This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

RAMAN: A Re-configurable and Sparse tinyML Accelerator for Inference on Edge

Adithya Krishna, Srikanth Rohit Nudurupati, Chandana D G, Pritesh Dwivedi, André van Schaik, Mahesh Mehendale and Chetan Singh Thakur*

***Abstract*—Deep Neural Network (DNN) based inference at the edge is challenging as these compute, and data-intensive algorithms need to be implemented at low cost and low power while meeting the latency constraints of the target applications. Sparsity, in both activations and weights inherent to DNNs, is a key knob to leverage. In this paper, we present RAMAN, a *R*e-configurable and sp*A*rse tiny*M*L *A*ccelerator for infere*N*ce on edge, architected to exploit the sparsity to reduce area (storage), power as well as latency. RAMAN can be configured to support a wide range of DNN topologies - consisting of different convolution layer types and a range of layer parameters (feature-map size and the number of channels). RAMAN can also be configured to support accuracy vs. power/latency tradeoffs using techniques deployed at compile-time and run-time. We present the salient features of the architecture, provide implementation results and compare the same with the state-of-the-art. RAMAN employs novel dataflow inspired by Gustavson's algorithm that has optimal input activation (IA) and output activation (OA) reuse to minimize memory access and the overall data movement cost. The dataflow allows RAMAN to locally reduce the partial sum (Psum) within a processing element array to eliminate the Psum writeback traffic. Additionally, we suggest a method to reduce peak activation memory by overlapping IA and OA on the same memory space, which can reduce storage requirements by up to 50%. RAMAN was implemented on a low-power and resource-constrained Efinix Ti60 FPGA with 37.2K LUTs and 8.6K register utilization. RAMAN processes all layers of the MobileNetV1 model at 98.47 GOp/s/W and the DS-CNN model at 79.68 GOp/s/W by leveraging both weight and activation sparsity.**

***Keywords*—Convolutional neural networks (CNNs), deep learning, hardware acceleration, sparse processing.**

I. Introduction

Deep neural networks (DNNs) have become ubiquitous in various cognition and learning problems [1]–[4]. DNNs are often computed in the cloud, and the inferred result is delivered back to an edge node, introducing delay owing to constrained communication bandwidth. Thus, the deployment of DNNs directly on edge has recently attracted more attention since it offers many inherent benefits, including privacy, bandwidth savings, and latency reductions. However, the computation on an edge device poses numerous challenges due to power, memory, and resource constraints. GPUs and CPUs conventionally used in cloud platforms are extremely power intensive and area inefficient and thus cannot be directly deployed on edge. A promising avenue to pursue in this respect is the development of an accelerator tailored for neural computations on edge. A customized hardware design for neural networks provides an opportunity to optimize dataflow, memory access and exploit network sparsity to overcome edge computing bottlenecks.

Sparsity is an inherent attribute in most DNNs which can be leveraged on hardware. It is estimated that approximately 40% of the input activations (IAs) and 50% of weights (Ws) are sparse in the MobileNet model [5], [6] trained on the Imagenet dataset [1] with the hardware-aware pruning strategy presented in this paper. Other well-established networks like AlexNet [2], VGG-16 [7] , and ResNet-50 [8] show similar sparsity statistics. Thus sparsity induces a lot of ineffectual zero computations that can be skipped. Aggressive pruning strategies can further reduce computations if a slight reduction in inference accuracy is acceptable. In this direction, several attempts have been made in the literature to maximize the sparsity by zeroing out low-magnitude weights [9]–[11]. In addition to weight sparsity, the commonly used rectified linear unit (ReLU) activation function clamps all negative activation values to zero, resulting in sparse output activations (OAs), which become IAs to the subsequent layer. Even though exploiting weight sparsity in hardware has been thoroughly investigated, leveraging activation sparsity in hardware efficiently is a topic of research and needs further exploration. This disparity is primarily because of the fact that it is possible to enforce structured sparsity in weights during training by pruning in a hardware-aware fashion (by knowing the underlying hardware architecture and the dataflow) that maximizes the overall hardware utilization and efficiency. However, the activation sparsity is unstructured and highly challenging to leverage on hardware as the data varies dynamically and depends on the environment [12].

