Ristretto: An Atomized Processing Architecture for Sparsity-Condensed Stream Flow in CNN

Gang Li*
Shanghai Jiao Tong University
Shanghai, China
gliaca@sjtu.edu.cn

Weixiang Xu*
Institute of Automation, CAS
Beijing, China
xuweixiang2018@ia.ac.cn

Zhuoran Song
Shanghai Jiao Tong University
Shanghai, China
songzhuoran@sjtu.edu.cn

Naifeng Jing
Shanghai Jiao Tong University
Shanghai, China
sjtuj@sjtu.edu.cn

Jian Cheng
Institute of Automation, CAS
Beijing, China
jcheng@nlpr.ia.ac.cn

Xiaoyao Liang✉
Shanghai Jiao Tong University
Shanghai, China
liang-xy@cs.sjtu.edu.cn

***Abstract*—Low-precision quantization and sparsity have been widely explored in CNN acceleration due to their effectiveness in reducing computational complexity and memory requirements. However, to support variable numerical precision and sparse computation, prior accelerators design flexible multipliers or sparse dataflow separately. A uniform solution that simultaneously exploits mixed-precision and dual-sided irregular sparsity for CNN acceleration is still lacking. Through an in-depth review of existing precision-scalable and sparse accelerators, we observe that a direct combination of low-level multipliers and high-level sparse dataflow from both sides is challenging due to their orthogonal design spaces. To this end, in this paper, we propose condensed streaming computation. By representing non-zero weights and activations as atomized streams, the low-level mixed-precision multiplication and high-level sparse convolution can be unified into a shared dataflow through hierarchical data reuse. Based on the condensed streaming computation, we propose Ristretto, an atomized architecture that exploits both mixed-precision and dual-sided irregular sparsity for CNN inference. We implement Ristretto in a 28nm technology node. Extensive evaluations show that Ristretto consistently outperforms three state-of-the-art CNN accelerators, including Bit Fusion, Laconic, and SparTen, in terms of performance and energy efficiency.**

***Keywords*-Mixed Precision; Dual-sided Sparsity; Condensed Streaming Computation; Atomized Processing Architecture**

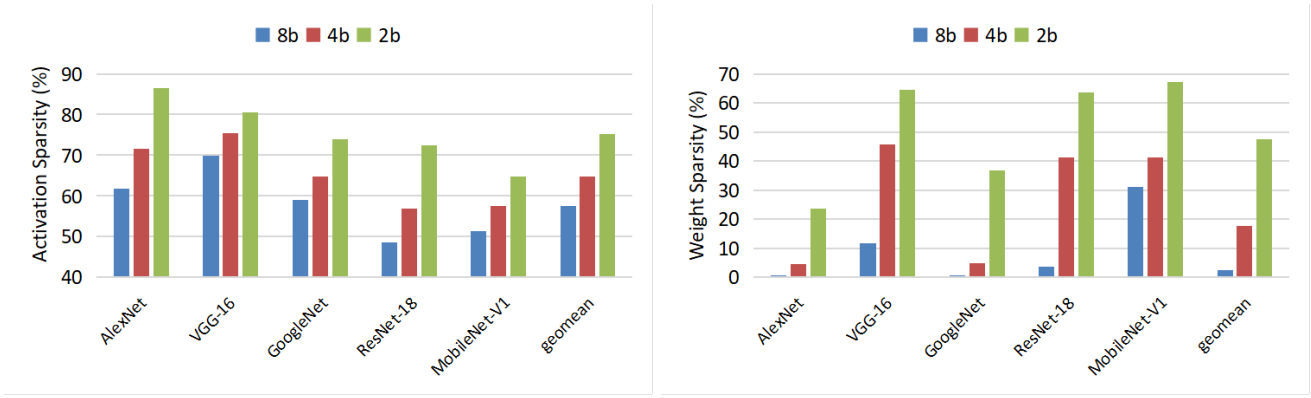
I. Introduction

Nowadays, deep convolutional neural networks have been ubiquitous in various artificial intelligence fields, including computer vision [22, 29], natural language processing [10], and robotics [11, 37]. In order to reduce the computational complexity and memory requirements of CNN for resource-constraint inference, a spectrum of network compression methods have been proposed, such as pruning [21, 33] and quantization [6, 53]. However, the side effect of these techniques is that the compressed network has diverse numerical precisions and irregular sparse patterns across the layers, which is difficult to witness performance gains from conventional general-purpose processors and dedicated accelerators designed for single-precision dense models [5, 8, 9, 26, 43]. Therefore, addressing mixed precision and sparsity simultaneously is crucial for achieving optimal hardware efficiency.

Existing state-of-the-art CNN accelerators handle mixed precision and sparsity separately. Precision-scalable accelerators [3, 27, 44–46] focus on designing flexible multipliers to support variable bit-width at runtime, with little attention to eliminating computations and movements of zero values. Sparse accelerators [13, 17, 19, 39, 57] aim to squeeze out zero values from dataflow, without considering multi-precision adaption in processing elements. Since the design space of precision-scalable accelerators is orthogonal to that of sparse accelerators, we observe that a direct combination of the custom-designed components from both sides is challenging and not optimal. Therefore a comprehensive co-design from the low-level multiplier up to the high-level dataflow will be essential to the efficient handling of both mixed precision and irregular sparsity.

A key idea of this work is that both the high-level sparse convolution and low-level mixed-precision multiplication can essentially be expressed as the computation between two compact streams of non-zero elements. Concretely, 1) at the sparse convolution level, each non-zero weight in the $k \times k$ kernel is multiplied by all the non-zero activations in the input feature map, and each input feature map is convolved with multiple kernels to form partial results of different output feature maps. That is, the sparse convolution can be calculated as the outer product between two compact streams containing only non-zero activations and weights. 2) At the integer multiplication level, when regarding an m-bit integer as a stream composed of several n-bit atoms ($m \geqslant n$), the mixed-precision integer multiplication is equivalent to the outer product of two non-zero atom streams followed by aggregating partial results with proper shiftings. Since data

*Equal Contribution



(a) Activation Sparsity

(b) Weight Sparsity

Figure 1: Visualization of average sparsity of activation and weight under different quantization bit-width. All five networks are trained on the ImageNet dataset and quantized using the same uniform quantization method *without* pruning.

reuse exists in both value-level convolution and bit-level multiplication, the stream-based outer products of the two levels can be seamlessly merged. Then the mixed-precision sparse convolution can be calculated between sparsity-condensed weight and activation atom streams. In this way, the low-level processing elements for mixed-precision multiplication and high-level dataflow for sparse convolution can be unified into a common design space, with a single framework to handle both issues, which is fundamentally different from prior arts. Based on the proposed condensed streaming computation, we propose Ristretto, a dedicated accelerator that employs an atomized processing architecture for CNN inference. The contributions of this paper are as follows:

- To exploit both mixed precision and dual-sided irregular sparsity for accelerating CNN inference, we propose the condensed streaming computation. It represents sparse feature maps and convolution kernels as non-zero atom streams. With the help of hierarchical data reuse in bit-level multiplication and value-level convolution, mixed-precision sparse convolution can be carried out through efficient intersection between atom streams. The fundamental goal of the condensed streaming computation is to unify the low-level mixed-precision multiplication and the high-level sparse convolution into a common dataflow, avoiding the inefficiency of the separate design methodology.
- To support the condensed streaming computation, we propose an accelerator named "Ristretto", which employs an atomized processing architecture featuring massive parallelism and constant input data bandwidth. Ristretto exploits a channel-wise dataflow for streaming-based outer product computation. Incorporated with the compressed data format, Ristretto can eliminate all zero weight/activation movements and zero atom computations during inference. In addition, to mitigate the performance loss caused by the synchronization issue of parallel computing tiles, we propose a greedy-based load balancing approach that can leverage the statistics of both weight and activation.
- To evaluate Ristretto, we implement it on a 28nm technology node and compare it with two state-of-the-art precision-scalable accelerators Bit Fusion [46] and Laconic [44] as well as a state-of-the-art sparse accelerator SparTen [17] on a collection of quantized CNN models. Evaluations show that Ristretto consistently outperforms three accelerators under different quantization bit-width.

The rest of this paper is organized as follows. Section II introduces the background and motivation of this work. Section III describes the proposed condensed streaming computation. Section IV details the Ristretto architecture and load balancing method. Section V shows the evaluation results to manifest the efficiency of the Ristretto accelerator. Section VI concludes this paper.

II. Background and Motivation

A. Accelerating CNN: Sparsity and Mixed Precision

Sparsity is an inherent property of deep convolutional neural networks, which refers to a few weights and activations in the trained network being zeros. The sources of sparsity mainly lie in two aspects. One is the use of the ReLU activation function [38], which transforms a considerable part of negative convolution results to zero. The other is from network pruning [21, 33], which randomly or structurally forces some weights or activations to zero. During network inference, all zero movements and computations can be directly removed without affecting the final results, so sparsity is often used for hardware acceleration of CNN. Low-precision quantization [18, 21] is another key technique in accelerator design. It converts high-precision floating-point numbers into low-precision fixed-point counterparts, which can significantly reduce the area and power consumption as well as bandwidth requirements while maintaining model

Table I: State-of-the-art dual-sided sparse CNN accelerators.

