# SPRITE: Sparsity-Aware Neural Processing Unit with Constant Probability of Index-Matching

Sungju Ryu, Youngtaek Oh, Taesu Kim, Daehyun Ahn, and Jae-Joon Kim
Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea
{sungju.ryu, youngtaek.oh, taesukim, daehyun.ahn, jaejoon}@postech.ac.kr

***Abstract*—Sparse neural networks are widely used for memory savings. However, irregular indices of non-zero input activations and weights tend to degrade the overall system performance. This paper presents a scheme to maintain constant probability of index-matching for weight and input over a wide range of sparsity overcoming a critical limitation in previous works. A sparsity-aware neural processing unit based on the proposed scheme improves the system performance up to 6.1× compared to previous sparse convolutional neural network hardware accelerators.**

***Index Terms*—Sparse neural network, convolutional neural network, hardware accelerator, index-matching.**

## I. Introduction

Sparse convolutional neural networks have been widely adopted to many neural applications thanks to their small memory footprint. However, the effective throughput of MAC units is substantially degraded when the pruned sparse weight matrices are computed on the conventional hardware optimized for dense matrices, because the sparse weight matrices consist of a large number of zero values thereby having most of the MAC units perform meaningless *multiplication-by-zero* operations.

Several hardware accelerators [1]–[3] have been presented to efficiently deal with such sparse networks. In the hardware for sparse networks, index-matching logic is typically required to match non-zero input activation and weight pairs which participate in the effective multiplications. During the index-matching process, the utilization of the multipliers is largely decreased or many stalls occur in the memory path depending on the sparsity, resulting in serious performance degradation.

Based on the observation, this paper introduces SPRITE, a high-performance sparse CNN hardware accelerator. The key feature of the proposed architecture is the index-matching logic which has the constant matching probability over a wide range of the input/weight sparsity.

## II. Preliminary: Previous Sparse CNN Hardware Accelerators

SCNN [1] proposed a concept to make all the input activations and weights participate in the effective multiplications by using a cartesian-product based approach (Fig. 1a). In the SCNN architecture, a scatter network is required to send partial-sums (psums) from multipliers to an accumulation buffer. During psum communications between multipliers and accumulation buffers, writeback traffic congestion frequently occurs in the scatter network, thereby degrading the utilization of the multipliers.

SNAP [2] and SparTen [3] first perform channel-dimension reduction in processing elements (PEs) by using an inner-product based approach (Fig. 1b), and then perform pixel-dimension reduction in(SparTen)/outside(SNAP) PEs. Performing the channel-dimension reduction first reduces sparse tensor multiplications to (almost) dense matrix multiplications in pixel dimension so that psum writeback traffic congestion can be substantially reduced compared to SCNN case. However, such an approach experiences difficulty in handling a wide range of input/weight sparsity. When computing the sparse layers (Fig. 1b left), an index-matching unit matches a small number of the input activation and weight pairs, so the number of matched indices is insufficient to fully utilize the multipliers. Therefore, the scheme fetches as many input activations and weights as possible into PEs to maximize the utilization of multipliers for computations of sparse layers. On the other hand, such an approach puts a burden on a small number of multipliers to process an excessive number of matched input/weight pairs over multiple clock cycles when relatively dense layers are computed (Fig. 1b right). In this case, a large number of data stalls may occur at PEs and system performance can be significantly hurt. Fig. 1b illustrates a performance degradation when such an inner-product based approach is used. Because of the trade-off between the data stalls and the under-utilization of multipliers depending on the input/weight sparsity, system performance cannot be maximized for a wide range of sparsity. Therefore, there is a pressing need to develop a sparsity-aware

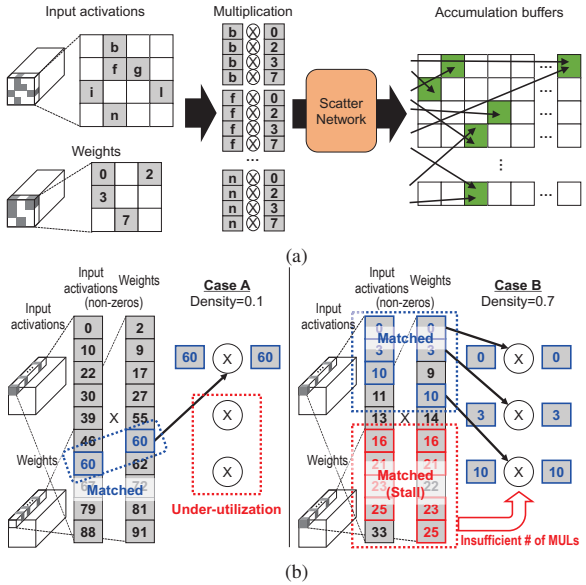


Fig. 1. Previous hardware acceleration methods for sparse convolutional neural networks. (a) Cartesian-product based approach and (b) inner-product based approach.