A. Related Work

Early work in this domain used the indirection principle to exploit sparsity in one of the operands meaning in either weights or activations, but not both. Cnvlutin [16] exploits sparsity in IA by storing them in a compressed format as value and index pairs. The index information of the non-zero activations is used to perform in-direct memory access to extract dense weights. In another work, Cambricon-X [14]

A. Krishna, S. R. Nudurupati, D G Chandana, P. Dwivedi, M. Mehendale and C. S. Thakur (Email: csthakur@iisc.ac.in) are with the Department of Electronic Systems Engineering, Indian Institute of Science, Bangalore - 560012, India; and A. van Schaik is with the International Centre for Neuromorphic Systems, The MARCS Institute, Western Sydney University, Australia. A. Krishna is currently with the International Centre for Neuromorphic Systems, The MARCS Institute, Western Sydney University, Australia.
This work was supported by the Department of Science and Technology of India under Grant DST/IMP/2018/000550 and Pratiksha Trust Grant FG/SMCH-22-2106.
*Corresponding author

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

TABLE I: Qualitative comparison of RAMAN with prior works in terms of sparsity leveraged in activations (A) and weights (W), pruning, activation memory optimization by IA/OA overlapping, flexibility in quantization, layers supported, implementation platform, and application. W-HAP denotes Hardware-aware pruning of weights. The supported layers are denoted as follows: C: standard convolutions (CONV), D: Depth-wise (DW), P: Point-wise (PW), Pl: Pooling, S: Shift and F: Fully Connected (FC).

Accelerator	Sparsity				Pruning		Act. Mem. Opt.	Flexible Quant.	Layers Supported	Platform	Application
	A	W	Gate 0	Skip 0	A	W-HAP					
Eyeriss [13]	✓	✗	A	✗	✗	✗	✗	✗	C	ASIC	Standard
Cambricon-X [14]	✗	✓	✗	W	✗	✗	✗	✗	C+F+Pl	ASIC	Standard
SCNN [15]	✓	✓	A+W	A+W	✗	✗	✗	✗	C	ASIC	Standard
Cnvlutin [16]	✓	✗	✗	A	✗	✗	✗	✗	C	ASIC	Standard
Sticker [17]	✓	✓	A+W	A+W	✗	✗	✗	✗	C+F	ASIC	Standard
EIE [18]	✓	✓	✗	A+W	✗	✗	✗	✗	F	ASIC	Standard
SNAP [19]	✓	✓	✗	A+W	✗	✗	✗	✗	C+F	ASIC	Standard
NullHop [20]	✓	✗	✗	A	✗	✗	✗	✗	C+F+Pl	FPGA	Standard
McDanel et al. [21]	✓	✓	A	W	✗	✓	✗	✗	C+P+F	FPGA	Standard
SpWA [22]	✗	✓	✗	W	✗	✗	✗	✗	C	FPGA	Standard
Lu et al. [23]	✗	✓	✗	W	✗	✗	✗	✗	C	FPGA	Standard
Yin et al. [24]	✗	✓	✗	W	✗	✗	✗	✗	C+D+P	FPGA	Standard
Xie et al. [25]	✗	✓	✗	W	✗	✗	✗	✗	C+D+P+F+Pl	FPGA	Standard
Zhu et al. [26]	✓	✓	A	W	✗	✓	✗	✗	C+F	FPGA	Standard
Sense [27]	✓	✓	A+W	A+W	✗	✓	✗	✗	C+Pl+F	FPGA	Standard
Meng et al. [28]	✗	✓	✗	W	✗	✓	✗	✗	C+D+P+F+Pl	FPGA	Standard
Zhang et al. [29]	✓	✓	✗	A+W	✗	✗	✗	✗	C	FPGA	Standard
Yin et al. [30]	✗	✓	✗	W	✗	✗	✗	✗	C+D+P	FPGA	Standard
Lu et al. [31]	✓	✓	A	W	✗	✗	✗	✗	C+F+Pl	FPGA	Edge
Choi et al. [32]	✗	✗	✗	✗	✗	✗	✗	✗	D+P	FPGA	Edge
FitNN [33]	✗	✗	✗	✗	✗	✗	✗	✗	D+P+Pl	FPGA	Edge
Hao et al. [34]	✗	✗	✗	✗	✗	✗	✗	✗	C+P+D+Pl	FPGA	Edge
Synetgy [35]	✗	✗	✗	✗	✗	✗	✗	✗	S+P+Pl	FPGA	Edge
Wu et al. [36]	✗	✓	✗	W	✗	✓	✗	✗	C+Pl+F	FPGA	Edge
RAMAN	✓	✓	A+W	A+W	✓	✓	✓	✓	C+D+P+F+Pl	FPGA	Edge