Accelerator	Sparse Dataflow			Compute Unit	
	Pre-Processing	Compute	Post-Processing	MAC	Precision
SCNN [39]	broadcast	outer product	crossbar	2D array	16b
SparTen [17]	inner-join	inner product	permute network	Scalar	8b
SNAP [57]	associative index matching	inner product	two-level reduction	2D array	16b

Table II: State-of-the-art precision-scalable CNN accelerators.

Accelerator	Compute Unit		Dataflow
	MAC	Precision	
LOOM [45]	bit-serial	1∼16b	2D broadcast
Bit Fusion [46]	bit-decomposition	2/4/8b	2D systolic
BitBlade [41]	bit-decomposition	2/4/8b	2D broadcast

accuracy. In conventional single-precision quantization, all layers in a network share the same numerical precision, such as INT8/INT16. Most recently, due to the higher accuracy and lower computational complexity, mixed-precision quantization [6, 14, 35, 48, 53] has shown its superiority in accelerating CNNs in many applications. For example, AdaBits [25] trains a single network that can be used for 2/4/8bit inference adaptively according to the resource constraints of different hardware platforms. VideoIQ [49] dynamically allocates quantization bit-width to different frames for fast video recognition.

Although sparsity and mixed precision are often used separately for accelerating CNN inference, increasing research efforts on the algorithm side have demonstrated that the combination of both can achieve optimal accuracy-performance trade-off [51, 52, 54, 55]. Moreover, there is an intrinsic relationship between sparsity and precision in the quantized models even if pruning is not carried out. As shown in Figure 1, when the quantization bit-width of all five networks goes down from 8bit to 2bit, the sparsity of both weight and activation boosts significantly. Particularly, the average sparsity of weight and activation in the 2bit quantized networks are 47.43% and 75.25%, respectively. This suggests that a joint design for dual-sided sparsity and mixed precision is promising for the accelerator to achieve optimal performance and energy efficiency.

B. Challenge on Accelerator Design

1) Orthogonal Design Spaces: Although the potential of hardware acceleration for both sparsity and mixed precision is obvious, prior accelerators are only designed for either of the two, i.e., sparse accelerators [4, 13, 17, 19, 20, 28, 32, 39, 40, 56–59] and precision-scalable accelerators [3, 7, 12, 15, 27, 31, 36, 41, 44–46, 48]. The goal of a sparse accelerator is to minimize zero-related data movements and computations to reduce inference latency and energy consumption. According to whether the sparsity of weight and activation are exploited simultaneously, the sparse accelerators can be further divided into two categories: one-sided and dual-sided accelerators. The precision-scalable accelerator is designed for integer computations of variable bit-width. Techniques based on spatial or temporal multiplexing are commonly applied in designing bit-flexible multipliers.

We observe that the design spaces of existing sparse and precision-scalable accelerators are orthogonal. Concretely, The sparse accelerator focuses on the design of sparse dataflow, including 1) *pre-processing stage*: extracting non-zero data from feature maps and filters and matching them into pairs; 2) *compute stage*: addressing efficient mapping, scheduling, and computation for disordered and unbalanced non-zero workloads; 3) *post-processing stage*: dispatching and reorganizing the out-of-order results obtained from the second stage, as shown in Table I. For example, to calculate multiple vector inner products in parallel, SparTen [17] broadcasts an activation vector to multiple compute units (CU). In each cycle, each CU uses an inner-join module to extract a non-zero weight-activation pair and feed it into the scalar-based MAC for computation. Results of CUs are finally routed to the data buffer through a permute network. SNAP [57] employs an associative index matching (AIM) to pair non-zero elements in the weight and activation vectors and calculates vector inner product through a 2D MAC array followed by a two-level partial sum reduction. These sparse accelerators exploit either scalar-based MAC or standard 2D MAC array for computation, leaving the majority of design efforts to the sparse dataflow. On the contrary, designing low-level compute units that support variable precision, rather than high-level dataflow, is the key to precision-scalable accelerators. For example, both Bit Fusion [46] and Bitblade [41] design the spatial decomposable PE to support 2/4/8bit multiplications, whereas, at the dataflow level, they select weight stationary-based systolic array and broadcast-based 2D mesh, respectively. The overall architecture of LOOM [45] is also a 2D broadcast-based mesh. However, bit-serial multipliers are applied for 1∼16bit inference, as shown in Table II.

2) Inefficiency of Direct Combination: To design an accelerator that supports both sparsity and mixed precision, an intuitive approach is to directly combine the well-designed sparse dataflow and bit-flexible processing elements of prior works. However, due to the orthogonal design spaces, we notice that it is hard to achieve optimal efficiency in this separate design methodology. Below, we showcase two naive

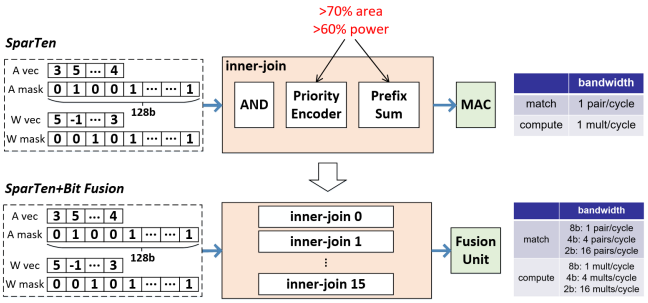


Figure 2: An illustration of combining SparTen with Bit Fusion for mixed-precision sparse computation.

designs to help understand the challenges and inferiority of direct combination.

a) Sparse Accelerator (SparTen) ← Mixed-Precision (Bit Fusion). SparTen [17] is a state-of-the-art dual-sided sparse CNN accelerator. It contains several CUs to calculate the inner product between activation and weight vectors. To eliminate computation and movement of zero values, SparTen employs the bitmap compression format, where a data vector is represented as a compact non-zero vector along with a bitmask that records the positions of non-zero values. To match and extract weight-activation pairs from the compressed vectors for computation, each CU exploits an inner-join module that operates on the bitmasks through priority encoding and prefix summation. In each cycle, The inner-join module searches out a non-zero pair of data and feeds them into the 8bit MAC for computation. It is clear that the sparse dataflow of a CU has an equal bandwidth of data extraction and multiplication, i.e., one effectual multiplication per cycle, as shown in Figure 2.

To support mixed-precision computation, a naive approach is to replace the fixed-precision MAC with a bit-flexible multiplier, such as the fusion unit in Bit Fusion [46] that can perform one 8bit, four 4bit, or 16 2bit multiplications in one cycle. However, this will cause a bandwidth mismatch between data extraction and computation when the bit-width is lower than 8. That is, the sparse dataflow cannot meet the requirement of matching and feeding more than one pair of data to the fusion unit. To increase the bandwidth of non-zero matching, up to 16 inner-join modules are needed to work in parallel in each CU, as shown in Figure 2. This will result in two problems: 1) significant overhead on the area and power consumption since one inner-join module accounts for more than 60% of the area and power consumption of a CU; 2) severe resource underutilization when the CU does not work at the peak throughput (16 mults/cycle). Moreover, the unbalanced workloads of the parallel inner-joins can also lead to non-negligible performance loss. Note that similar issues caused by bandwidth mismatch between high-level sparse dataflow and low-level multiplication can also be witnessed in other sparse accelerators [39, 57] when directly replacing

Table III: Sparsity exploitation of state-of-the-art precision-scalable accelerators.

Accelerator	Weight	Activation	Weight Bit	Activation Bit
Bit-Pragmatic [3]				✓
Bit-Tatical [12]	✓			✓
Laconic [44]			✓	✓
Ristretto (this work)	✓	✓	✓	✓

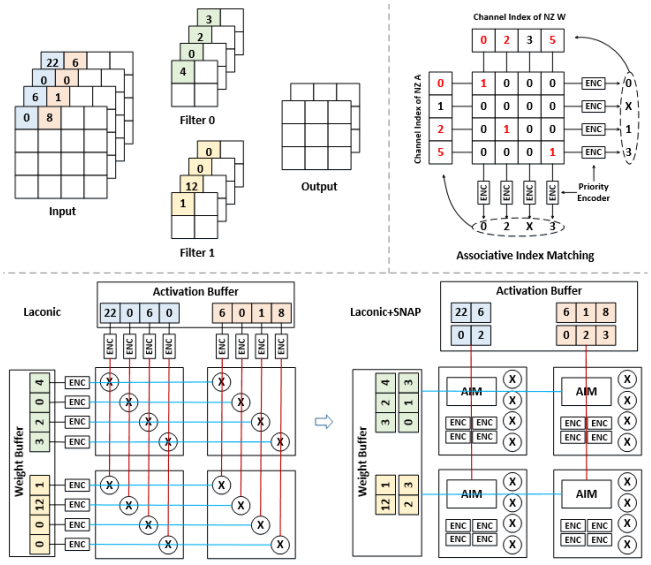


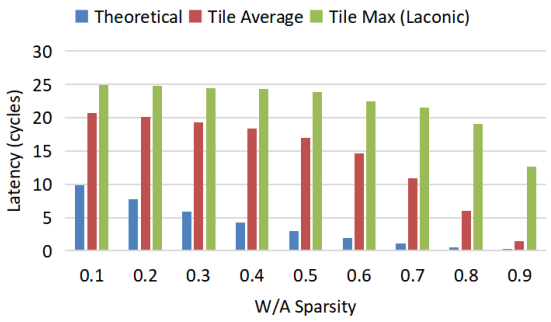
Figure 3: An illustration of incorporating Laconic with SNAP to eliminate movement of zero values.

the fixed-precision MAC with a precision-scalable MAC.