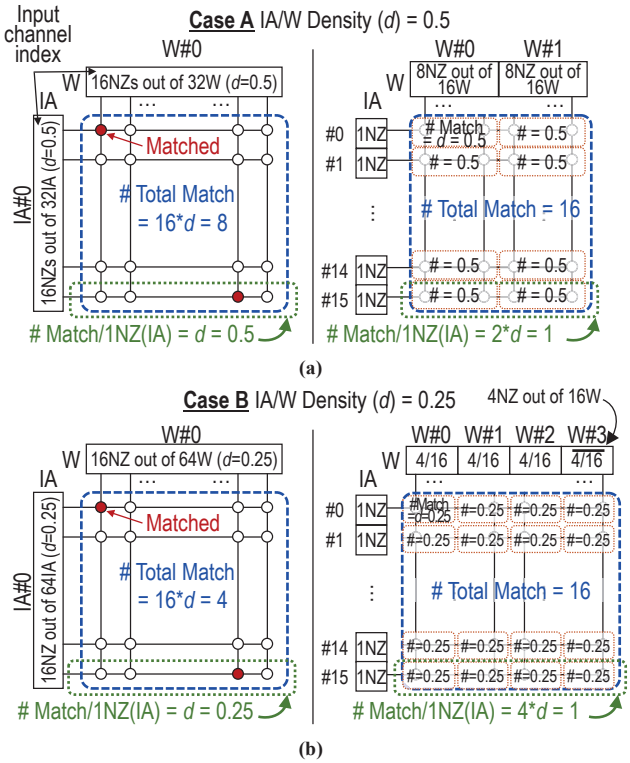


Fig. 2. Conventional inner-product based index-matching method (left) vs. proposed index-matching method (right) for various densities. (a) density=0.5 and (b) density=0.25. NZ: Non-zero values. IA: Input activations. W: Weights. IA#(S): S-th spatial domain. W#(OC): OC-th output channel.

processing unit to maintain the high performance over a wide range of input/weight sparsity in neural network layers.

## III. Proposed Architecture

### A. Index-Matching with Constant Probability

Fig. 2 illustrates the differences of matching probabilities between the previous and proposed index-matching methods depending on input/weight densities. `Case A` shows an example of density ($d$) = 0.5. The density is defined as the number of non-zero elements divided by the total number of elements in a set. In this example, an index-matching unit (dashed blue box) consists of 16×16 comparators, and each of the comparators indicates whether an index of the input activation is the same as an index of the paired weight. For both index-matching methods, we assume that 16 input channel indices for non-zero input activations and 16 input channel indices for non-zero weights are fetched at a time. In the conventional inner-product based approach (Fig. 2a left), 16 indices of non-zero input activations/weights are picked from 32 original dense indices of input channels ([31:0]) on average when density = 0.5. For the 16 non-zeros out of 32 weights, the average total number of matched pairs with corresponding inputs becomes 8 because half of the 16 non-zero inputs share the same indices with the non-zero weights statistically when the indices of inputs and weights are selected from the same index pool with $d$=0.5 (dashed blue box). In this case, we can observe that the average number of matched input/weight pairs becomes only the half of the maximum number which the matching unit can produce.

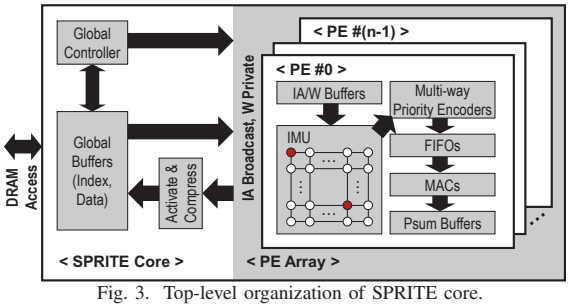


Fig. 3. Top-level organization of SPRITE core.

