SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference

Jie-Fang Zhang, *Graduate Student Member, IEEE*, Ching-En Lee, *Student Member, IEEE*, Chester Liu, *Member, IEEE*, Yakun Sophia Shao, *Member, IEEE*, Stephen W. Keckler, *Fellow, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— Recent developments in deep neural network (DNN) pruning introduces data sparsity to enable deep learning applications to run more efficiently on resource- and energy-constrained hardware platforms. However, these sparse models require specialized hardware structures to exploit the sparsity for storage, latency, and efficiency improvements to the full extent. In this work, we present the sparse neural acceleration processor (SNAP) to exploit unstructured sparsity in DNNs. SNAP uses parallel associative search to discover valid weight (W) and input activation (IA) pairs from compressed, unstructured, sparse W and IA data arrays. The associative search allows SNAP to maintain a 75% average compute utilization. SNAP follows a channel-first dataflow and uses a two-level partial sum (psum) reduction dataflow to eliminate access contention at the output buffer and cut the psum writeback traffic by 22× compared with state-of-the-art DNN accelerator designs. SNAP's psum reduction dataflow can be configured in two modes to support general convolution (CONV) layers, pointwise CONV, and fully connected layers. A prototype SNAP chip is implemented in a 16-nm CMOS technology. The 2.3-mm^2 test chip is measured to achieve a peak effectual efficiency of 21.55 TOPS/W (16 b) at 0.55 V and 260 MHz for CONV layers with 10% weight and activation densities. Operating on a pruned ResNet-50 network, the test chip achieves a peak throughput of 90.98 frames/s at 0.80 V and 480 MHz, dissipating 348 mW.**

***Index Terms*— Channel index matching, deep neural network (DNN), energy-efficient accelerator, sparse neural network, unstructured sparsity.**

Manuscript received May 9, 2020; revised September 18, 2020; accepted December 1, 2020. Date of publication December 29, 2020; date of current version January 28, 2021. This article was approved by Associate Editor Edith Beigne. The work of Jie-Fang Zhang, Ching-En Lee, Chester Liu, and Zhengya Zhang was supported in part by the Defense Advanced Research Projects Agency (DARPA) Common Heterogeneous Integration and IP Reuse Strategies (CHIPS), in part by DARPA Circuit Realization at Faster Timescales (CRAFT), and in part by the Toyota Research Institute (TRI). *(Corresponding author: Jie-Fang Zhang.)*

Jie-Fang Zhang, Chester Liu, and Zhengya Zhang are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: jfzhang@umich.edu; cwhliu@umich.edu; zhengya@umich.edu).

Ching-En Lee was with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: lchingen@umich.edu).

Yakun Sophia Shao was with Nvidia Research, Santa Clara, CA 95051 USA. She is now with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: ysshao@berkeley.edu).

Stephen W. Keckler is with NVIDIA, Austin, TX 78717 USA (e-mail: skeckler@nvidia.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2020.3043870.

Digital Object Identifier 10.1109/JSSC.2020.3043870

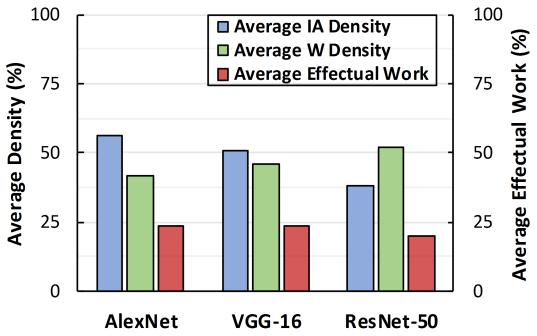

Fig. 1. Average density of input IA and W data and average effectual work of common DNNs after network pruning.

I. Introduction

DEEP learning or more specifically, deep neural network (DNN), has emerged to be a key approach to solving complex cognition and learning problems [1], [2]. State-of-the-art DNNs [3]–[9] require billions of operations and hundreds of megabytes to store activations and weights. Given the trend toward even larger and deeper networks, the compute and storage requirements will prohibit real-time, low-power deployment on platforms that are resource and energy constrained. The compute and storage challenges motivated efforts in network pruning to zero out a large number of weights (W) in a DNN model with only little inference accuracy degradation [10]–[12]. In addition to sparsity in weights, the commonly used rectifier linear unit (ReLU) clamps all negative activations to zeros, resulting in sparsity in output activations (OAs), which becomes input activations (IAs) of the next layer.

Fig. 1 shows that the typical density of nonzero W (after network pruning [10]) and IA (due to ReLU) in well-known network models: AlexNet, VGG-16, and ResNet-50. An average of 50% density is common. Because the nonzero W and IA are nearly randomly distributed, the amount of effectual computation, i.e., computation that does not involve a zero, is only 25%. If a small sacrifice in inference accuracy can be tolerated, the density of operands and the effectual computation can be further reduced.

Data sparsity can be exploited to save power. Many DNN accelerators, e.g., Eyeriss [13], gates the computation, e.g., by turning off the clock, whenever a zero in the IA is detected in runtime. Most dense DNN accelerators can incorporate

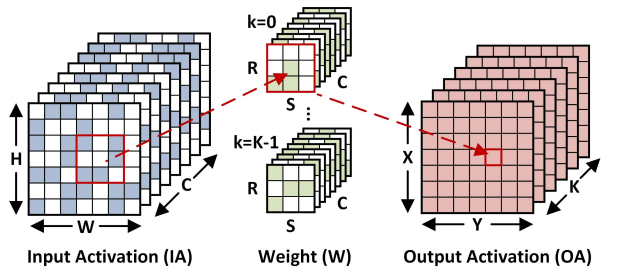

Fig. 2. Convolution computation between unstructured sparse IA and W in a sparse DNN. The colored cells indicate nonzero entries, and the white cells indicate zero entries.

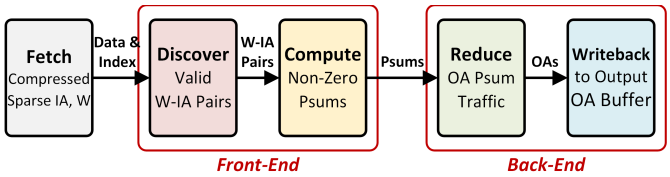

Fig. 3. Processing pipeline of a sparse DNN processor.

this technique to reduce power, but it does not shorten the latency or improve the throughput.

Cnvlutin [14] and Cambricon-X [15] are well-known early architectures that exploit sparsity in compressed IA for latency reduction and throughput improvement. However, they were designed to work with the sparsity in one of the two operands, W or IA, but not both. A dense processing architecture can be easily adapted to support one-operand sparsity by indirect data access.

To fully exploit sparsity in both operands, W and IA are stored in a compressed form where nonzero elements are represented by value–index pairs. Storage in a compressed form can reduce the memory size and bandwidth. However, unlike the common dense array and matrix storage, compressed storage is not amenable to regular and efficient vector processing. One approach is to decompress the compressed form before processing, but decompression costs performance, memory, and power. Instead, state-of-the-art sparse DNN accelerators [16]–[20] process data directly in the compressed form, offering both low memory bandwidth and high degree of acceleration.

Fig. 2 shows sparse convolutions, and Fig. 3 shows the high-level computation pipeline of DNN processing in the compressed format (which will be referred to as sparse DNN processing for simplicity). A sparse DNN processor loads Ws and IAs in a compressed form consisting of a data array (zeros removed) and an index array. Compressed W and IA arrays are paired by matching indices, dispatched to a multiplier array, and the resulting partial sums (psums) are accumulated to their respective OAs in output buffers.

Data sparsity leads to better performance and efficiency, but major challenges remain as follows.

1) *Front-End Challenge:* Multiplier under-utilization due to an insufficient number of W-IA pairs that can be extracted and dispatched to the multiplier array.
2) *Back-End Challenge:* Data traffic and access contention to support the accumulation of psums whose destination addresses are seemingly random.
3) *Flexibility Challenge:* Limited support for different kernel sizes and layer types.