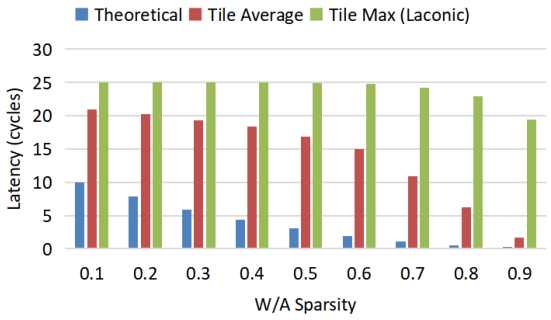
b) Precision-Scalable Accelerator (Laconic) ← Sparsity (SNAP).

Bit-serial computation has been widely applied in precision-scalable accelerator design. Compared to bit-parallel computation, it enables exploring bit-level sparsity for fine-grained acceleration. Table III lists three state-of-the-art bit-serial CNN accelerators. Among these, Laconic [44] has been demonstrated to outperform the other two in terms of performance.

The architecture of a Laconic tile is depicted in Figure 3. It employs a 2D mesh architecture, each PE in the 2D array contains multiple bit-serial multipliers that undertake the computation of the inner product between weight and activation vectors. Filters and feature maps in the buffers are broadcast within rows/columns, where each row shares the same weights to calculate different spatial convolution windows, and each column shares the same activations to calculate different output features. To extract effectual terms (non-zero bit and corresponding shift offset), booth encoders are placed at the boundary of the PE array. During inference, non-zero terms in activations and weights are traversed to perform multiplication. Due to data sharing and parallel computing, the latency of a PE is determined by the non-zero weight-activation pair that has the maximum



(a) 4×4 PEs



(b) 8×8 PEs

Figure 4: Visualization of Laconic performance with respect to different levels of value sparsity. Each PE contains 16 parallel bit-serial multipliers to calculate inner product between 8bit vectors. The sparse vectors are randomly generated from a uniform distribution, and the results are an average of 1000 runs.

$\#term_a \times \#term_w$, and the overall latency of a tile is determined by the slowest PE.

We can notice that Laconic is insensitive to value-level sparsity. For a better understanding, we conduct an experiment to investigate the performance of a Laconic tile for inner product computation under different levels of weight/activation sparsity, as shown in Figure 4. The theoretical latency is the performance upper bound, which is measured by dividing the total number of workloads by the number of multipliers in the PE array, where the workload of one multiplication is denoted as $\#term_a \times \#term_w$ for a weight-activation pair. The average tile latency assumes that data sharing among PEs is disabled, and only parallel bit-serial computations are performed in each PE. It is clear that: 1) both data sharing among PEs and parallel computing within a PE contribute significant degradations to the performance; 2) an increasing value-level sparsity does not lead to significant improvement in performance, and the benefits of sparsity will decrease as the tile size gets larger.

To eliminate movements of zero values from Laconic, compression formats such as CSR can be applied to the

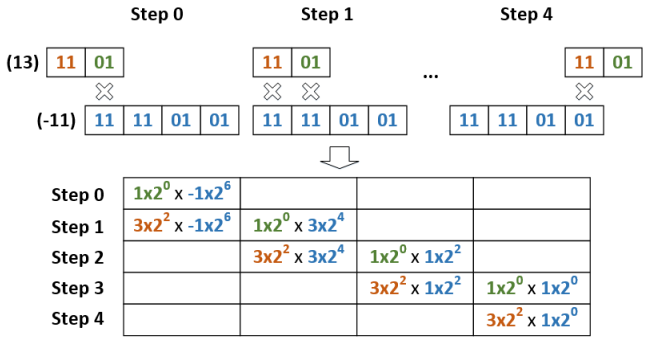


Figure 5: An example of multiplication between 4bit activation and 8bit weight using 1D convolution.

dense tensors. In this situation, the inner product between condensed vectors is performed in each PE. Since the sparse patterns of pair-wise vectors are different among PEs, local matching logics that extract weight-activation pairs are needed. Besides, the booth encoders at the array boundary should be moved into the PEs for local encoding. To perform local data matching, an associative index matching (AIM) from SNAP can be integrated into each PE. The modified Laconic architecture is shown in Figure 3. It has two problems: 1) considerable area overhead introduced by local matching and encoding; 2) significant PE underutilization caused by zero weights and activations, especially when the value-level sparsity is high. Note that Bit-Pragmatic [3] and Bit-Tactical [12] in Table III have a similar broadcast-based 2D mesh architecture with Laconic, it is challenging for them to address both dual-sided bit-sparsity and value-sparsity efficiently, so does Bit Fusion [46] that features a canonical 2D systolic array.

In conclusion, due to the orthogonality between sparse dataflow and bit-flexible compute units in prior works, an incremental design that simply combines the components from both sides is not ideal. To tackle this issue, in this paper, we propose Ristretto, an accelerator that, for the first time, exploits both dual-sided irregular sparsity and mixed precision for CNN inference. Based on the proposed condensed streaming computation, the low-level mixed-precision multiplication and high-level sparse convolution can be seamlessly merged into a shared dataflow and efficiently calculated with an atomized processing architecture.

III. Condensed Streaming Computation

In this section, we present the dataflow of the Ristretto accelerator. First, we introduce the hierarchical data reuse that resides in CNNs, which serves as the bridge to unify the stream-based representations of high-level sparse convolution and low-level mixed-precision multiplication. Then we detail the proposed condensed streaming computation, an atomized framework for mixed-precision sparse processing.

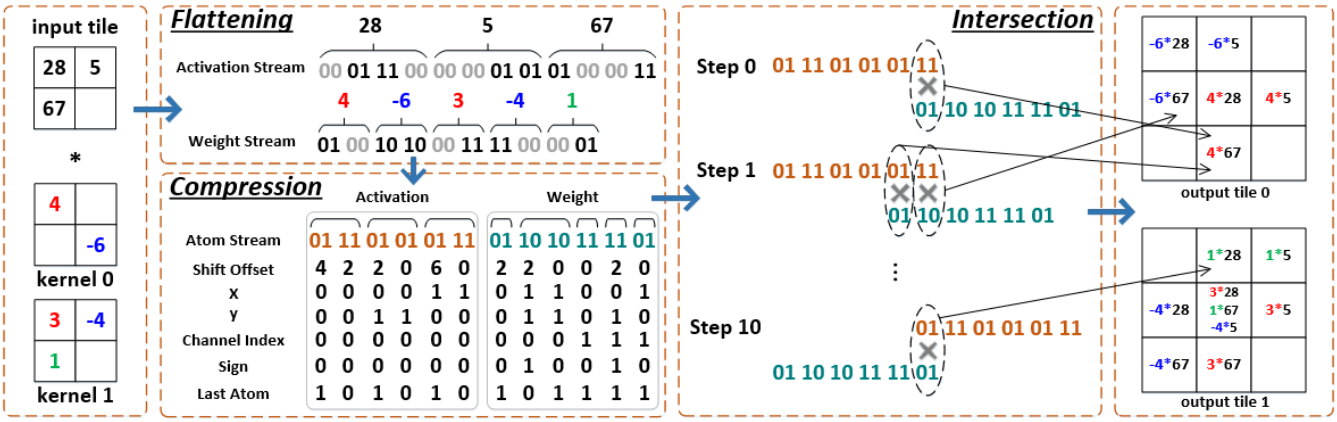


Figure 6: An illustration of the condensed streaming computation for mixed-precision sparse convolution.

A. Integer Multiplication as 1D Convolution

In quantized models, all weights and activations are represented as integers. Due to the use of the ReLU function [38], activations are usually regarded as unsigned integers, whereas weights are signed integers. Arithmetically, an m-bit integer can be viewed as the summation of $\lceil m/N \rceil$ terms, where each term refers to the product of an N-bit atom and its corresponding shift offset. For example, 29 (01_11_01) is equal to the summation of terms in $\{1 \cdot 2^4, 3 \cdot 2^2, 1 \cdot 2^0\}$ under 2bit atom granularity. Therefore, the product of an m-bit and an n-bit integer can be expressed as the summation of the outer products between two streams with $\lceil m/N \rceil$ and $\lceil n/N \rceil$ terms, respectively. Note find that this computation can be reformulated as a 1D convolution between two streams. Figure 5 shows an example of calculating -11×13 using 1D convolution. In this example, the 4bit activation and 8bit weight contain two and four 2bit atoms, respectively. The entire computation consists of five steps. In each step, parallel atom multiplications are carried out in the intersection region between two streams. The final result is the aggregation of all partial products over the five steps. It can be noticed that there is massive atom-level data reuse throughout the computation. For example, each weight atom is multiplied by all activation atoms over five steps, which is similar to the weight sharing scheme in CNN.