uses the same indirection principle, assuming sparse synaptic connections. The input neurons with non-zero synaptic connections are transferred to a computational unit to perform MAC (multiply-accumulate) operations. These architectures are inefficient as they are intended to exploit sparsity present in only one of the operands (W or IA). The dense processing core can quickly adapt to accommodate one operand sparsity by indirect memory access. EyerissV1 [13] is one of the early works investigating activation sparsity to save power by employing data-gating logic. This method improves energy efficiency, but it does not reduce latency. EyerissV2 [37] exploits sparsity further in both IA and W by preserving the data in compressed form all the way to the computational element and adopting a row stationary dataflow like EyerissV1. EyerissV2 employs an external DRAM for storing the activations and parameters and requires a complex hierarchical mesh network to route the data to different computational elements on-chip. We avoid such complexities in RAMAN as we target tinyML edge applications with all on-chip memory implementation.

Computer architects face two significant challenges in designing a sparse neural network accelerator leveraging both IA and W sparsity. First is the front-end challenge, where a sufficient number of non-zero IA and W pairs stored in a compressed format must be transferred to the computational unit to keep the MAC utilization high. Second, the back-end challenge where the partial-sum (Psum) addresses have to be aligned and immediately reduced within the computational element before a writeback. If the addresses are not aligned, then the Psums cannot be reduced, leading to significant writeback traffic and access contention. SCNN [15] and Sticker [17] try to overcome the front-end challenge; however, these works are plagued by the back-end problem. SCNN [15] uses channel-last dataflow to maximize the multiplier utilization at the cost of high output traffic and access contention. Sticker [17] also uses channel-last dataflow and adopts a two-way set associative PEs to reduce memory access contention and collisions. However, since this strategy requires data re-ordering to prevent collisions, which are done offline using the CPU, it ultimately defeats the whole purpose of latency and energy reductions that sparsity offers. SNAP [19] tries to overcome the back-end problem by employing channel-first dataflow where non-zero W and IA data are organized and processed in the channel dimension first and subsequently in the pixel dimension. This ensures that the Psums are locally reduced within the multiplier array, thus decreasing the writeback traffic. However, pairs of non-zero IA and W of matching channels have to be extracted, and they utilize associative index matching (AIM) units and sequence decoders to do the same. The AIM uses $N \times N$ comparators to match W and IA channel indices, making it highly inefficient as the area and power grow quadratically with N. EIE [18] exploits both IA and W sparsity but only supports fully connected (FC) layer making it incompatible to run convolution operations. In addition, several FPGA-based implementations have been proposed in the literature [20]–[26], [31]–[35], [38]–[48], discussed in detail in Section IV-C. Table I presents a qualitative comparison between RAMAN and previous works. RAMAN offers an extensive range of essential architectural features tailored for tinyML applications, setting it apart from

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

other approaches that only support a limited subset of these features.

Several recent implementations of neural networks have integrated sparsity optimizations into their designs [27]–[30], [36], [49], [50]. The work by [27], [29], [36] demonstrates a method for exploiting weight sparsity within Systolic arrays. Wu et al. [36] introduce a fine-grained pruning scheme and compression strategy tailored for edge computing. Sun et al. [27] present an approach to balance input feature maps and weights across processing elements through channel clustering and co-designed load-balancing weight pruning. Zhang et al. [29] propose a novel scheduling strategy utilizing row stationary dataflow to exploit sparse kernels (weights) and address low processing element utilization caused by load imbalance. Meng et al. [28] put forward a dense/sparse-aware CNN accelerator aimed at achieving high processing element utilization and reconfigurability. Yin et al. [30] propose a CNN accelerator based on block sparse weight pruning; however, the architecture is constrained to leveraging weight sparsity and does not consider activation sparsity.