State-of-the-art sparse DNN accelerators, including EIE [16], SCNN [17], Sticker [18], [19], and Eyeriss v2 [20], addressed some of the challenges in sparse DNN processing, but did not solve all of them. EIE [16] exploits both W and IA sparsity but is restricted to fully connected (FC) layers. SCNN [17] is the first attempt at exploiting both W and IA sparsity for convolution (CONV) layers. It maximizes multiplier utilization at the cost of massive psum writeback traffic and access contention, and it supports only CONV layers. Sticker [18], [19] follows SCNN's dataflow and uses two-way set-associative processing elements (PEs) to alleviate the access contention but requires offline preprocessing to re-arrange IA data. Without the data re-arrangement, the access contention remains as significant as in SCNN. Eyeriss-v2 [20] employs a two-step search front end to find effectual W-IA pairs by first fetching nonzero IAs and then using the channel index of the IA to look for nonzero Ws. Eyeriss-v2 adopts an Eyeriss-like row stationary dataflow [13] to avoid memory access contention.

We present sparse neural acceleration processor (SNAP) [21] that adopts a channel-first dataflow and is optimized for the efficient processing of unstructured sparse DNNs. To solve the front-end challenge, SNAP uses parallel associative index matching (AIM) units and sequence decoders to extract a sufficient number of W–IA pairs to maintain a high multiplier array utilization of 75%. To solve the back-end challenge, SNAP adopts a two-level psum reduction that consists of PE-level and core-level reduction to eliminate memory access contention and reduce psum writeback traffic to an average of 2.79 OA reductions/cycle for a core of 63 multipliers. The core-level reduction is configurable to support common layers in vision-based DNN models, including general $R \times S$ CONV ($R, S > 1$), pointwise CONV, and FC.

The rest of this article is organized as follows. Section II introduces the channel-last dataflow, an approach adopted by state-of-the-art sparse accelerators, and analyzes its advantages and inefficiencies in processing sparse DNNs. Section III presents our channel-first dataflow and quantitatively compares it against the channel-last dataflow to demonstrate its advantages. In Section IV, we describe our solution to the front-end challenge using parallel index matching, and in Section V, we describe our solution to the back-end challenge using two-level psum reduction. The configurable core-level reduction makes the SNAP architecture flexible to support different kernel sizes and layer types. Section VI presents the overall SNAP architecture, followed by measurement and evaluation results using both synthetic sparse workloads and commonly used pruned networks. Finally, Section VII concludes this article.

II. Background and Related Work

A dense CONV operation can be described by (1), where f represents the activation function. For simplicity, the bias is ignored and IA padding is assumed to be zero. FC can be

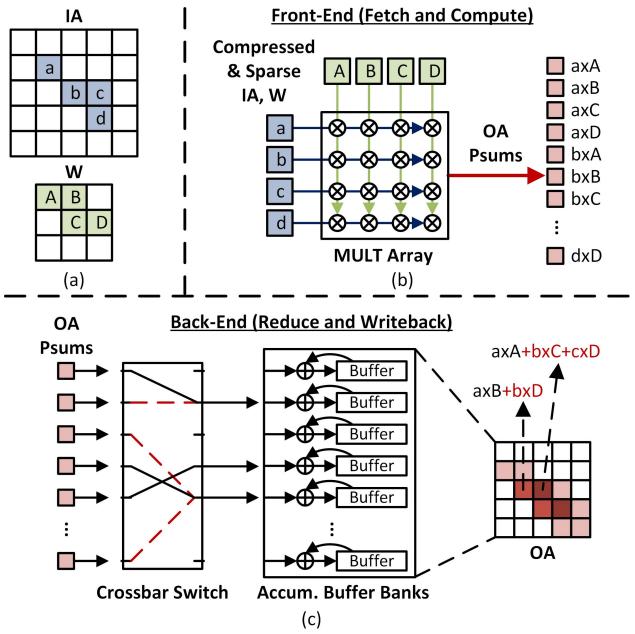

Fig. 4. Illustration of channel-last dataflow for sparse DNN processing: (a) IA and W data in dense format; (b) front-end dataflow; and (c) back-end dataflow.

viewed as a special case of CONV. In this work, we will use the following indexing convention: a W index of (r, s, c, k) corresponds to (row, column, channel, kernel), an IA index of (h, w, c) corresponds to (row, column, channel), and an OA index of (x, y, k) corresponds to (row, column, output channel (kernel)):

$$\mathrm{OA}_{(x,y,k)} = f\left(\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{c=0}^{C-1}\mathrm{IA}_{(x+i,y+j,c)} \times W_{(i,j,c,k)}\right). \quad (1)$$

As shown in (1), there are two main steps in computing CONV (besides the activation function): 1) IAs and Ws are multiplied to produce psums and 2) the psums are accumulated (or reduced) along the channel dimension (C) and along the pixel dimension (R, S). The channel-dimension reduction and pixel-dimension reduction are commutative. The channel indices of a W and an IA need to match for the two to be multiplied together.

In ordering inputs for storage and processing, we can choose either the pixel dimension first ((r, s) for W and (h, w) for IA) followed by the channel dimension (c), termed channel-last dataflow, or vice versa, termed channel-first dataflow. Both SCNN [17] and Sticker [19] adopt the channel-last dataflow.

A. Channel-Last Dataflow for Sparse DNN Processing

The channel-last dataflow is shown in Fig. 4. In the channel-last dataflow, the nonzero W and IA data are ordered in the pixel dimension first for storage and processing. Because data are ordered pixel dimension first, as W and IA data are streamed in, their channel indices are aligned. Since any nonzero W can be multiplied with any nonzero IA of the same channel in an CONV operation, the W and IA can be freely paired to produce a large number of W-IA pairs for a multiplier array.

The simple W-IA pairing results in a simple front end for the channel-last dataflow. As shown in Fig. 4(a) and (b), the compressed W and IA data of size n can be broadcast over an $n \times n$ 2-D multiplier array vertically and horizontally, respectively, so that each W is multiplied to every IA. The drawback of the channel-last dataflow is that the address of the OA that a psum needs to be accumulated to (will be referred to as the psum address) does not follow deterministic ordering. According to (1), if a nonzero W and a nonzero IA have matching channel index c, they can be multiplied to produce a psum with an index of $(x, y, k) = (h-r, w-s, k)$. The 3-D index is then translated to a 1-D physical address. Hence, the $n \times n$ 2-D multiplier array produces n^2 psums whose (x, y) indices are $\{(h-r, w-s)\}$ with random drawings of $h \in \{0, \ldots, H-1\}$, $r \in \{0, \ldots, R-1\}$, $w \in \{0, \ldots, W-1\}$, and $s \in \{0, \ldots, S-1\}$. It is highly likely to have two or more psums that share the same address, and in theory, they should be accumulated, or reduced, before writeback. However, it is challenging to organize psums and reduce them before writeback. Without any psum reduction, the writeback traffic becomes congested, and frequent contentions are possible at the OA buffer. It requires complex hardware or wiring, e.g., a crossbar switch, to resolve the contention, and results in pipeline stalls and low multiplier array utilization.

This back-end challenge is shown in Fig. 4(c). The psums need to be distributed by a switch to the corresponding buffer bank. The red lines indicate the psum writebacks that lead to buffer contentions. To avoid contentions, conflicting psums need to be held. In the example, one output requires the accumulation of three psums, resulting in a three-cycle writeback where the multiplier array stalls for two cycles.

B. Other Related Work

Numerous DNN accelerators have been proposed to exploit the parallelism in DNN inference operations [13]–[28]. To leverage the sparsity in Ws and IAs, some work implemented power-saving techniques by clock-gating a PE when a zero IA is detected [13], [25] to increase energy efficiency. Some work exploited sparsity at the bit level [26] by skipping the computation for the zero-valued bits in the bit-serial multiplication. Compared with earlier methods, exploiting sparsity in bit level reduces the overall computation cycles and increases both efficiency and throughput. However, the dataflows are still similar to traditional dense accelerators, where zero elements are fetched on-chip, incurring unnecessary data transfers.