B. A Unified Framework

There are two types of value-level data reuse in CNN: input reuse and weight reuse. The input reuse refers to an input feature map is convolved with different convolution kernels to obtain multiple output feature maps, whereas the weight reuse denotes that in 2D convolution, each weight in the $k \times k$ kernel is multiplied by the activations of a specific input region to form the partial results of an output feature map. Since data reuse exists at both the atom and value level, the high-level convolution and low-level integer multiplication have the potential to be merged to form a unified representation. Based on this observation, we propose condensed streaming computation (CSC), a unified dataflow to exploit both dual-sided sparsity and mixed precision for CNN inference. The CSC pipeline includes three parts: flattening, compression and intersection. Figure 6 shows an example of mixed-precision sparse convolution using CSC, where an 8bit input feature map tile is convolved with two 4bit kernels to obtain two output feature map tiles. Concretely, the CSC workflow is as follows:

1) **Flattening.** Both the input feature map tile and convolution kernels are flattened to compact 1D streams. As depicted in Figure 6, the 2×2 feature map tile and two 2×2 kernels are reshaped in a zigzag manner. Note that along with each weight/activation, there is also metadata, i.e., the spatial coordinates and channel index, that is used for recording the position of each non-zero value.
2) **Compression.** This phase squeezes out all zero atoms from weight and activation streams and reorganizes them into compact atom streams. In this way, both value-level and bit-level sparsity are exploited, enabling the maximal reduction of ineffectual workloads under a certain atom granularity. Since weights and activations in each stream are broken down into atoms, shift offsets and last atom flags are generated to identify the relative position of each non-zero atom in a weight/activation. Besides, a sign bit is also generated for each atom to denote whether it is signed or unsigned.
3) **Intersection.** In this phase, 1D convolution between atom streams is performed, which is similar to the computation in Figure 5. The weight stream is kept static, and the activation stream is gradually shifted from the head to the tail of the static stream with a step size of one atom. In each step, parallel atom multiplications are conducted in the intersection region between two streams, and proper shifting is added to each partial product before accumulating to the

final results. The alignment and aggregation of partial products are scheduled by the metadata, including shift offsets, spatial coordinates, and channel indices. Note that a long static stream can be partitioned into several short streams. In this case, the shifting of the activation stream will be repeated multiple times.

From the above workflow, it is clear that by leveraging hierarchical data reuse, the high-level sparse convolution and low-level mixed-precision multiplication are unified into a common dataflow, where computations and movements of zero atoms are eliminated. Note that the flattening and compression of filters can be conducted offline with the help of compression format (detailed in Section IV-B). For feature maps, both value-level and atom-level compression are implemented on-chip. Specifically, the zero values in feature maps can be squeezed out through a post-processing unit when the computation of an output group is finished, and the zero atoms in activations can be removed on the fly with a negligible cost (detailed in Section IV-C1).

Characteristics. We notice that the condensed streaming computation has the following key characteristics:

- It operates at a constant input data bandwidth under different bit-width. For example, assume that there are 16 2bit multipliers that enable 16 2bit atom multiplications in parallel. In each step, it can calculate one 8bit, four 4bit, or 16 2bit multiplications with a constant input bandwidth of 2bit/step. However, with the same computational throughput, the input (activation) bandwidth of a fusion unit in Bit Fusion [46] under 8bit, 4bit, and 2bit configurations are 8 bit/cycle, 16 bit/cycle, and 32 bit/cycle, respectively.
- The number of steps in the intersection phase is solely determined by the length of two streams, which can be precisely estimated in advance. Assume the number of activations and weights in input feature map and kernels are A and W, and the density at value and atom (of non-zero values) levels are α_v, α_a, and β_v, β_a. Then the length of non-zero atom streams are $\alpha_v\alpha_a A$ and $\beta_v\beta_a W$, respectively. Ideally, the number of steps in the intersection phase is $\alpha_v\alpha_a A * \lceil \beta_v\beta_a W/N \rceil + \varepsilon$, where N is the length of the static stream.
- Changing the order of atoms (and associated metadata) within a stream do not affect the final result since each atom in one stream is calculated with every atom in the other stream.
- Due to data reuse, the number of workloads among all atoms is consistent, which indicates that there is no balancing issue within a stream during intersection.

In the next section, we will elaborate on how these characteristics are leveraged for microarchitecture design and optimization.

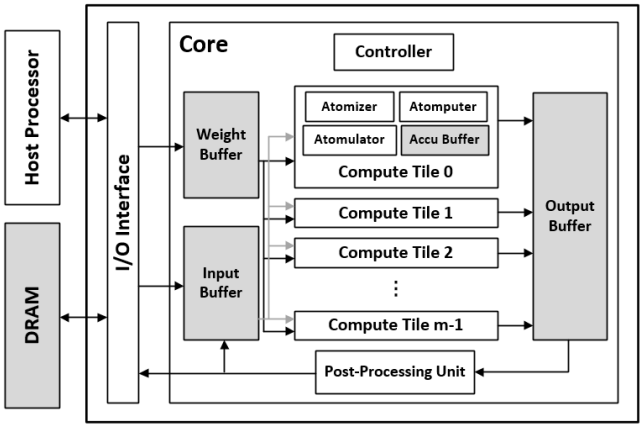


Figure 7: Overall Architecture of the Ristretto Accelerator.

IV. Ristretto Architecture

A. Overview

The sparsity and diverse numerical precision in deep neural networks provide a unique opportunity for dedicated accelerators to achieve optimal efficiency. We propose Ristretto, a novel CNN accelerator that can simultaneously handle dual-sided irregular sparsity and mixed-precision computation. Figure 7 shows the overall architecture of the Ristretto accelerator. The on-chip compute cores are communicated with the off-chip processor and DRAM via an I/O interface. Each compute core contains data buffers, compute tiles, a post-processing unit, and a controller.

The workflow of the Ristretto accelerator is as follows. When the host processor issues a start command, a batch of compressed feature map tiles and convolution kernels in the off-chip DRAM are loaded to the compute cores and stored in the input and weight buffers. Before the calculation starts, the compressed kernels and associated indices required for the current round of calculation are read from the weight buffer and fed to the Atomputer (abbreviated from the "atom computer") in each compute tile under the control of the core controller. When the calculation starts, the activation words are read from the input buffer and sent to the compute tiles in parallel. By removing zero atoms from an activation word, the Atomizer pops the non-zero atoms and associated metadata into the subsequent Atomputer and Atomulator (abbreviated from the "atom accumulator") for computation. During the process, partial results of each compute tile are aggregated and written to the shared output buffer periodically. When a group of output feature maps is obtained in the output buffer, they are sent to the off-chip DRAM or input buffer after post-processing.

B. Compression Format and Buffer Organization

As mentioned in Section III-B, the flattening and compression phases of the condensed streaming computation

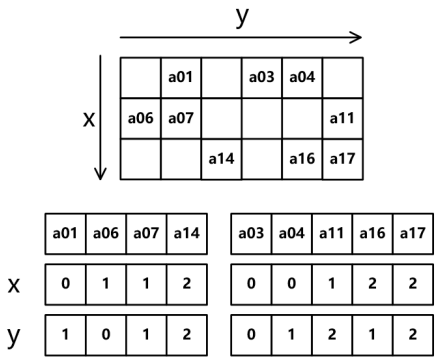


Figure 8: Block COO-2D format.

Table IV: Shift ranges under different activation bit-width.

Activation Bit-width	8b	6b	4b	2b
Shift Range	0,2,4,6	0,2,4	0,2	0

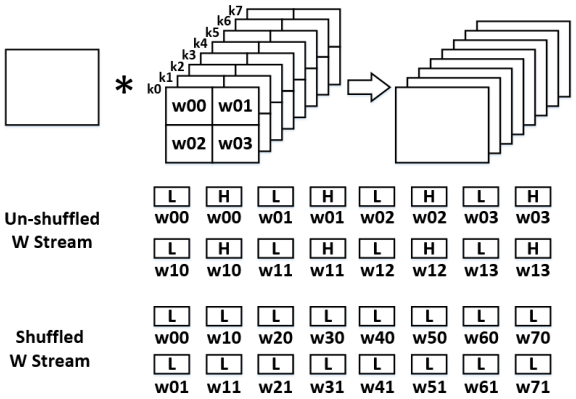


Figure 9: An illustration of stream shuffle. Each kernel contains four 4bit weights. L and H refer to the low 2bit and high 2bit atoms respectively. For clarity, all atoms are regarded as non-zeros in this example.

can be implicitly realized via compression format and buffer organization. Concretely, Ristretto employs the COO-2D format to compress the feature maps and filters to avoid both on-chip and off-chip movement of zero values. Since the input feature maps are partitioned into tiles during inference, the block COO-2D encoding is adopted, where the coordinate of the non-zero activation represents the spatial offset to the starting position of a tile, as shown in Figure 8. As for the zero atoms in non-zero values, since the network weights are fixed after training and quantization, Ristretto removes zero atoms from weights offline. That is, only the non-zero weight atoms and the corresponding metadata are transferred during inference. However, for feature maps, it is not practical to exclude zero atoms off-chip. First, the massive bit operations at runtime will lead to a significant burden on the host processor. Second, since feature maps dominate the off-chip traffic during inference, the overhead caused by metadata (shift offsets, position indices, etc) of feature map tiles is much larger than that of kernels. In Section IV-C1 we will show that the zero atoms in feature maps can be squeezed out on the fly through the Atomizer with a negligible cost.