B. Our Contributions

This work presents a re-configurable and sparse deep neural network accelerator that exploits both IA and W sparsity. To address the front-end challenge, we employ Gustavson's inspired dataflow [51] with the hardware-aware balanced weight pruning strategy to keep workload uniform across all the PEs and maintain high MAC utilization. The back-end issue is resolved by reducing Psum locally within a processing element (PE) array; this reduces the writeback bandwidth and eliminates memory access contention because only the final result is sent back to memory. Latest advancements in quantization techniques [52] in DNNs have eliminated the need for floating-point arithmetic (during inference), and simple energy-efficient fixed-point arithmetic has proven adequate to achieve reasonable accuracy. Despite having a highly efficient computational realm, today's design has a memory bottleneck which is evident from [53]. Memory access (especially DRAM) is orders of magnitude more expensive than conventional arithmetic operations. The most state-of-the-art DNN accelerators, except for EIE, utilize an external DRAM to store IAs and Ws and pay the penalty of unprecedented memory access cost. Since our architecture is targeted at tinyML edge computing applications with stringent energy budgets, using external DRAM and paying high energy costs becomes untenable. Thus, in this work, we do away with the requirement for a DRAM and only use the on-chip SRAM to store the model weights and activations. However, the memory size of the on-chip SRAM is limited due to area constraints, which necessitates model optimizations to fit modern networks such as MobileNets [5], [6] in SRAM. In this work, we introduce a hardware-aware pruning strategy to shrink the model size and intelligent memory scheduling to minimize peak activation memory to accommodate both model and activations on-chip. In summary, the following are the contributions and features of the RAMAN architecture presented in this paper:

Sparsity: RAMAN exploits both IA and W sparsity to achieve higher throughput and energy efficiency compared to most prior works focusing on sparsity in one of the operands [14], [16], [20], [22], [23]. Sparsity is leveraged in storage and computation. Input activations are stored in compressed form in a cache, and a simplified version of the compressed sparse row (CSR) format [54] is used for storing weights in both on-chip global memory and cache. In computation, sparsity is leveraged in two ways; for the layers with the maximum computational density (such as PW), RAMAN can skip the processing cycles with zero data, improving throughput and energy efficiency. However, skipping processing cycles won't reap any benefit in the layers with relatively low computational density (such as DW); in such cases, the design only data-gates the cycles with zero data but does not skip them reducing architectural complexity. Furthermore, the activation sparsity engine (ASE) designed to leverage IA sparsity is developed at a low cost (<1% of LUTs, 3% registers and 2% memory) compared with other prior implementations [15], [17], [19], [20]. Furthermore, we present a hardware-aware balanced weight pruning strategy (c.f III-C) that reduces memory storage, access, and processing latency by software-hardware co-optimization.

Programmability: RAMAN supports traditional CNN models, separable convolution models constituting depth-wise (DW) and point-wise (PW) layers, max pooling, average pooling, and fully connected layers. Most prior implementations in the literature cater to either a specific network topology like MobileNet [45]–[47] or YOLO [42], [43], or a particular type of layer [22], [23], [38], [41]. The unsupported layers are often executed offline, making the overall system-level computation inefficient. In many cases, these implementations demand FPGA re-synthesis to adapt to different network topologies. On the other hand, RAMAN provides an instruction memory and a dedicated instruction set for storing and programming diverse network topologies. This capability empowers RAMAN to execute various layer configurations without requiring FPGA design re-synthesis.

Dataflow: RAMAN incorporates novel dataflow inspired by Gustavson's algorithm [51] to reduce memory access for the PW layer, which is the most computationally intensive layer. The dataflow reduces the memory access cost of a single PE by 1.9x and 6.5x compared to output stationary [35] and input/weight stationary [21] dataflows. Additionally, the NoC is reconfigured to support weight stationary dataflow for DW and standard convolutional (CONV) layers. This hybrid dataflow architecture, enabled by dynamic NoC reconfiguration, maps the computation of a particular layer to its optimum dataflow to achieve maximum energy efficiency and minimum data movement cost.

Peak Activation Memory Reduction: The state-of-the-art accelerators logically partition the IAs and OAs inside the memory, where the total activation memory is the sum of IA and OA memory spaces. This logical partition is eliminated in our work, and the OAs are directly overwritten onto the IA memory space, lowering the peak activation memory requirements by up to 50%. RAMAN employs an intelligent memory scheduling scheme presented in Section III-A to

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

prevent memory collision issues that ensue after removing the logical partition.

Run-time Activation Pruning (RAP): RAMAN employs run-time hardware-aware activation pruning to enhance activation sparsity, and the architecture effectively leverages this strategy to increase throughput and energy efficiency. Notably, this approach is one-of-a-kind, and our preliminary offline experiments substantiate the network's resilience to these pruning techniques. The RAP contributes to an additional latency reduction of around 12-16%, IA cache access reduction by 8-10%, and parameter cache reads reduction by 13-21%. Section II-B provides a detailed description of RAP.