In this article, we focus on the DNN inference accelerator design that exploits sparsity at data level and operates in the compressed form. Only nonzero elements are fetched on-chip for computation, such as in [16], [17], [19], and [20]. This type of accelerator skips all unnecessary data transfers and computation to optimize energy efficiency and computation throughput. We will refer to this type of accelerator as a sparse accelerator for discussion and comparison.

III. Channel-First Processing Dataflow

Compared with the channel-last dataflow, the channel-first dataflow orders W and IA data across the channel dimension

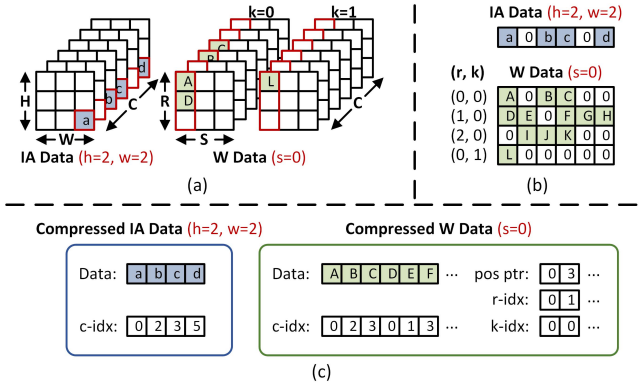

Fig. 5. Channel-first compression in SNAP: (a) IA and W data in dense format; (b) uncompressed IA and W bundles; and (c) compressed IA and W bundles.

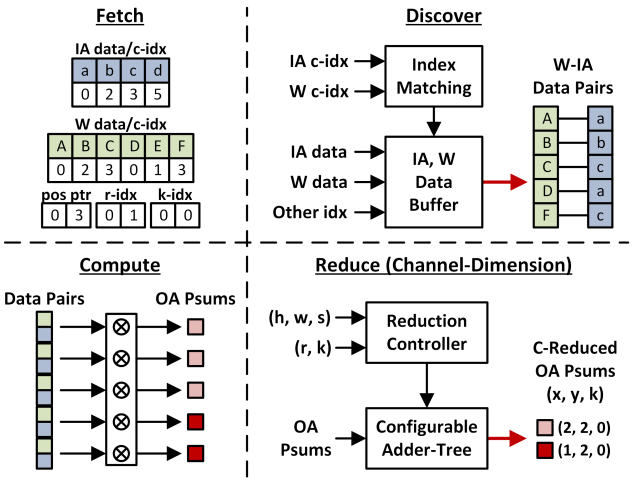

Fig. 6. SNAP's channel-first dataflow with channel index matching and psum reduction along the channel dimension.

first for storage and processing. The channel-first dataflow allows the psums computed by the multiplier array to be locally reduced before writeback.

A. Compression Format

In the channel-first dataflow, nonzero W and IA data are ordered and processed in the channel dimension first and then in the pixel dimension. An example of a 3×3 CONV is shown in Fig. 5. The channel-first dataflow supports arbitrary IA data size and 3×3 IA data are chosen for the ease of illustration.

In Fig. 5(a), the W and IA data are illustrated in the dense form. In our channel-last dataflow, a bundle of nonzero W data and a bundle of nonzero IA data are provided to a PE at a time. An IA bundle contains data in $(h, w, \underline{c})$, where $\underline{c}$ represents all nonzero elements along the channel dimension, and a W bundle is larger and it contains data in $(\underline{r}, s, \underline{c}, \underline{k})$, where $\underline{r}$, $\underline{c}$, and $\underline{k}$ represent all nonzero elements along the row, channel, and kernel dimension, respectively. In general, a larger data bundle leads to a higher utilization and processing efficiency, but we limit the IA bundle not to span more than 1-pixel location to simplify channel index matching.

Fig. 5(b) shows the IA bundle of $(h, w, \underline{c}) = (2, 2, \underline{c})$ and the W bundle of $(\underline{r}, s, \underline{c}, \underline{k}) = (\underline{r}, 0, \underline{c}, \underline{k})$. The bundles are stored in the compressed form with all zeros squeezed out, as shown in Fig. 5(c). The IA data are stored channel first. The IA storage consists of a data array that stores nonzero IA data and a channel index (c-idx) array that stores the channel index of the corresponding IA data. The W data are also stored channel first, followed by row (r-idx) and kernel (k-idx). The W storage consists of a data array, a c-idx array, an r-idx array, and a k-idx array, as well as a position pointer (pos-ptr) array to track the starting points in the data array of the next r-idx and/or k-idx. For instance, the pos-ptr array stores 0 and 3, indicating that the first three data values A, B, and C have $(r, k) = (0, 0)$, and the data values D, E, and F have $(r, k) = (1, 0)$. An IA bundle and a W bundle are sent to one PE for processing.

B. Channel-First Dataflow

In a channel-first dataflow, the nonzero W and IA data are streamed in channel first, the addresses of the psums computed are aligned, and the psums can be immediately reduced along the channel dimension. Despite the appeal of the channel-first dataflow, a W and an IA can only be paired and multiplied if their channel index matches. Hence, channel index matching must be performed at the front end to extract the valid W-IA pairs. Compared with the channel-last dataflow, this additional channel index matching step introduces an overhead, but it provides immediately reducible psums to cut the writeback traffic, leading to potential improvements in both power and performance.

The channel-first dataflow is shown in Fig. 6. Each PE receives a W bundle $(\underline{r}, s, \underline{c}, \underline{k})$ and an IA bundle $(h, w, \underline{c})$, as shown in Fig. 5(c). The W c-idx is matched with the IA c-idx to generate valid W-IA pairs. Valid W-IA data pairs are fetched and multiplied to produce psums. The psums are accumulated and saved to the OAs at $(x, y, k) = (h - r, w - s, k)$. Due to the channel-first input ordering and bundled processing, the address of the psums computed by one PE will stay the same until the PE completes an IA bundle and switches to a new IA bundle (change of h, w) or until the r-idx or k-idx changes (change of r and k) for a W bundle. Due to the high locality of psum address, the majority of the psums are immediately reduced to one within a PE. To process a complete convolution, the IA and W data are decomposed and compressed into IA and W bundles. The IA and W bundles are distributed to different PEs for processing as shown in Fig. 6 until all bundles are fully processed.

In a channel-first dataflow, channel-dimension psum reduction is done first. A second-level pixel-dimension psum reduction can be done on-chip to further reduce the writeback bandwidth. To enable the second-level psum reduction, PEs can be arrayed and coordinated to facilitate the psum reduction across the PEs. The second-level psum reduction will be described in more detail in Section V-B.

To evaluate the benefits of the channel-first dataflow, we prototyped a channel-last dataflow that follows the processing pipeline described in Fig. 4 and quantify the key differences between the two dataflows, as shown in Fig. 7.

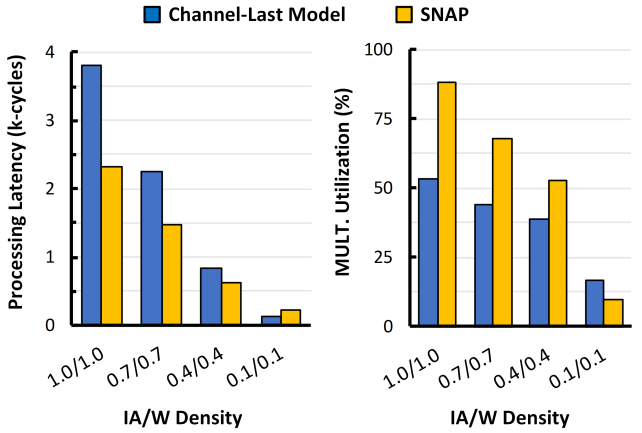

Fig. 7. Comparison between channel-last dataflow and SNAP (channel-first dataflow) for dense, medium, and sparse workloads.