According to the condensed streaming computation, multiple output feature maps can be calculated in parallel. Therefore, the on-chip buffers exploit a multi-bank design to enable parallel access. Each bank of input buffer continuously stores multiple compressed feature map tiles that are mapped to the same compute tile. The metadata associated with feature maps is stored in a separate buffer in the same order. Similarly, kernels corresponding to the same input feature map in the weight buffer are stored continuously.

C. Compute Tile Microarchitecture

In Ristretto, the compute tiles are responsible for the condensed streaming computation. Apart from an accumulate buffer, each compute tile contains an Atomlizer, an Atomputer, and an Atomulator, which deals with the following issues respectively: 1) how to parse the incoming activation words and metadata to generate compact non-zero atoms and associated control information; 2) how to perform computation between streams; 3) how to collect and accumulate partial results. In this section, we detail the microarchitecture of the compute tile that supports 2/4/8bit inference. By default, the atom bit-width is set to 2bit.

1) Atomizer: The Atomizer takes an 8bit activation word and a (x, y) coordinate as input. In each cycle, it scans the activation word through a low-cost leading one detection logic to generate a 2bit non-zero activation atom, a shift offset, a last flag, and a (x, y) coordinate. The first three are the input to the Atomputer, and the last one is sent to the Atomulator. Note that the generation of sign bit for activation atom is ignored in the Atomizer because all atoms are unsigned when the ReLU function is used. Since zero values are removed from feature maps beforehand, each 8bit word contains at least one, two, and four non-zero atoms under 8bit, 4bit, and 2bit quantization. This indicates that the Atomizer will hold an 8bit word for at most four cycles before reading the next word from the data buffer. When a word contains multiple (2bit or 4bit) activations, the generated last flag is used for recording whether the current atom is the last atom of the same activation. Similarly, if multiple atoms belong to the same activation, the (x, y) coordinate is latched for more than one cycle before the next is read. For a 2bit atom, the possible shift offset is within the range of $\{0, 2, 4, 6\}$, as shown in Table IV.

2) Atomputer: The Atomputer undertakes the intersection operation of the condensed streaming computation, where parallel atom multiplications are performed between the static weight and dynamic activation streams in a systolic style. However, an arbitrary organization of atoms in the weight stream will lead to considerable overheads in microar-

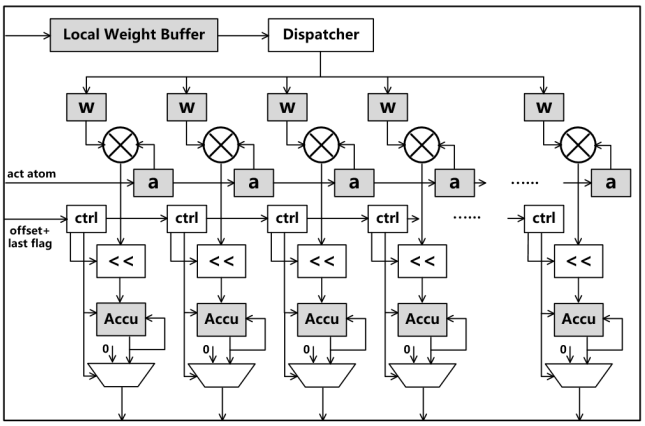


Figure 10: Microarchitecture of the Atomputer.

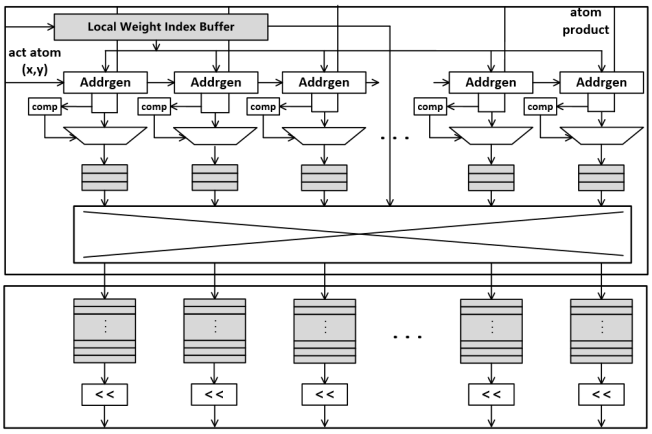


Figure 11: Microarchitecture of the Atomulator and accumulate buffer.

chitecture design. First, for each 2bit atom multiplication, the combination of shift offsets from both operands will cause a wide shift range, i.e., $\{0,2,4,6,8,10,12\}$, which is not area-efficient. Second, if atoms belonging to the same weight are continuously mapped, complex reduction units (such as augmented reduction tree in [30]) are required to aggregate the adjacent partial products destinated to the same output. Fortunately, since shuffling atoms within a stream does not affect the final result, the above issues can be alleviated through dataflow-microarchitecture co-design. As depicted in Figure 9, we make the following restrictions on the static weight stream: 1) To reduce the complexity of shifting, atoms corresponding to the same slice of weight are grouped and mapped continuously. This allows a decoupled shift operation, where only the shift offset of the activation atom is required in each multiplication, leaving the other to operate on the accumulated results; 2) To eliminate coordinate contention, weight atoms are mapped in a channel-first manner to calculate different output feature map.

Figure 10 shows the microarchitecture of the Atomputer. It contains a local weight buffer, a dispatcher, and a chain of atom multipliers. Each multiplier is connected to a shift unit followed by an accumulator. Before calculation starts, the weight atoms and associated sign bits are read from the local weight buffer and latched in the weight atom registers to form a static stream. In each cycle, the leftmost multiplier receives a non-zero activation atom from the Atomizer to perform multiplication, and right shifts the current atom before the next arrives. Before sending to the accumulator, each product is aligned in the shift unit according to the shift offset of the activation atom. When the last flag is 1, the accumulator delivers the valid result outside and clears the accumulation register to zero at the end of the cycle. Note that all the weight atom registers work in a ping-pong manner. This allows an intermediate update of the weight atoms when the computation of a weight atom is done.

3) Atomulator: When the atom products are obtained in the Atomputer in each cycle, the Atomulator is responsible for generating output coordinates and routing these products to the accumulate buffer. Figure 11 shows the microarchitecture of the Atomulator. It contains a local weight index buffer, a chain of address generators, and a crossbar. The address generators also work in a systolic manner, which is synchronized with the multipliers in the Atomputer. In each cycle, the address generators receive the (x,y) coordinates of the weight and activation atom to calculate the write addresses to the accumulate buffer. The coordinates of weight and activation atoms are from the local weight index buffer and the Atomizer, respectively. Mathematically, given the spatial coordinates of input activation (x_{in},y_{in}) and weight (x_w,y_w), the output coordinate (x_{out},y_{out}) can be calculated as:

$$\begin{cases} x_{out} = k-1-x_w+x_{in} \\ y_{out} = k-1-y_w+y_{in} \end{cases} \tag{1}$$

where k is the kernel size. Accordingly, the address can be calculated as:

$$address = y_{out} * (W_{in}+k-1)+x_{out} \tag{2}$$

where W_{in} denotes the width of the input feature map tile. This address is the offset to the original point of the current output feature map tile. Since the coordinates calculated by Equation 1 may be out of the boundary of an output feature map, the generated address is validated through the *comp* module before pushing to the subsequent queue. Similar to [39], we do not address the non-unit convolution stride in the Atomputer due to the high complexity of managing out-of-order coordinates of compressed streams. Therefore the output coordinate in Equation 1 is always calculated assuming the stride is equal to 1. Although this will inevitably introduce ineffectual outputs, the stride access to the effectual output values can be easily realized in the accumulate buffer with negligible cost.

According to the dataflow illustrated in Section IV-C2, the output channel index of each atom product is solely determined by the weight atom. Therefore, the pair-wise atom product and address are routed through the crossbar under the control of channel indices of weight atoms. When more than one product is destined to the same output feature map in one cycle, a direct routing will cause buffer access contention and computation stall. To alleviate this issue, we add low-cost FIFOs before the crossbar, as in SCNN [39].

4) Accumulate Buffer: The accumulate buffer exploits a multi-bank design and is implemented as register files. Each bank is used for caching an output feature map tile of a specific channel. To enable parallel access, the number of banks is kept consistent with the length of the static stream in the Atomizer. Results of different compute tiles are aggregated and written to the output buffer when the computation of one weight slice is finished. To overlap computation with data transfer, each bank is double-buffered.

As illustrated in Section IV-C2, Ristretto exploits a decoupled shift to reduce the complexity of the Atomputer. Therefore, to perform the second stage of shift, each buffer bank is connected to a shift unit, as shown in Figure 11. During aggregation, each partial result is read and aligned according to the shift offset of the corresponding weight slice.

D. Support for 16/32bit Inference

Ristretto can be configured to support high-precision inference with two methodologies: spatial extension and temporal decomposition. Unlike prior work that leverages additional PEs to calculate higher bytes and lower bytes separately in spatial extension [44], we can simply modify the shifters in Ristretto to support a wider shift range, such as $\{0,2,4,6,8,10,12,14\}$ for up to 16bit inference. A more economical method is to decomposite a high-precision model into several low-precision models and calculates in sequence. For example, a 16bit model can be partitioned into four 8bit sub-models. This allows much smaller shifters to reduce area overhead.