A main idea in the proposed index-matching method is to increase the number of buckets from which weight and input indices in the same range can be picked. At the same time, the number of inputs/weights in a bucket is reduced to maintain the same number of total non-zero values for a matching unit. In that case, the number of total matched pairs will increase for a given density because a weight (input) index can be compared with input (weight) indices that are selected from multiple buckets. For an example with the density($d$) = 0.5, the proposed index-matching method (Fig. 2a right) simultaneously handles input activations from 16 spatial domains and weights from two output channels unlike the previous inner-product based approach which used inputs from a spatial domain and weights from an output channel. The previous method receives 16 input activations and 16 weights in an input channel direction. In contrast, the proposed scheme receives 16 input activations from 16 different spatial domains. In terms of weights, the proposed scheme moves to the next output channel after receiving 8 consecutive non-zeros in an input channel direction. The average number of the matched pairs of input and weight from a spatial domain (eg. IA#0) and an output channel (eg. W#0) becomes 0.5(=$d$, brown dotted box) as the probability for an output channel (W#0) to have a weight which has the same index with the input activation from IA#0 is equal to density($d$). Because two output channels of weight for each input activation are handled simultaneously, the average number of matched pairs for an input activation is 1(=2×$d$). Therefore, the total number of matched pairs becomes 16 (blue dashed box) in the proposed scheme which is 2× larger than the previous design.

The difference in matching probability between the previous and proposed schemes becomes more prominent as the density becomes lower. When density = 0.25, the previous inner-product based approach (Fig. 2b left) still holds non-zeros in a spatial domain and an output channel only. Hence, the total number of matched pairs (blue dashed box) is decreased compared to `Case A`. On the other hand, the proposed approach (Fig. 2b right) deals with 4 output channels of weights when density=0.25. The number of matched pairs for a spatial domain of input activations (IA#(S)) and an output channel of weights (W#(OC)) is decreased by 4× when density is reduced by half (brown dotted box). However, *the total number of matched pairs (blue dashed box) is maintained regardless of densities* in the proposed scheme by dealing with more output channels.

### B. SPRITE Architecture

*1) **Top-Level Architecture**:* Fig. 3 shows a top-level organization of the proposed SPRITE core. The SPRITE core

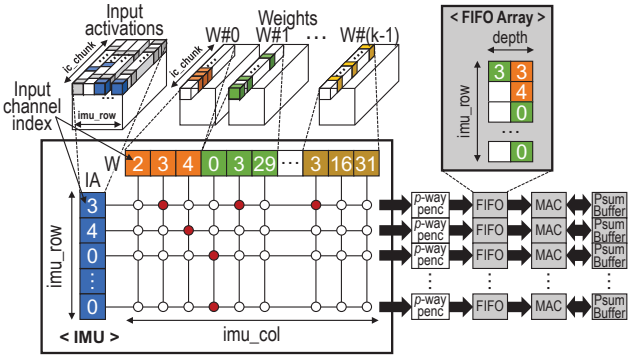


Fig. 4. Implementation of PE in SPRITE core.

consists of a global controller, global buffers, a compression logic, and a PE array. Global buffers are comprised of an activation buffer and a weight buffer. An activation buffer is shared by PEs, and a weight buffer is dedicated to each PE. Input activations, weights, and their indices are loaded from the global buffers to the PE array. In each PE, an index-matching unit (IMU) compares the input activation and the weight pairs. Matched pairs are multiplied and sent to psum buffers. Psums are accumulated in the psum buffers until the complete weighted sum is computed. The complete psums are sent out of the PE array to be activated and compressed.

*2) **Processing Elements**:* A processing element (PE) in our SPRITE core has an IMU, multi-way priority encoders, a FIFO array, MAC units, and psum buffers (Fig. 4).

***Index-Matching Unit:*** The IMU consists of index buffers and a comparator array. Each buffer in the IMU holds input channel indices of non-zero values. A buffer for input activations consists of $imu\_row$ indices. Each row of the buffer is dedicated to a spatial domain in the input feature-map. On the other hand, a weight index buffer stores $imu\_col$ indices. First, consecutive non-zero weights in the input channel direction are loaded to the weight buffer. If the input channel index is equal to or greater than $ic\_chunk$, we move to the following output channel instead of continuously loading the data in the input channel direction. A comparator array has $imu\_row \times imu\_col$ comparators, and each comparator produces a matching signal for an input activation and a weight pair when they have the same input channel indices (a circle filled with a dark red color). In the IMU, input activations are reused by weights, and weights are reused by input activations.