The channel-last dataflow is limited by the large number of OA buffer accesses and the access contention, causing the compute to stall and worsening both utilization and processing latency, and in comparison, the channel-first dataflow employs a front-end channel index matching to reduce the number of OA buffer accesses and remove the access contention. The channel-first dataflow provides on average 1.51× and 1.45× improvement over the channel-last dataflow in processing latency and multiplier utilization, respectively. Only when the data density drops below a threshold, e.g., 10% W and IA data density, does the channel-first dataflow starts to underperform the channel-last dataflow due to two factors: 1) the lack of reduction opportunities due to highly sparse data and 2) the imbalance of input bundle sizes causing PE workload imbalance.

The SNAP architecture follows the channel-first dataflow. The two key techniques, channel index matching and two-level psum reduction, will be presented in Sections IV and V.

IV. Channel Index Matching

Channel index matching extracts pairs of nonzero W-IA pairs of matching channel index. We propose an AIM unit to extract a sufficient number of W-IA pairs to sustain a high utilization of a parallel multiplier array. The AIM performs index matching and encodes the addresses of valid W-IA pairs, and a sequence decoder decodes the addresses and dispatches the pairs for parallel computation.

A. Associative Index Matching

Fig. 8(a) shows the microarchitecture of the AIM and illustrates its mechanism. The AIM consists of an $N \times N$ comparator array, where each row is connected to a priority encoder. During operation, an AIM receives the W and IA channel index arrays from a PE, and it compares each W channel index to each IA channel index. In Fig. 8(a), for example, the W channel indices 0, 2, 9, and 5 are matched to the IA channel index at position 0, 1, 4, and 2, respectively, whereas the W channel index 4 does not have a match. A priority encoder encodes the match result in each row into a valid bit to indicate a match and the matched position in the IA channel index array. Upon completion, an AIM returns a list of valid-position pairs to a PE for processing.

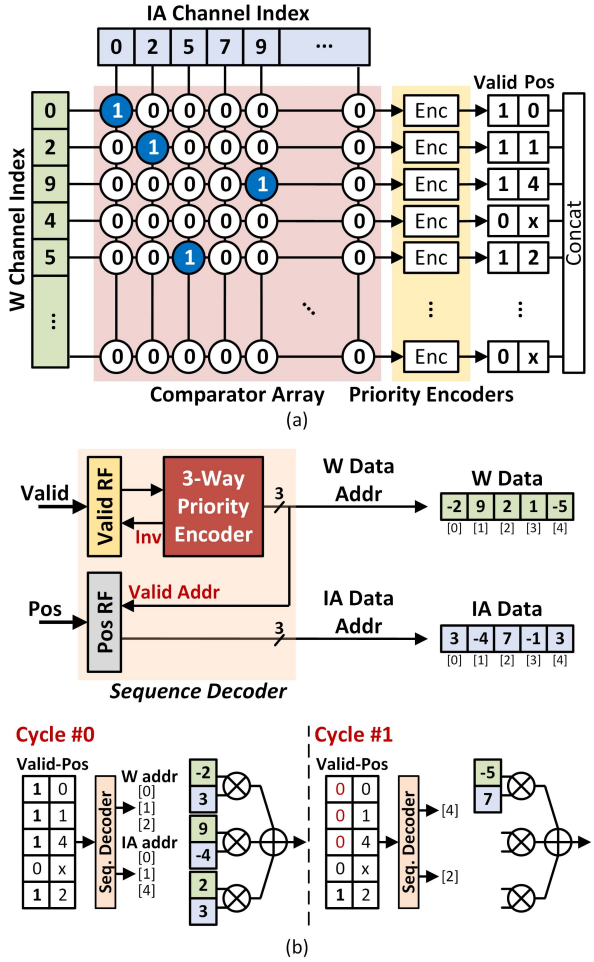

Fig. 8. (a) Microarchitecture of AIM. (b) Microarchitecture of sequence decoder and step-by-step example of W-IA data pair dispatch in a PE.

B. Sequence Decoder

Within a PE, a sequence decoder converts a list of valid-position pairs into W-IA data pairs. Fig. 8(b) shows the microarchitecture and illustrates the sequence detection mechanism with the valid-position list output from Fig. 8(a): 1) a three-way priority encoder converts three valid-position pairs (with valid bit = 1) at a time into W-IA data addresses; 2) the positions in the valid-position pairs are used as the addresses to fetch IA data and the indices of the valid-position pairs are used as the addresses to fetch W data; and 3) the three valid-position pairs are invalidated by overwriting the valid bits to 0, and the W and IA data are sent to the multipliers to compute psums. After completing the list of valid-position pairs, the PE requests a new list from the AIM.

C. Design Tradeoff Exploration

The size of the comparator array, N, determines AIM's throughput and effectiveness. N needs to be sufficiently large

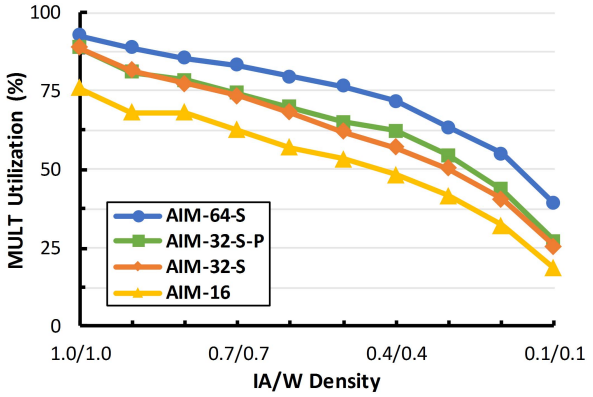

Fig. 9. Multiplier utilization of AIM designs at different data density levels.

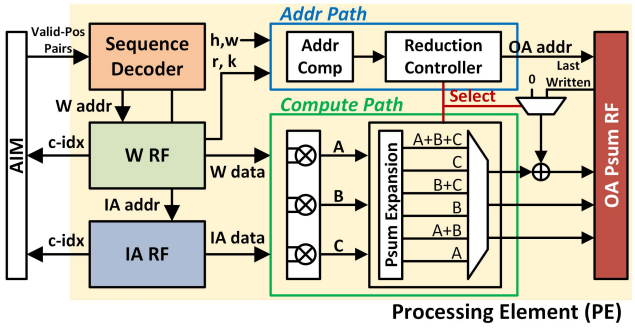

Fig. 10. PE microarchitecture and psum reduction along the channel dimension.

to even out the workload density imbalance and variations across W and IA bundles for extracting enough W-IA pairs to maintain a high multiplier utilization. However, the area and power consumption of AIM increase nearly quadratically with N. To balance these two competing factors, we avoid using a small AIM, e.g., one with a 4×4 or 8×8 comparator array, and instead use a larger AIM and time multiplex it between multiple PEs to amortize the cost.

Fig. 9 shows the multiplier utilization across a range of workload densities for $N = 16$, 32, or 64, where a suffix S indicates that a large AIM is time-multiplexed among an appropriate number of PEs and a suffix P indicates that a simple prefetch mechanism is implemented to further reduce the workload imbalance by pre-requesting valid-position pairs from the AIM. A larger AIM provides a higher multiplier utilization across all workload densities: the utilization improves by 10% from $N = 16$ to 32 and again from 32 to 64. A large AIM with $N = 64$ incurs an area overhead of 50%. A moderate-sized AIM with $N = 32$ cuts the area overhead below 12.5%, and adding prefetching increases the utilization by up to 5%. Therefore, in designing SNAP, we adopt a time-multiplexing, prefetching AIM design with an $N = 32$ comparator array to balance the performance, area, and power consumption. This design achieves an average multiplier utilization above 75% for our benchmarks.