E. Load Balancing

In Ristretto, the input feature maps are partitioned into several groups in the channel dimension. Each group is mapped on a compute tile to calculate the same group of output feature maps. Due to the varying number of non-zero atoms in each feature map and convolution kernels, the number of workloads in different compute tiles is unbalanced, leading to performance loss due to resource underutilization. Therefore, load balancing is essential to improve the accelerator efficiency.

Ideally, the most promising choice for load balancing is to use the statistics of both input feature maps and kernels. However, it is challenging to achieve this goal in prior works. One of the main reasons is that the statistics needed for load balancing can not be collected until the calculation starts. For example, SparTen [17] extracts non-zero pair of weight and activation on the fly during the calculation of the vector inner product so that the number of non-zero pairs between vectors can not be known in advance. However, in the condensed streaming computation, the latency is determined by the length of streams, which can be obtained before the calculation starts. First, the number of non-zero weight atoms can be collected offline when training and quantization are finished. Second, when a group of output feature maps is obtained, the number of non-zero atoms of each feature map can be counted through the post-processing unit using a low-cost module that is similar to the Atomizer. Therefore, Ristretto jointly utilizes weight and activation for load balancing.

Considering the computation of an input feature map with multiple convolution kernels in a compute tile. Assume that the number of non-zero atoms in a feature map tile and kernels is t and S, respectively, and the number of multipliers in the Atomputer is N. According to the principle of condensed streaming computation, the theoretical calculation cycles corresponding to one feature map tile can be estimated as:

$$C_t = t * \left\lceil \frac{S}{N} \right\rceil + \varepsilon \tag{3}$$

where,

$$\varepsilon = \begin{cases} mod(S,N) - 1, & mod(S,N) \neq 0 \\ N - 1, & mod(S,N) = 0 \end{cases} \tag{4}$$

Since ε is much smaller than the first term in Equation 3, we omit it for simplicity. Since the convolution kernels are shared among all tiles in the input feature map, the inference cycles corresponding to a whole input feature map can be formulated as:

$$C_T \approx \sum_0^m (t_i * \left\lceil \frac{S}{N} \right\rceil) = (\sum_0^m t_i) * \left\lceil \frac{S}{N} \right\rceil = T * \left\lceil \frac{S}{N} \right\rceil \tag{5}$$

where T is the number of non-zero atoms in the input feature map. It is clear that the inference cycle is determined by the non-zero atoms of the input feature map and corresponding kernels. Therefore, Ristretto uses the metric in Equation 5 to allocate workloads for different compute tiles.

Assume the number of input and output feature maps in one inference phase in m and n, respectively. Each input feature map is convolved with n kernels to obtain n output feature maps. Assume the number of compute tiles and multipliers is M and N. Then, the goal of load balancing is to divide the m input feature maps and corresponding kernels into M groups so that the number of workloads in different groups is as close as possible. Similar to SparTen [17], Ristretto exploits a greedy-based approach. First, the C_T values for each input feature map and corresponding kernels are calculated. Second, the m C_T values are paired in a "largest-smallest, second largest-second smallest" manner

Table V: Baseline accelerators evaluated in this work.

Accelerator	Value Sparsity	Bit Sparsity	Variable Precision
Bit Fusion [46]			✓
Laconic [44]		✓	✓
SparTen [17]	✓		
SparTen-mp	✓		✓

until the number of groups is equal to M. In this way, the input feature maps and kernels belonging to the same group are mapped to the same compute tile. Since the balancing can be performed before the calculation starts, each group can be loaded and stored in a separate buffer bank, avoiding the crossbar between the input buffer and compute tiles.

To balance the workloads of the input layer, an intuitive approach is to count the number of non-zero atoms in the host processor before inference starts. However, this will introduce non-negligible overheads, especially for low-end processors. Therefore, a compromise made by Ristretto is that it does not conduct load balancing for the input layer.

V. Evaluation

A. Methodology

1) Baseline Accelerators: As shown in Table V, we select three state-of-the-art CNN accelerators as our baselines: Bit Fusion [46], Laconic [44], and SparTen [17]. Among them, the first two are precision-scalable accelerators, and the latter is a sparse accelerator. To future investigate the efficiency of the direct combination of high-level sparse dataflow and low-level bit-flexible multiplier for mixed-precision sparse computation, we additionally design a naive baseline: SparTen-mp, as mentioned in Section II-B2. Specifically, SparTen-mp replaces the fixed-precision MAC with the fusion unit of Bit Fusion [46] that can achieve one 8bit, four 4bit, or 16 2bit multiplications in one cycle. To match the peak bandwidth between computation and data extraction in the 2bit setting, 16 inner-join modules are placed in a CU and work in parallel, each inner-join operates on a bitmask of length 32.

2) DNN Benchmark: In order to verify the proposed Ristretto accelerator, we select six CNN networks trained on the ImageNet dataset: AlexNet [29], VGG-16 [47], GoogLeNet [50], Inception-V2 [24], ResNet-18 [22], and ResNet-50 [22]. Unlike the channel-wise dataflow of Ristretto, all baseline accelerators are designed for the inner product of channel vectors so that they do not explicitly support depthwise convolution in PEs. Therefore we omit the evaluation of MobileNets [23, 42] in this work. To get the low-precision sparse models, we use the same uniform quantization method to quantize each network to 8bit, 4bit, and 2bit, and further prune the weight and activation without hurting the accuracy of the quantized models. We also collect 2/4bit mixed-precision models using EdMIPS [6], where the bit-width of weight and activation in each layer is independently chosen from {2,4}.

Table VI: Area breakdown of Ristretto accelerator.

Accelerator Component		Area (mm^2)
Compute Tile	Atomizer	0.001
	Atomputer	0.070
	Atomulator	0.128
	Accu Buffer	0.496
Data Buffer	Input	0.118
	Weight	0.302
	Output	0.154
Post-Processing Unit		0.023
Others		0.004
Total		1.296

3) Simulation: We use Verilog to implement Ristretto and the baseline accelerators, and use the Synopsys Design Compiler to synthesize the RTL with a TSMC 28nm HPC+ standard cell library at 500MHz to obtain the area and power consumption of the computing units. For SRAM-based on-chip buffers and register files, we use CACTI-P [34] to model their area and power consumption. For off-chip DRAM, we follow the methodology in [16] to get the energy per access. We develop cycle-accurate simulators for Ristretto and SparTen to collect the statistics of computation and buffer access for each workload. For the simulation of Bit Fusion and Laconic, we refer to their open-source implementations [1, 2]. The area breakdown of a single-core Ristretto is shown in Table VI. It contains 32 compute tiles, and each compute tile has 32 2bit multipliers.

B. Comparison with Bit Fusion

In this experiment, we compare Ristretto with Bit Fusion in terms of performance and energy consumption. Since each fusion unit of Bit Fusion contains 16 2bit multipliers, we also adopt a 2bit atom configuration in Ristretto. To highlight the contribution of sparsity to the performance gain, we additionally design a non-sparse Ristretto (named Ristretto-ns) to match the original design goal of Bit Fusion. For fair comparison, we follow the methodology in [13] that constrains all accelerators to have the same number of 2bit MACs. Specifically, Ristretto and Ristretto-ns use a single core with 32 compute tiles, each compute tile contains 32 2bit multipliers, whereas Bit Fusion employs an 8×8 2D systolic array. In this setting, all three accelerators contain 1024 2bit multipliers. Besides, we allocate the same size of on-chip buffer to all accelerators.

Figure 12 compares the area-normalized performance of the three accelerators on the DNN benchmark. On all models, Ristretto outperforms Bit Fusion by a large margin, with an average speedup of 8.2×, 7.47×, 7.13×, and 6.73× on 8bit, 4bit, 2bit, and mixed 2/4bit models, respectively. We can see that the performance of Ristretto-ns is very close to that of Bit Fusion when sparse computation is disabled. This suggests that the performance gain of Ristretto over Bit Fusion lies in the exploiting of sparsity at both value and atom levels.

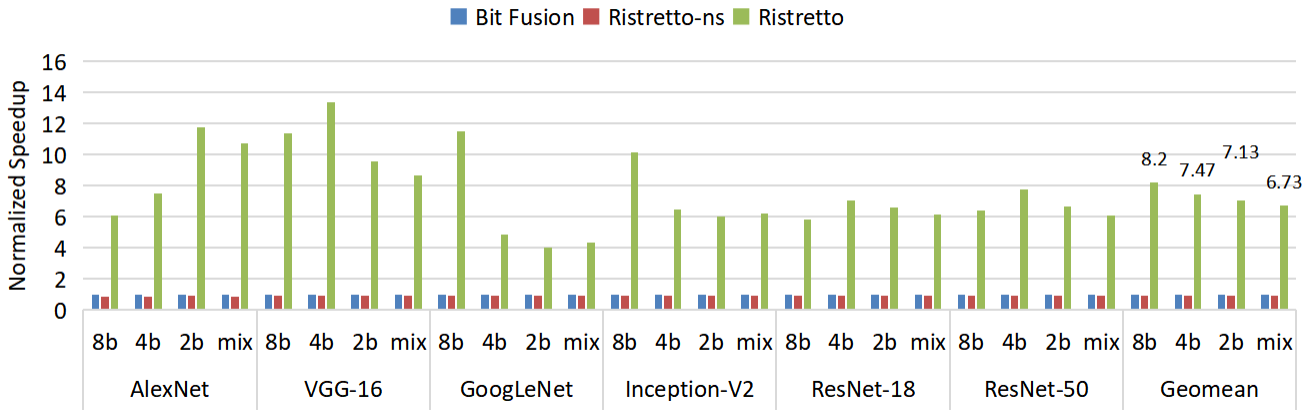


Figure 12: Comparison of performance between Ristretto and Bit Fusion. All results are normalized to Bit Fusion.