***Handling Fluctuation in the Number of Matches:*** From $imu\_row \times imu\_col$ comparators in our IMU, $imu\_row$ input activation and weight pairs are matched on average in a clock cycle. However, the number of matched pairs from the $imu\_row \times imu\_col$ comparator array is not always $imu\_row$, and it slightly varies every clock cycle. Hence, SPRITE adopted multi-way priority encoders to maximize the average number of matched pairs close to the ideal upper limit. The multi-way priority encoder sequentially finds multiple 1's from the output of the comparator array. Considering the trade-off between the average number of matched pairs and critical-path delay depending on the parameter $p$, we used $p$=3 in our design. A multi-way priority encoder is required for each $1 \times imu\_col$ comparator array, and hence $imu\_row$ multi-way ($p$-way) priority encoders are required for an $imu\_row \times imu\_col$ comparator array. When multiple matched pairs exist for an IMU row, sometimes even multi-way priority encoders cannot digest the entire ($imu\_col$) range of the weight buffer in a cycle. In such a *rare* case, unused indices of weights are compared in the next clock cycle.

TABLE I
DETAILED PARAMETERS AND AREA/POWER BREAKDOWN OF SPRITE ARCHITECTURE BY USING A 28NM CMOS TECHNOLOGY. DOUBLE-BUFFERING IS APPLIED TO ALL THE BUFFERS. BIT-PRECISION CONFIGURATION AND CAPACITY OF PSUM BUFFERS ARE THE SAME AS SCNN [1]. GLB INDICATES GLOBAL BUFFERS.

| PE | Size / Parameters | Area [$\mu m^2$] | Power [mW] |
|---|---|---|---|
| MACs | 7 MACs, IA/W=16b, Psum=24b | 7448 | 3.27 |
| IMU | imu_row=7, imu_col=ic_chunk=32 | 4879 | 1.69 |
| Priority encoders | 3-Way | 2162 | 0.32 |
| FIFOs | Depth=6 | 8109 | 2.20 |
| Psum buffers | 2.63KB (0.38KB/MAC) | 29813 | 1.79 |
| Total (PE) | - | 52412 | 9.27 |

| Top Level | Size / Parameters | Area [$mm^2$] | Power [mW] |
|---|---|---|---|
| PEs | 32 PEs | 1.68 | 297 |
| GLB (IA+W) | 1691KB | 1.81 | 283 |
| Total (SPRITE) | 32 PEs (224 MACs, 1775KB SRAM) | 3.48 | 580 |

***Back-End of a PE:*** After the indices of the matched pairs are extracted from the multi-way priority encoders, the pairs are sent to the index FIFOs. While 1 MAC and 1 psum buffer is assigned for each IMU row, multiple index pairs can be matched at an IMU row in a cycle albeit with low probabilities. Then, contention in the writeback stage can occur thereby leading to the stall in the front-end stages. To mitigate such a writeback contention issue, the index FIFOs temporarily store the matched pairs, and sequentially send a matched pair to the multiplier in the following stage every clock cycle. Then, the multiplication results are accumulated in the psum buffers. Each psum buffer is dedicated to each IMU row. The fully accumulated psums are sent to the activation and compression logic outside the PE.

***Dataflow:*** In the next clock cycle, the weight index buffer replaces currently stored weight indices with the following weight indices located in the next input channels and/or output channels. When streaming of the weights is complete, the input activation buffer loads the following indices of the input activations in the next input channels.

## IV. EVALUATION

### A. Methodology

We synthesized the accelerators in a 28nm CMOS technology. For a fair comparison, the same number of multipliers and comparators (Table I) were used for each design when needed. In addition, same bit-precision and operating clock frequency (400MHz) were used for all designs. When reproducing the SNAP, we did not apply a time-interleaving scheme and placed a comparator array on each PE to maximize the performance of SNAP architecture, which was also done for the SPRITE case. Original SNAP used a *large* number (32×32) of comparators

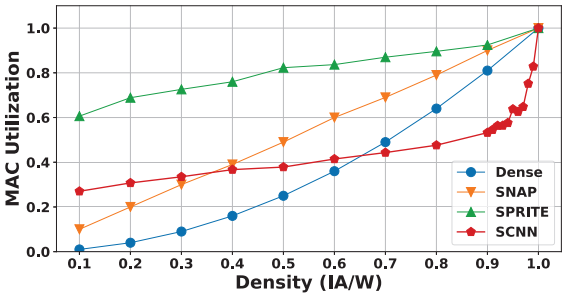


Fig. 5. Comparison of MAC utilization of SPRITE with previous sparse CNN accelerators.

