. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding," in *Proceedings of ICLR*, 2016.

[41] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture," in *Proceedings of MICRO*, 2019.

[42] S. Sharify, A. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *Proceedings of ISCA*, 2019.

[43] F. Sijstermans, "The NVIDIA Deep Learning Accelerator," in *Hot Chips*, 2018.

[44] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] L. Song, X. Qian, H. Li, and C. Yiran, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *Proceedings of HPCA*, 2017.

[46] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction Based Execution on Deep Neural Networks," in *Proceedings of ISCA*, 2018.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[48] Tesla, "Tesla Autonomy Day," 2019, https://www.youtube.com/watch?v=Ucp0TTmvqOE.

[49] S. Venkataramani, A. Ranjan, S. Avancha, A. Jagannathan, A. Raghunathan, S. Banerjee, D. Das, A. Durg, D. Nagaraj, B. Kaul, and P. Dubey, "SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks," 2017.

[50] J. Wang, Z. Yuan, R. Liu, H. Yang, and Y. Liu, "An N-way group association architecture and sparse data group association load balancing algorithm for sparse CNN accelerators," in *Proceedings of ASPDAC*, 2019.

[51] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie, "SNrram: an efficient sparse neural network computation architecture based on resistive random-access memory," in *Proceedings of DAC*, 2018.

[52] A. Yang, "Deep Learning Training At Scale: Spring Crest Deep Learning Accelerator (Intel Nervana NNP-T)," in *2019 IEEE Hot Chips 31 Symposium*. IEEE, 2019, pp. 1–20.

[53] T. Yang, H. Cheng, C. Yang, I. Tseng, H. Hu, H. Chang, and H. Li, "Sparse ReRAM engine: joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of ISCA*, 2019.

[54] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[55] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 548–560, 2017.

[56] Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, and H. Yang, "STICKER: An Energy-Efficient Multi-Sparsity Compatible Accelerator for Convolutional Neural Networks in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 55(2), 2020.

[57] Z. Yuan, J. Yue, H. Yang, Z. Wang, J. Li, Y. Yang, Q. Guo, X. Li, M. Chang, H. Yang *et al.*, "Sticker: A 0.41-62.1 TOPS/W 8Bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *2018 IEEE Symposium on VLSI Circuits*, 2018.

[58] J. Zhang, C. Lee, C. Liu, Y. Shao, S. Keckler, and Z. Zhang, "SNAP: A 1.67-21.55 TOPS/W Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference in 16nm CMOS," in *2019 Symposium on VLSI Circuits*, 2019.

[59] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep [-1pt] Neural Network Inference," *IEEE Journal of Solid-State Circuits*, 2020.

[60] 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 *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[61] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proceedings of MICRO*, 2018.

[62] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus," in *Proceedings of MICRO*, 2019.