V. Two-Level Partial Sum Reduction

To reduce the output bandwidth, after the psums are computed by the PEs, they should be maximally accumulated on chip to reduce the number of writebacks to the output buffer. Following the channel-first dataflow, SNAP implements a two-level psum reduction to minimize the read-accumulate-writes to the output buffer. The psums are first reduced along the channel dimension inside the PEs and then along the pixel dimension across PEs. The across-PE reduction is configurable to support not only CONV but also pointwise CONV and FC.

A. PE-Level Channel-Dimension Reduction

The PE microarchitecture is shown in Fig. 10. Each PE contains a compute path consisting of three multipliers and psum accumulators, an address path that computes the addresses and then selects the psum reduction pattern, a sequence decoder, a W register file (RF), and an IA RF to provide the inputs, and an OA psum RF to store the outputs. In each cycle, the sequence decoder dispatches three W-IA data pairs and their indices ((h, w) for IA data and (r, s, k) for W data). The W-IA data pairs are directed to the compute path to produce the psums, A, B, and C; and the W and IA indices are sent to the address path to compute the addresses (recall psum index is $(x, y, k) = (h-r, w-s, k)$, which is translated to a physical 1-D address). Given the three psum addresses, the reduction controller selects one psum reduction pattern in the compute path.

TABLE I
Selection of Reduction Pattern

OA addresses	Out[0]	Out[1]	Out[2]
$Addr_A = Addr_B = Addr_C$	A+B+C	-	-
$Addr_A = Addr_B < Addr_C$	A+B	C	-
$Addr_A < Addr_B = Addr_C$	A	B+C	-
$Addr_A < Addr_B < Addr_C$	A	B	C

The channel-first input processing order guarantees that the addresses for A, B, and C are ordered and non-decreasing, i.e., $\text{Addr}_A \leq \text{Addr}_B \leq \text{Addr}_C$. Due to the deep channels seen in modern DNN layers, in most cases, $\text{Addr}_A = \text{Addr}_B = \text{Addr}_C$, and the three psums can be accumulated and reduced to one. If not, the reduction controller selects one of the reduction patterns according to Table I. One, two, or three psums are produced and sent to the OA psum RF every cycle, avoiding stalls in the computation pipeline. A PE retains an OA psum in the RF until all possible reductions along the channel dimension are complete, cutting the psum writeback traffic by up to the channel depth compared to the channel-last dataflow. Note that the number of multipliers in a PE is set to 3 to keep a reasonable overhead for the reduction pattern selection.

B. Core-Level Pixel-Dimension Reduction

To reduce the writeback traffic, after the first-level psum reduction along the channel dimension, a second-level psum reduction across an array of PEs can be done. Fig. 11(a) shows a 3×3 PE array and the input data mapping for a 3×3 CONV kernel. The three W bundles of $s = 0$, 1, 2 are

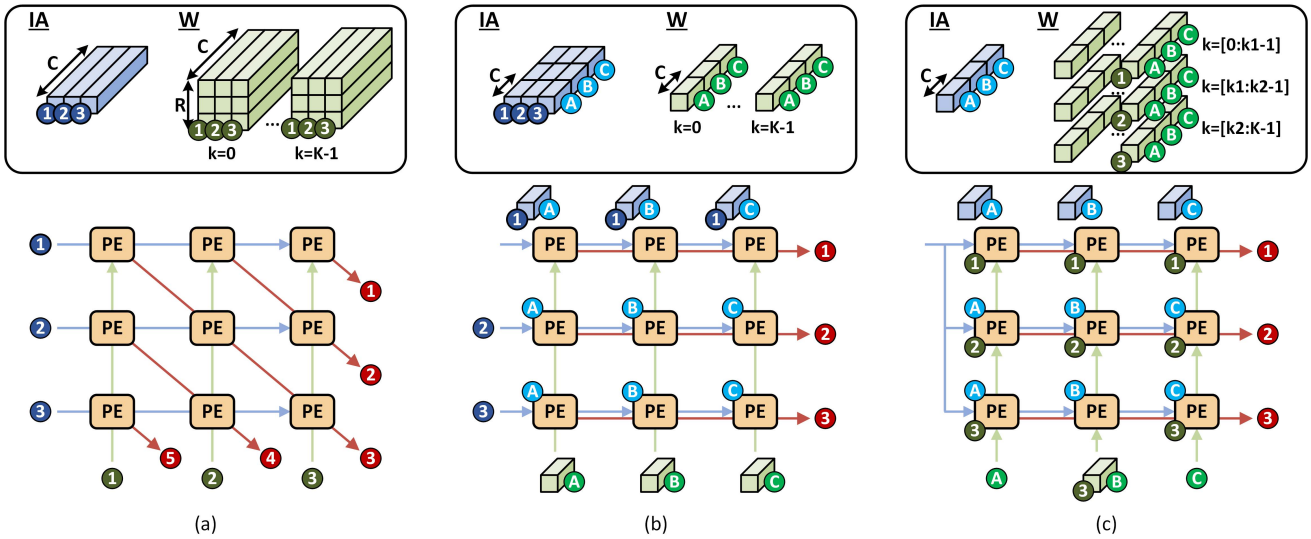

Fig. 11. Psum reduction configuration across an array of PEs. (a) 3×3 CONV in diagonal mode. (b) Pointwise CONV in row mode. (c) FC in row mode.

broadcast to the three PE columns, and the three IA bundles of $(h, w) = (2, 2)$, $(2, 3)$, $(2, 4)$ are broadcast to the three PE rows. This mapping allows W and IA reuse by a column and a row of PEs, respectively. Following this mapping, the PEs on the same diagonal lane produce psums of the same address. For example, assume $(r, k) = (0, 0)$ for the W bundles, PE_{11}, PE_{22}, and PE_{33} compute the psums going to the OA address of $(x, y, k) = (h - r, w - s, k) = (2, 2, 0)$; similarly, PE_{12} and PE_{23} compute the psums going to the OA address of $(x, y, k) = (h - r, w - s, k) = (2, 1, 0)$. Therefore, the psums along the diagonal lanes are accumulated.

We name an array of PEs a core. To support the popular 3×3 CONV seen in modern DNN models, the number of columns can be set to 3 (which also sets the number of PEs along a diagonal to 3) to achieve the full utilization. The number of rows can be set based on the throughput requirement for an application. Going beyond the 3×3 CONV, the diagonal mode reduction is used for general $R \times S$ CONV $(R, S > 1)$. If $R, S > 3$, the CONV kernel is divided into $R \times 3$ sub-kernels and then distributed and processed independently on multiple compute cores. A global accumulator merges the psums from the multiple cores to compute the final OA for writeback.

The core-level psum reduction along the pixel dimension cuts the writeback traffic by 2.3–3.0×. The two-level reduction resolves the access contention seen in prior work [17]–[19]. It reduces the writeback traffic to only 2.79 OAs per cycle on average for a core of 7×3 PEs that contain a total of 63 multipliers. The output bandwidth of a channel-first dataflow using the two-level reduction is 22× lower than the channel-last dataflow with an equal number of multipliers.

C. Support for Pointwise CONV and FC

In our work, we considered the pointwise CONV and FC as special cases of the CONV computation shown in Fig. 2 with size constraints $R = S = 1$ and $R = S = H = W = 1$, respectively. The size constraints in pointwise CONV and FC eliminate the possibility of pixel-dimension reduction. The interconnection between the PEs and the core reducer is configurable to support not only the diagonal mode, but also provide a row mode to support DNN layers that do not have any pixel-dimension reduction opportunities.

To reuse the same architecture for a pointwise CONV including the same output bandwidth and to achieve a high utilization, the inputs are divided into groups in the channel dimension, and the core is reconfigured to perform reduction along the channel dimension. For example, in Fig. 11(b), a W bundle is divided into three groups in the channel dimension, and each group is broadcast to a column of PEs; an IA bundle is divided into three groups in the channel dimension and multicast to the three PEs in a row. This mapping allows W reuse by the PEs along a column. Following this mapping, the PEs on the same row produce the psums going to the same OA address. Therefore, the core reducer is configured to accumulate the psums from the PEs along the rows.