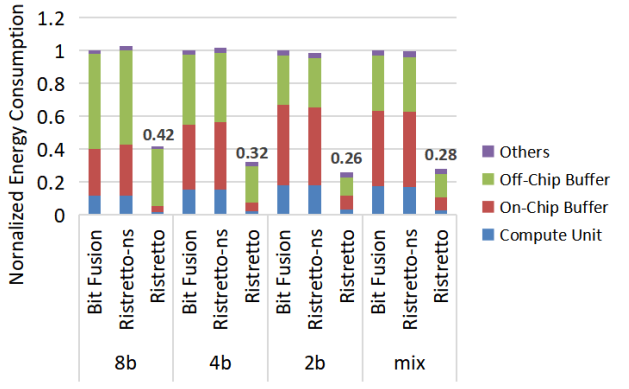


Figure 13: Comparison of energy consumption between Ristretto and Bit Fusion. All results are normalized to Bit Fusion.

Figure 13 shows the average energy consumption of three accelerators under different bit-width. Due to the use of compression format, the energy consumption of both on-chip and off-chip buffer accesses in Ristretto is significantly reduced. The energy consumption under 8bit, 4bit, 2bit, and mixed 2/4bit are 41.84%, 32.29%, 33.33%, and 26.16% of Bit Fusion, respectively.

C. Comparison with Laconic

This experiment compares Ristretto with Laconic. Since booth encoding is applied to remove zero bits, both weights and activations in Laconic are represented as sequences of shift offsets, converting conventional MACs to exponent additions followed by decoding-based accumulation. For fair comparison, we follow the methodology in [44] that configures Ristretto and Laconic to use the same compute area and on-chip buffer size. In this case, Ristretto contains 32 compute tiles, each compute tile consists of 16 2bit multipliers. Laconic has a 6×8 PE array, each PE calculates a vector inner product of length 16.

Figure 14 shows the results of inference performance. Ristretto outperforms Laconic in all settings, with an average speedup of 3.58×, 4.18×, 6.12×, and 5.69× on 8bit, 4bit, 2bit, and mixed 2/4bit models, respectively. It can be noticed that the speedup increases as the quantization bit-width narrows. As detailed in Section II-B2, due to the inefficiency of broadcast-based dataflow and SIMD-based inner product computation, Laconic cannot fully take advantage of value-level sparsity to improve performance. However, thanks to the massive data reuse in the condensed streaming computation, Ristretto can achieve much higher resource utilization. From Figure 15, it is clear that Ristretto is more prone to benefit from the increased sparsity. Figure 16 shows the comparison of energy consumption. Since both feature maps and filters are stored and communicated in the dense format in Laconic, the energy consumption of Ristretto caused by on-chip and off-chip buffer accesses is much lower.

D. Comparison with SparTen and SparTen-mp

In this part, we compare Ristretto with SparTen and SparTen-mp. Since SparTen and Ristretto use 8bit and 2bit multipliers respectively, we constrain all three accelerators to have the same peak BitOps/cycle for fair comparison. In this setting, Ristretto contains 32 compute tiles, each tile includes 16 2bit multipliers. Both SparTen and SparTen-mp use 32 CUs. It is worth noting that there is no on-chip global data buffer in SparTen accelerator, and feature maps and filters in compute units are directly loaded from the off-chip DRAM. To study the performance difference caused by computing units rather than memory hierarchy, we add data buffers to SparTen and SparTen-mp, and constrain all three accelerators to use the same size of on-chip data buffer.

Figure 17 shows the area-normalized performance of three accelerators. Ristretto can consistently achieve the highest performance on all models. Compared to SparTen, the average speedup of Ristretto on 2bit, 4bit, 8bit, and mixed 2/4bit

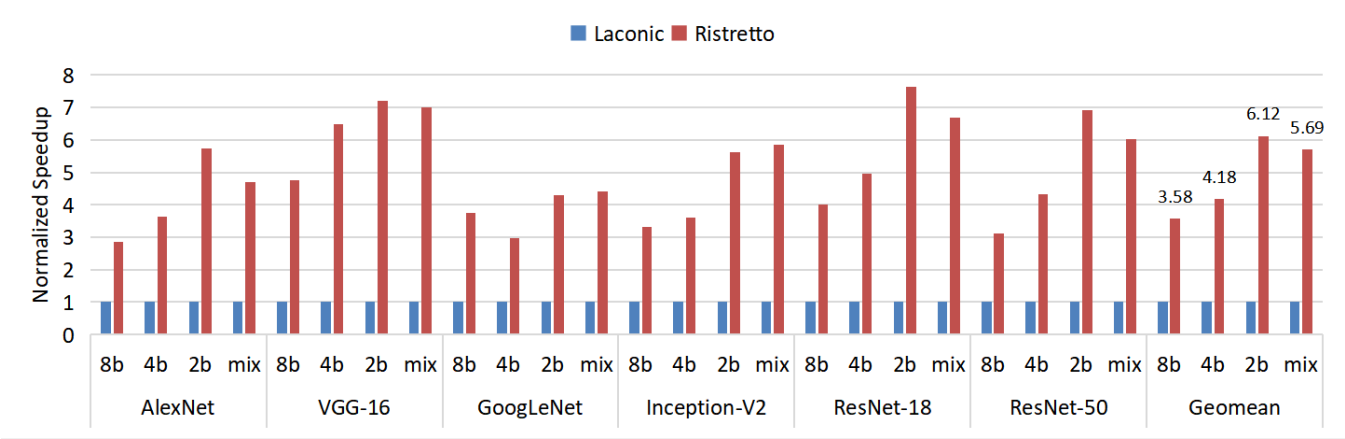


Figure 14: Comparison of performance between Ristretto and Laconic. All results are normalized to Laconic.

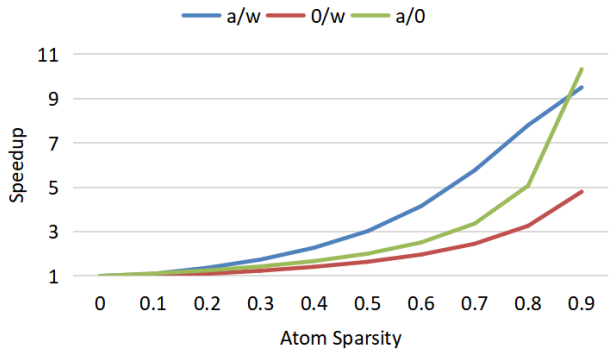


Figure 15: Performance of Ristretto with respect to different levels of atom sparsity. To precisely control the sparsity, this result is measured on the randomly generated sparse tensors using one compute tile.

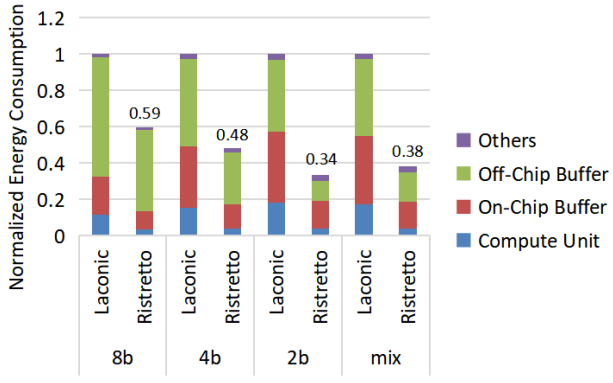


Figure 16: Comparison of energy consumption between Ristretto and Laconic.

models are 8.54×, 7.70×, 3.01×, and 8.25× respectively. For 4bit, 2bit, and mixed 2/4bit inference, the speedup of Ristretto over SparTen is significant. The reason lies in the constant throughput of SparTen's sparse dataflow, where only one non-zero pair of weight and activation can be extracted for calculation in each cycle, as detailed in Section II-B2. As expected, SparTen-mp can achieve a higher average performance than SparTen thanks to the bit-flexible multiplier and increased ability of non-zero data matching. However, due to the significant area overhead introduced by parallel inner-joins in each CU, Ristretto still outperforms SparTen-mp by a large margin.

E. Effectiveness of Load Balancing

In this experiment, we compare the proposed load balancing method (named "w/a balancing") with two baselines, i.e., "no balancing" and "w balancing." The no-balancing method continuously allocates input feature maps and corresponding kernels to compute tiles cyclically without considering the statistics of workloads. The w balancing method is also greedy-based. Unlike w/a balancing, w balancing groups input feature maps according to the number of non-zero weight atoms in their corresponding kernels. Figure 18 visualizes the balancing results of layer conv3_2 in 4bit ResNet-18, where 128 input feature maps and 32 filters are allocated to 32 compute tiles. It is clear that the difference in the number of workloads under w/a balancing is minimal.