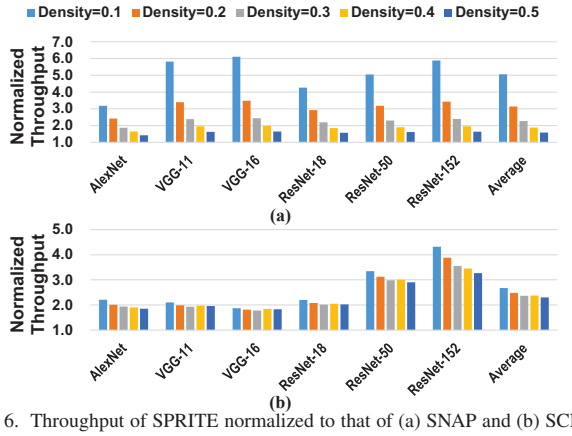


Fig. 6. Throughput of SPRITE normalized to that of (a) SNAP and (b) SCNN.

with the time-interleaving scheme and just a *small* number of (3) multipliers per PE thereby maintaining high utilization (explained in Section II), but the number of the comparators and multipliers is set to be same as those of SPRITE for a fair comparison in this study.

### B. Experimental Results

***MAC Utilization:*** We first compare the MAC utilization of the SPRITE with that of previous sparse CNN accelerators (Fig. 5). It can be observed that SPRITE shows better utilization than other accelerators for a wide range of densities of the input activations and weights thanks to the proposed index-matching scheme. In the SCNN case, the scatter network in the backend of SCNN architecture leads to low utilization of multipliers because psums from different multipliers frequently try to access the same accumulation bank. In the SNAP case, the number of comparators is not large enough to produce a sufficient number of matched input and weight pairs to fully utilize the multipliers (Fig. 1b left) when dealing with sparse layers.

***Throughput and Energy Comparison:*** Fig. 6 shows the comparison of performance. In the sparse layers, SNAP shows significantly lower performance than the performance of SPRITE because the index-matching unit cannot match a sufficient number of index pairs thereby reducing the utilization of the multipliers. SCNN performs pixel-dimensional tiling for the input activations, and each tiled group of the input activations is dedicated to each PE. Part of (ResNet-18) or most (ResNet-50/152) layers in ResNet have small pixel-dimensional size of the input activations, so SCNN cannot fully utilize

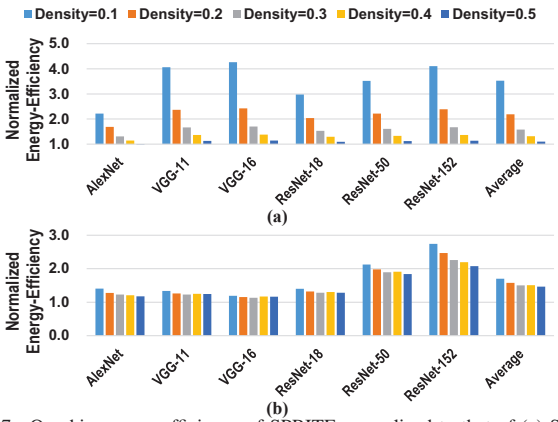


Fig. 7. On-chip energy-efficiency of SPRITE normalized to that of (a) SNAP and (b) SCNN.

the multipliers in the ResNet layers. SPRITE improved the performance by up to 6.1× compared to the previous sparse CNN architectures.

Comparison of on-chip energy-efficiency (Fig. 7) shows a similar trend to the comparison of performance. The on-chip energy-efficiency of SPRITE was improved by 1.30×(density=0.5) - 2.61×(density=0.1) compared to the previous sparse CNN accelerators. The comparison of performance and energy-efficiency was made up to density level = 0.5 because it is well known that the benefit of using sparse matrix format becomes too small when the IA/W density is not low enough due to the overhead from the additional bits to express the location of non-zero elements.

## V. Conclusion

We present a sparse convolutional neural network hardware accelerator, SPRITE, which exploits sparsity of both input activations and weights. Different from previous works which suffer from significant performance degradation such as the data stalls and the under-utilization when covering wide range of sparsity, SPRITE maintains high utilization rate of processing elements thanks to the index-matching unit which matches the indices of input activation and weight pairs with constant probability over wide range of sparsity. The proposed design was implemented in a 28nm CMOS technology and showed an increase in throughput by up to 6.1× compared to the state-of-the-art sparse convolutional neural network hardware accelerators.

## Acknowledgment

This work was supported by Institute of Information communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-01309, Development of artificial intelligence deep learning processor technology for complex transaction processing server).

## References

[1] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 27–40.

[2] J.-F. Zhang *et al.*, "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*. IEEE, 2019, pp. C306–C307.

[3] A. Gondimalla *et al.*, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 151–165.