In processing an FC, W data cannot be reused in batch-1 processing. Similar to the pointwise CONV, the PE array is utilized in channel-dimension reduction. An IA bundle is divided into groups in the channel dimension; a W bundle is divided into groups in both the channel and kernel dimensions. For example, in Fig. 11(b), a W bundle is divided into three groups in the channel and kernel dimension and multicast to the three PEs in a column; an IA bundle is divided into three groups in the channel dimension and broadcast to the three PEs in a column. This mapping allows IA reuse by the PEs along a column. Similar to the pointwise CONV, the PEs on the same row produce the psums going to the same OA address, and the core reducer is configured to accumulate the psums from the PEs along the rows.

VI. Implementation and Evaluation Results

We present the SNAP system architecture based on the techniques introduced earlier. The SNAP architecture is prototyped in a 16-nm test chip. The chip is measured and evaluated using

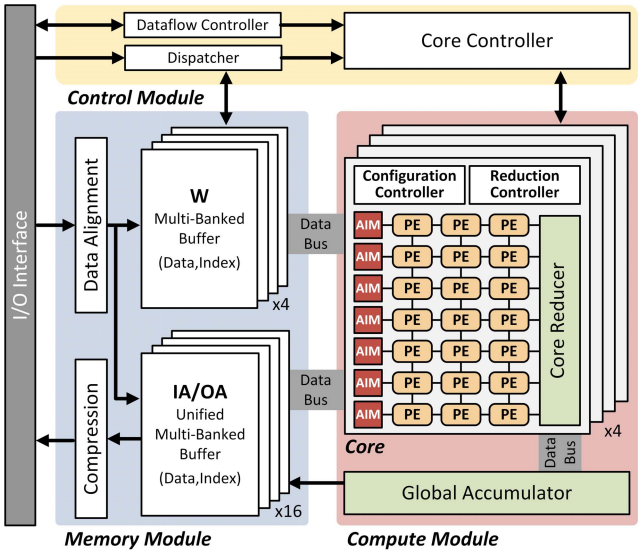

Fig. 12. SNAP system architecture.

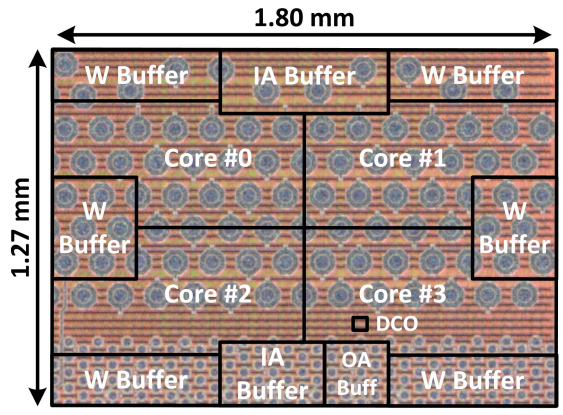

Fig. 13. Microphotograph of the 16-nm SNAP test chip.

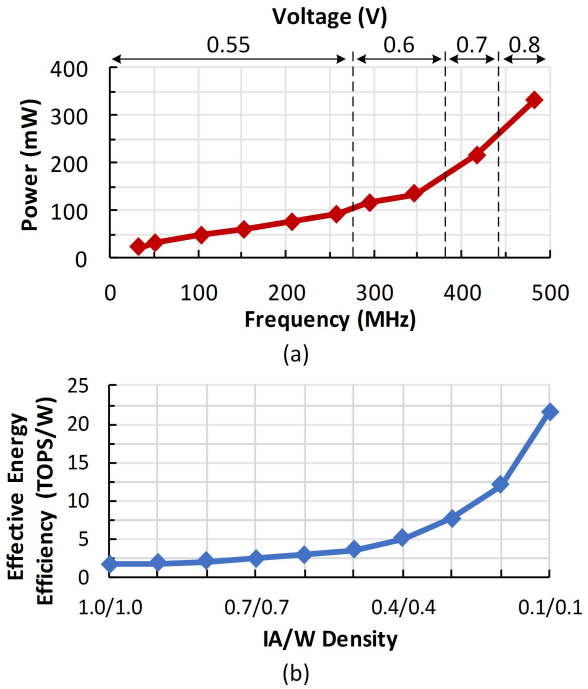

Fig. 14. (a) Measured power consumption at different operating frequencies and the optimal supply voltages. (b) Measured effectual energy efficiency for synthetic sparse workloads at different data density levels.

workloads of different sparsity levels and a pruned ResNet-50 model. The results are summarized and compared with state-of-the-art dense and sparse DNN accelerators.

A. SNAP Architecture Overview

The SNAP high-level architecture is shown in Fig. 12. It consists of multiple cores, a control module, and a memory module. The control module provides the configuration of the compute cores and coordinates the communication with the external interface. The memory module is composed of multi-banked IA and OA buffers shared between the compute cores and non-shared W buffers of each compute core. The compressed W and IA data are fetched from off-chip and aligned by bundles. W bundles are stored in each core's W buffers following the system configuration, and IA bundles are stored in the IA buffers. The output OAs are compressed before writeback to the external memory.

Following the high-level architecture, we designed a SNAP test chip that is made of four cores, and each core is implemented in a 7×3 PE array. Within a core, seven AIM units are shared in a time-multiplexed fashion between the three PEs in a row. Each PE is implemented with three multipliers and a sequence decoder. The PEs output psums, which are accumulated by the core reducer, and a global accumulator is used to further accumulate psums before the final writeback.

The SNAP test chip provides a total of 252 multipliers of 16-bit fixed-point precision and a total of 280.6-kB SRAM. Note that an 8-bit design would work equally as well to demonstrate the architectural advantages and show an even better performance and energy efficiency. The only difference is that the overhead of index matching, measured as a fraction of the compute core, increases from 12.5% (in a 16-bit design) to 17% (in an 8-bit design). The test chip is implemented using a 16-nm CMOS process technology with a core area of 2.3 mm^2. Fig. 13 shows the chip microphotograph.

B. Performance Analysis

In our evaluations, a 16-bit fixed-point multiply and accumulate (MAC) is counted as two operations (OPs). Each PE computes at most three MACs or six OPs every clock cycle. For the evaluations, we used both synthetic sparse workloads and real workloads of pruned DNN models, including AlexNet, VGG-16, and ResNet-50, pruned with less than 0.5% accuracy loss using the technique from [10], to measure performance, power consumption, and effectual energy efficiency (considering the input IA/W density).

The measured chip power consumption is shown in Fig. 14(a). At each operating frequency, the power is reported at the lowest supply voltage. At 0.55 V and 260 MHz, the test chip achieves the peak effectual energy efficiency

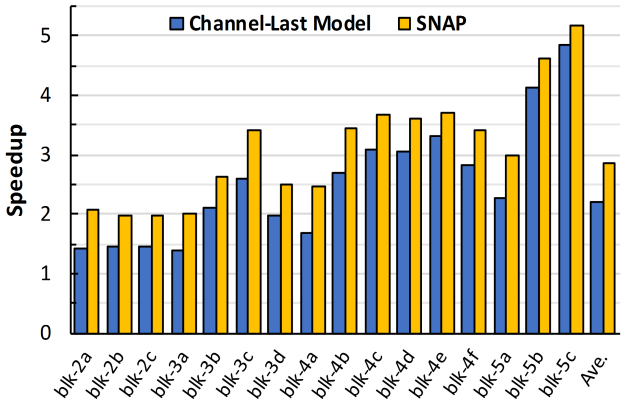

Fig. 15. Processing speedup by SNAP (a channel-first dataflow) and a channel-last dataflow over a dense accelerator baseline running the residual blocks of a sparse ResNet-50 model.

of 3.61 TOPS/W for a sparse ResNet-50 model with an average IA/W density of 0.38/0.52. The chip achieves the highest inference throughput of 90.98 frames/s for the same sparse ResNet-50 model at 0.8 V, 480 MHz, consuming 348 mW.