Note that SparTen [17] employs a similar w balancing method to allocate filters to different CUs. However, we find that the improvement of w balancing over no balancing is negligible in Ristretto. The main reason is that the latency of Ristretto is closely related to the number of non-zero atoms in both weight and activation, making it more sensitive to either of the two. In contrast, the latency of a CU in SparTen is not directly related to the number of non-zero weights in the weight vector, so the statistic of the offline weights is a good proxy metric for load balancing.

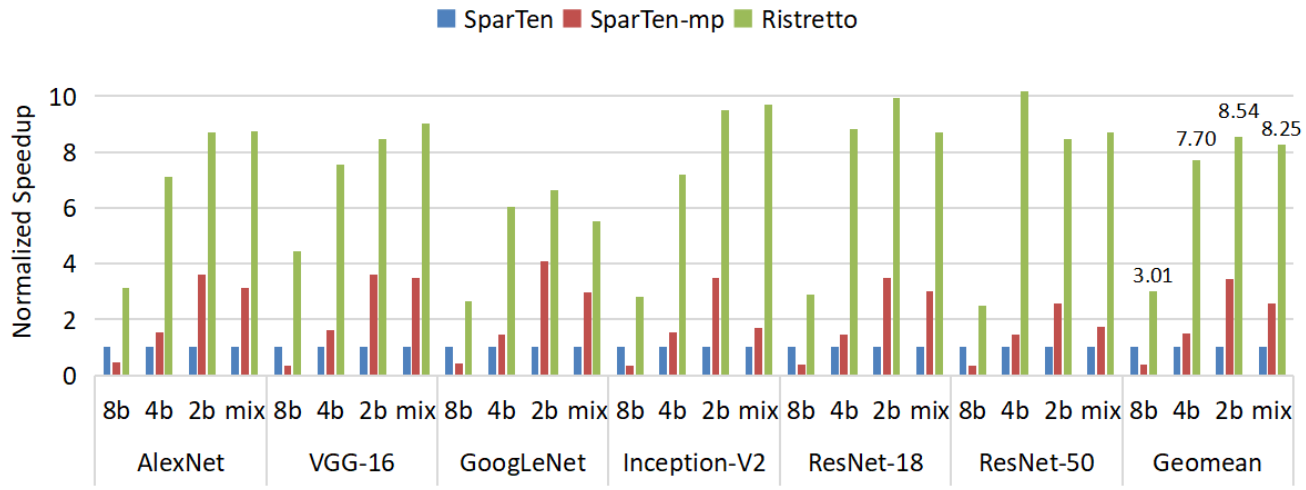


Figure 17: Comparison of performance between Ristretto, SparTen, and SparTen-mp. All results are normalized to SparTen.

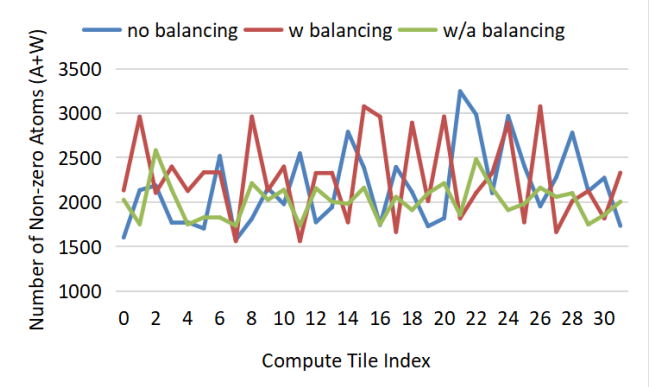
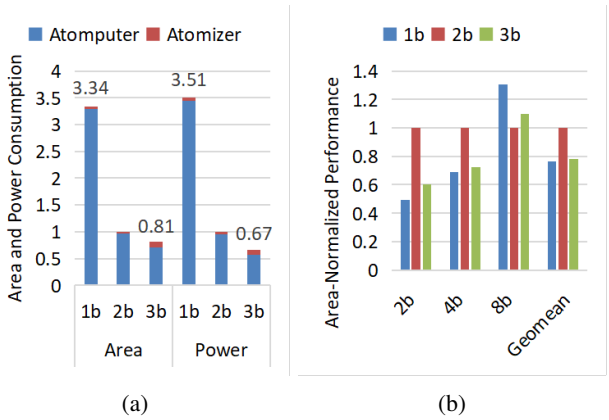


Figure 18: Visualization of load balancing results under different methods.



(a) (b)

Figure 19: (a) Comparison of area and power consumption of compute units under different atom granularity. (b) Comparison of average performance on DNN benchmark under different bit-width.

F. Impact of Atom Granularity

By default, Ristretto is designed under the 2bit granularity. In this experiment, we examine the impact of atom granularity on performance. We choose three atom configurations: 1bit, 2bit, and 3bit. For fair comparison, we constraint the Atomputers in all settings to have similar BitOps/cycle. Specifically, we configure all three accelerator to use 32 compute tiles. The numbers of multipliers in 1bit, 2bit, and 3bit Atomputers are 64, 16, and 7, respectively.

Figure 19a shows the area and power consumption of compute units in three accelerators. Compared to the 2bit Ristretto, the 1bit variant consumes 3.34× and 3.51× area and power. The reason is twofold. First, the range of shift offset is $\{0,1,2,3,4,5,6,7\}$ under 1bit granularity, which leads to a much larger shift unit for each 1bit multiplication in the Atomputer. Second, the increased number of accumulate registers accounts for a large portion of the area and power consumption. Although the 3bit variant consumes the least area and power, it suffers from the resource underutilization, especially for 2bit and 4bit models. Figure 19b shows the area-normalized performance on the DNN benchmark. Overall, the 2bit Ristretto can achieve the highest average performance.

VI. Conclusion

This paper presents Ristretto, an atomized processing architecture that can exploit both dual-sided sparsity and mixed-precision for CNN inference. By leveraging hierarchical data reuse, the low-level mixed-precision multiplication and high-level sparse convolution can be unified into a shared dataflow and efficiently performed through a condensed streaming computation. Extensive evaluations show that Ristretto consistently outperforms three state-of-the-art CNN accelerators. One limitation of Ristretto is that it currently only targets low-precision CNN inference. Since the deep learning community

has witnessed great success in Transformer and GNN based algorithms, how to extend Ristretto to support the most recent algorithms and large-scale training is critical to the practical value. We leave this for future work.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (Grant No.61972242). The authors would like to thank the support from Biren Technology.

References

[1] "Dnnsim," https://github.com/isakedo/DNNsim.

[2] "Simulator for bitfusion," https://github.com/hsharma35/bitfusion.

[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*, 2017, pp. 382–394.

[4] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[5] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[6] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2349–2358.

[7] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, 2019.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[9] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.

[10] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.

[11] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, and N. Djuric, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2090–2096.

[12] A. Delmas Lascorz, P. Judd, D. M. 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 the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 749–763.

[13] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "Gospa: An energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1110–1123.

[14] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Releq: A reinforcement learning approach for deep quantization of neural networks," *arXiv preprint arXiv:1811.01704*, 2018.

[15] Y. Fu, Y. Zhao, Q. Yu, C. Li, and Y. Lin, "2-in-1 accelerator: Enabling random precision switch for winning both adversarial robustness and efficiency," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 225–237.

[16] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[17] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 151–165.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[19] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[20] ——, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of*

the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[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] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[25] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2146–2156.

[26] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[27] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[28] D. Kim, J. Ahn, and S. Yoo, "Zena: Zero-aware neural network accelerator," *IEEE Design & Test*, vol. 35, no. 1, pp. 39–46, 2017.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[30] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[31] 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 Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2018.

[32] F. Li, G. Li, Z. Mo, X. He, and J. Cheng, "Fsa: A fine-grained systolic accelerator for sparse cnns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3589–3600, 2020.

[33] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[34] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 694–701.

[35] Q. Lou, F. Guo, L. Liu, M. Kim, and L. Jiang, "Autoq: Automated kernel-wise neural network quantization," *arXiv preprint arXiv:1902.05690*, 2019.

[36] H. Lu, X. Wei, N. Lin, G. Yan, and X. Li, "Tetris: re-architecting convolutional neural network computation for machine learning accelerators," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[37] C. McCool, T. Perez, and B. Upcroft, "Mixtures of lightweight deep convolutional neural networks: Applied to agricultural robotics," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1344–1351, 2017.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[39] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017.

[40] 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 *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[41] S. Ryu, H. Kim, W. Yi, and J.-J. Kim, "Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[43] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[44] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 304–317.

[45] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks,"

in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[46] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

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

[48] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 1010–1021.

[49] X. Sun, R. Panda, C.-F. R. Chen, A. Oliva, R. Feris, and K. Saenko, "Dynamic network quantization for efficient video inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7375–7385.

[50] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[51] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 3, pp. 568–579, 2018.

[52] M. Van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning," *Advances in neural information processing systems*, vol. 33, pp. 5741–5752, 2020.

[53] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[54] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2078–2087.

[55] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency," in *European Conference on Computer Vision*. Springer, 2020, pp. 259–277.

[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, no. 2, pp. 465–477, 2019.

[57] 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 neural network inference," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 636–647, 2020.

[58] 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 *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[59] 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 *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 15–28.