The measured effectual energy efficiency of the SNAP test chip running synthetic sparse workloads of different data density levels is shown in Fig. 14(b). In previous work, SCNN [17] reports the results at the IA/W density of 0.2/0.2 and 0.1/0.1, and Sticker [19] reports the results at the IA/W density of 0.15/0.15 and 0.05/0.05. We used the IA/W density of 0.1/0.1 to approximately match the previous work, so we can fairly compare the designs for energy efficiency. For dense (1.0/1.0), medium (0.4/0.4), and extremely sparse (0.1/0.1) IA/W data density levels, a peak effectual energy efficiency of 1.67, 5.06, and 21.55 TOPS/W is achieved, respectively.

We benchmark the measured throughput of the SNAP architecture and a simulated channel-last dataflow architecture over a dense accelerator baseline running a sparse ResNet-50 model. In Fig. 15, we show the speedup in processing each residual block. For a fair comparison, the dense accelerator baseline is constructed to be the same as SNAP, but with uncompressed, dense inputs. Overall, the channel-last dataflow demonstrates an average speedup of 2.20× over the dense baseline; and the SNAP design obtains an average speedup of 2.87× over the dense baseline, which is 30.3% better over the channel-last dataflow.

Although SNAP, a channel-first dataflow, generally provides a higher performance than a channel-last dataflow, the performance gap is more visible at the shallow and middle layers, but less visible for deep layers, as shown in Fig. 15. For instance, SNAP has a 46.2% better performance than the channel-last dataflow for the first residual block blk-2a, but only 7% better performance for the last residual block blk-5c. The difference is attributed to two factors: 1) the deep layers are generally sparser after pruning, resulting in less access contention and better performance for the channel-last dataflow, and 2) extremely sparse workloads often come with imbalanced zero distribution, causing challenges to SNAP's index matching and psum reduction.

The imbalanced zero distribution, or workload imbalance, is a design challenge for both the SNAP front end and back end. At the front end, the workload imbalance causes PEs to receive a varied number of W-IA pairs for computation, and the overall performance is limited by the PE with the heaviest workload. This problem may be resolved by implementing a more aggressive prefetching scheme that provides a larger work chunk for PEs to prevent them from stalling. At the back end, the workload imbalance causes each PE to have less channel-dimension reduction opportunity before writeback to the output buffer, resulting in a higher writeback bandwidth. In addition, the PEs on the same core-level reduction lane may have varied reduction progress, requiring the faster PEs to be stalled to wait for the slowest one. The back-end problem may be mitigated by implementing a larger OA psum RF to store more psums in a PE and the OA output buffers using two-port SRAMs or multi-banked SRAMs to provide a higher accumulation bandwidth.

C. Comparison Against State-of-the-Art Works

Compared with state-of-the-art inference accelerators that support sparsity at data level or for power saving shown in Table II, SNAP exploits sparsity in both compressed W and IA data for both CONV and FC layers. SNAP employs a 16-bit fixed-point precision for data storage and computation. SNAP has a comparable number of multipliers as many previous silicon designs. Energy efficiency is reported for the effective efficiency evaluated on both synthetic workloads and real sparse network workloads.

To provide a fair comparison, we also present the synthesis results of SNAP-65nm-8b, the same SNAP design with 8b storage and computation in a 65-nm GP process. SNAP-65 nm-8b was synthesized at 250 MHz with the SRAM modules re-generated to support 8-b processing. It is estimated to have an area of 9.32 mm^2 and consume 500 mW. For a sparse AlexNet, SNAP-65nm-8b is estimated to achieve an effective energy efficiency of 0.74 TOPS/W running at 250 MHz at a nominal voltage of 1.0 V.

1) Dense and Sparse Workloads: For high- and medium-density workloads, SNAP achieves a higher effectual energy efficiency than all the other sparse accelerators. More specifically, the channel-last accelerators, including SCNN and Sticker, suffer from memory contention and compute stalls, resulting in lower performance. SNAP also benefits from the large channel-dimension reduction opportunity and a low workload imbalance to achieve a high compute utilization and better efficiency. The network-on-chip (NoC) in Eyeriss-v2 consumes extra power and latency for conventional DNN workloads. Compared with Eyeriss-v2, SNAP benefits from a larger search depth and is specifically optimized for CONV ($R \times S$ and 1×1) and FC layers. For extremely sparse workloads, Sticker [19] reports better effectual energy efficiency than SNAP but uses 8bit storage and computation, while SNAP uses 16-bit storage and computation.

In sparse workload evaluations, SNAP outperforms all the other works. Compared with sparse accelerators,

TABLE II
COMPARISON WITH PRIOR WORKS

	This Work	Sticker [19]	Eyeriss-v2 [20]	SCNN [17]	Envision [25]	Eyeriss-v1 [13]
Sparsity Support	Data-Level	Data-Level	Data-Level	Data-Level	Power-Saving	Power-Saving
Layer Support	CONV+FC	CONV+FC	CONV+FC	CONV	CONV+FC	CONV
Technology	16nm	65nm	65nm	16nm	28nm	65nm
Silicon	Yes	Yes	No	No	Yes	Yes
Core Area	2.3mm^2¶	7.8mm^2	2695k gates (NAND-2)	7.9mm^2	1.87mm^2	12.25mm^2
Num. of MULTs	252	256	384	1024	1024/512/256	168
Data Width	16b	8b	8b	16b	4/8/16b	16b
On-Chip SRAM	280.6KB	170KB	246 KB	1200KB	144KB	108KB
Supply Voltage	0.55–0.80V	0.67–1.0V	N/A	N/A	0.55–1.1V	0.82–1.17V
Frequency	33–480MHz	20–200MHz	200 MHz	1000MHz	50–200MHz	100–250MHz
Power	16.3–364mW	20.5–248.4mW	N/A	N/A	7.5–300mW	235–332mW
Peak Effectual Efficiency† (TOPS/W)	D: 1.67 (16b) M: 5.06 (16b) S: 21.55 (16b)	D: 0.5 (8b) M: 2.5 (8b) S: 30 (8b)	D: 0.25 (8b) S: N/A	N/A	0.53 (16b)	0.31 (16b)
Sparse AlexNet‡ (TOPS/W)	3.86 (16b)	2.82 (8b)	0.96 (8b)	N/A	0.8-3.8§	0.17 (16b)
Sparse VGG-16‡ (TOPS/W)	3.79 (16b)	N/A	N/A	N/A	2.0§	0.09 (16b)
Sparse ResNet-50‡ (TOPS/W)	3.61 (16b)	N/A	N/A	N/A	N/A	N/A

*One multiply-accumulate (MAC) computation is counted as 2 operations (OPs). Data width of the OPs is labeled in bracket.
†Peak effectual energy efficiency in TOPS/W evaluated on synthetic sparse workloads at 1.0/1.0 (D), 0.4/0.4 (M), and 0.1/0.1 (S) IA/W density levels.
‡Effectual energy efficiency in TOPS/W for sparse networks pruned within 0.5% accuracy loss.
§Data width not specified. ¶Post-shrinkage area. (pre-shrinkage area: 2.4 mm^2 [21])

Sticker (8b) [19] and Eyeriss-v2 (8b) [20], SNAP achieves up to 3.34× and 6.68× better effectual energy efficiency running synthetic workloads, respectively. For sparse AlexNet, SNAP achieves 1.37× and 4.02× better effectual energy efficiency than Sticker (8b) and Eyeriss-v2 (8b), respectively.

2) Index Matching Search Depth: Among all the references, Eyeriss-v2 is the closest to SNAP in terms of architectural design. The difference between SNAP and Eyeriss-v2 is mainly attributed to two factors. First is the difference in search granularity in index matching. SNAP adopts a more coarse-grained compression format. During the discover stage, the AIM searches in the first 32 nonzero entries of the compressed W data. On the other hand, Eyeriss-v2 uses the CSC format. During the discover stage, Eyeriss-v2 first identifies IA's channel index and then searches for nonzero W data only across all k indices. Due to a smaller search depth, Eyeriss-v2's index matching mechanism is more likely to encounter fragmentation of nonzero W data, resulting in an insufficient number of W-IA pairs sent to the MAC array and possibly a lower compute utilization. The second difference is that SNAP's dataflow is specifically optimized for CONV and FC, whereas Eyeriss-v2 implements a fully flexible NoC for routing and switching. The NoC may be a burden on performance and power consumption.

3) Load Balancing Consideration: Load imbalance can be caused by front-end workload distribution or back-end writeback or both. The channel-last accelerators, such as Sticker and SCNN, are not affected by the front-end workload imbalance seen in channel-first accelerators, such as SNAP and Eyeriss-v2. The varying number of W-IA pairs assigned to different PEs affects the compute utilization of each PE. With a large enough search depth, SNAP, a channel-first accelerator, can minimize the front-end workload imbalance. Eyeriss-v2, another channel-first accelerator, used a similar approach, except that its search depth is smaller than SNAP.

The channel-last accelerators suffer from the back-end workload imbalance when there are memory access contentions between the accumulation buffers and the multipliers, especially for medium-to-high density workloads. The memory contention affects the number of cycles needed to complete a chunk of W and IA data and causes progress differences between PEs, resulting in the faster PEs stalling for the slowest one. This imbalance situation is worse for higher density workloads where contention is more frequent. A channel-first accelerator such as SNAP effectively mitigates the back-end workload imbalance by streamlined, maximal OA psum reduction before writeback.

VII. CONCLUSION

We present the SNAP that exploits unstructured sparsity in sparse DNNs for efficient inference acceleration. SNAP adopts a new channel-first dataflow with channel index matching as

the front end and a two-level psum reduction back end for sparse DNN processing.

At the front end, channel index matching is done by efficient AIM units and sequence decoders to discover valid W-IA pairs for computation. It results in an average compute utilization of 75% while limiting the area overhead to only 12.5% of the compute core. At the back end, the combination of PE-level psum reduction along the channel dimension and core-level psum reduction along the pixel dimension eliminates write-back access contention at the output buffer and reduces the psum writeback traffic by 22× compared with the previous sparse accelerator designs. The core-level psum reduction is configurable to support general $R \times S$ CONV, pointwise CONV, and FC layers.

A SNAP test chip is designed using 252 16-bit multipliers organized in four cores of 7×3 PEs. The chip is measured to achieve an effective energy efficiency of up to 21.55 TOPS/W running synthetic sparse workloads and 3.61 TOPS/W running a pruned ResNet-50. Compared with the state-of-the-art dense and sparse accelerators, SNAP offers competitive performance and energy efficiency by maintaining high compute utilization and low writeback data traffic.

ACKNOWLEDGMENT

The authors would like to thank the DARPA CRAFT program for providing chip fabrication. They would also like to thank C. Fu and T. Wesley for their assistance with the chip design and A. Parashar, J. Emer, and W. J. Dally from Nvidia Research for their advice.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

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

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 580–587.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[10] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[11] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[12] T. Zhang *et al.*, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 184–199.

[13] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[14] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[15] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2016, pp. 1–12.

[16] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[17] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27–40, 2017.

[18] Z. Yuan *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 *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 33–34.

[19] Z. Yuan *et al.*, "STICKER: An energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, Feb. 2020.

[20] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[21] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67–21.55TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16nm CMOS," in *Proc. Symp. VLSI Circuits (VLSI)*, Jun. 2019, pp. C306–C307.

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

[23] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[24] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan./Feb. 2016, pp. 262–263.

[25] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[26] J. Albericio *et al.*, "Bit-pragmatic deep neural network computing," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2017, pp. 382–394.

[27] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

[28] C.-E. Lee *et al.*, "Stitch-X: An accelerator architecture for exploiting unstructured sparsity in deep neural networks," in *Proc. Syst. Mach. Learn. (SysML)*, 2018.

Jie-Fang Zhang (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2015, and the M.S. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 2018, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

His current research interests include energy-efficient hardware architecture and accelerator design for machine learning, computer vision, and robotics applications.

Ching-En (Alex) Lee (Student Member, IEEE) received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2012, the M.S. degree in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2015, and the Ph.D. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2020.

From 2015 to 2016, he was with Intel Labs, Hillsboro, OR, USA, where he worked on real-time machine learning hardware acceleration for full-duplex radios. From 2018 to 2019, he was with Iluvatar Corex, San Jose, CA, USA, where he led the development of deep learning inference systems on chip (SoCs) and systems for edge computing applications. He is currently an Associate with McKinsey & Company, Shanghai, China. His past research interests focus on efficient domain-specific computing architectures and full-stack systems design for machine learning, deep learning, computer vision, and robotics.

Chester Liu (Member, IEEE) received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2008, and the M.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2010. He is currently pursuing the Ph.D. degree in electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA.

From 2010 to 2013, he was with MediaTek, Hsinchu, where he worked on the platform design and verification for smartphone SoCs. He has been with the University of Michigan since 2014. His current research interests include neuromorphic computing, efficient hardware accelerator design for machine learning and robotics, and 2.5-D integration technologies.

Yakun Sophia Shao (Member, IEEE) received the B.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2009, and the S.M. and Ph.D. degrees in computer science from Harvard University, Cambridge, MA, USA, in 2014 and 2016, respectively.

She was a Senior Research Scientist with NVIDIA, Inc., Santa Clara, CA, USA. She is currently an Assistant Professor with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA. Her research interest is in the area of computer architecture, with a special focus on specialized accelerators, heterogeneous architecture, and agile VLSI design methodology.

Stephen W. Keckler (Fellow, IEEE) received the B.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1990, and the M.S. and Ph.D. degrees in computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1992 and 1998, respectively.

From 1998 to 2012, he was a tenure-track Professor with The University of Texas at Austin (UT-Austin), Austin, TX, USA, where he developed scalable parallel processor and memory system architectures, including nonuniform cache architectures; explicit data graph execution processors, which merge dataflow execution with sequential memory semantics; and micro-interconnection networks to implement distributed processor protocols. All of these technologies were demonstrated in the TRIPS experimental computer system. In 2010, he joined NVIDIA, Austin, and is currently the Vice President of Architecture Research, where he focuses on architectures for massively parallel and energy-efficient systems, domain-specific accelerators, and memory systems.

Dr. Keckler is a fellow of the Association for Computing Machinery (ACM) and an Alfred P. Sloan Research Fellow. He was a recipient of the NSF CAREER Award, the ACM Grace Murray Hopper Award, the President's Associates Teaching Excellence Award at UT-Austin, and the Edith and Peter O'Donnell Award for Engineering.

Zhengya Zhang (Senior Member, IEEE) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley (UC Berkeley), Berkeley, CA, USA, in 2005 and 2009, respectively.

He has been a Faculty Member with the University of Michigan, Ann Arbor, MI, USA, since 2009, where he is currently an Associate Professor with the Department of Electrical Engineering and Computer Science. His research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing.

Dr. Zhang was a recipient of the David J. Sakrison Memorial Prize from UC Berkeley in 2009, the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Award in 2013, and the University of Michigan College of Engineering Neil Van Eenam Memorial Award in 2019. He has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS since 2015. He has been serving on the Technical Program Committee of the Symposium on VLSI Circuits and the IEEE Custom Integrated Circuits Conference (CICC) since 2018. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS from 2013 to 2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS from 2014 to 2015.