Flexible quantization: RAMAN supports variable precision quantization of both weights and activations supporting 2b, 4b, and 8b precisions. Unlike most current approaches that adhere to a fixed quantization precision (usually 8b or 16b for all layers) [20], [22], [38], [47], [48], RAMAN's architecture is programmable for dynamic precision adjustment at the layer level to meet the required accuracy, storage, and latency target. For instance, the initial layers could employ 8b IAs and Ws, while the subsequent layers may adopt 4b or 2b IAs and Ws. This approach effectively mitigates latency and optimizes storage and memory access, all while adhering to accuracy constraints.

The rest of the paper is organized as follows: Section II presents the architecture of the RAMAN accelerator. The architectural features that make RAMAN feasible on edge are highlighted in the Section III. Section IV provides implementation results, and Section V concludes this article.

II. System Architecture

A. Top-Level Architecture

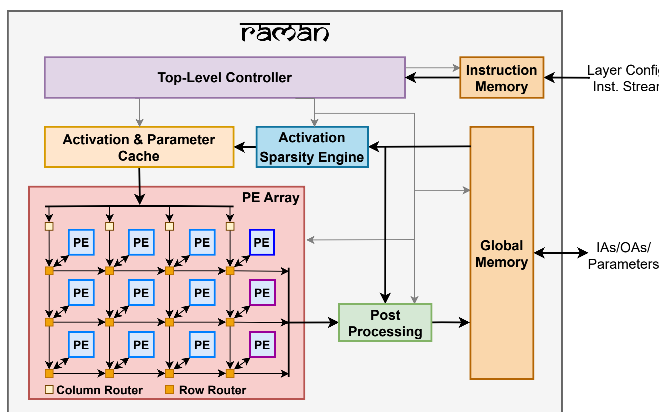


Fig. 1: Top-level architecture.

Fig. 1 shows the top-level architecture of the RAMAN accelerator system. The architecture can be broadly categorized into:

Compute: The compute sub-system comprises a processing element (PE) array, an activation sparsity engine (ASE) and a post-processing module (PPM). The multiply and accumulate (MAC) operations are performed by 12 spatial PEs in the PE array, which are arranged in a 3x4 rectangle. The ASE leverages input activation sparsity to minimize latency and power by skipping ineffectual zero computations. The PPM performs ReLU, quantization, pooling, bias and residual addition operations. Sections II-B-II-C gives a detailed description of the PE array and ASE. A detailed description of PPM is provided in Section III of the supplementary document.

Memory: The memory sub-system comprises an on-chip global memory (GLB-MEM), activation and parameter cache, and instruction memory. GLB-MEM stores the parameters of all layers and the input and output activations of a specific layer. Cache exploits temporal reuse in parameters and activations to reduce energy-expensive data access to the large on-chip GLB-MEM. We employ a three-level memory hierarchy composing GLB-MEM, cache and RFs (inside PEs). The reg-file is used for Psum reduction locally inside the PE to overcome the back-end challenges. In addition, we have the instruction memory to store layer configuration instructions of individual layers. A detailed description of the GLB-MEM and cache is provided in Section V of the supplementary document.

Control: The control sub-system encompasses a top-level controller to coordinate: 1) data transfer between the GLB-MEM and cache; 2) traffic between the cache and PE array utilizing the NoC; 3) traffic between the PE array, PPM and GLB-MEM; 4) operations of the ASE, PE array, NoC, and PPM. We do not use a separate controller for each of the 12 PEs since they are all identical and operate in lockstep, meaning that their processing states are equivalent with regard to one another. The top-level controller is responsible for issuing the control signals to all 12 PEs. A detailed description is provided in Section IV of the supplementary document.

B. Activation sparsity Engine (ASE)

The activation sparsity engine shown in Fig. 2 serves two purposes. First, it reduces the cycles needed to write a block of data from the GLB-MEM to cache through ping-pong-based shift registers. Second, it aids in exploiting activation sparsity. Sparsity is leveraged in two ways: 1) by data gating the cycles with zero data and disabling the memory read to prevent the datapath from switching, thereby reducing dynamic power; 2) by skipping the processing cycles entirely to improve energy efficiency and throughput. IAs are routed through the ASE to record the position of zeros, and this information is utilized during computation to either gate or skip zero computation, depending on the layer under execution. The layers with low computing density (e.g., DW) adopt the gating strategy, and the layers with relatively high computational density (e.g., PW) employ the zero skipping technique in addition to data gating. This hybrid approach reduces architectural complexity since imposing zero skipping during DW execution has no positive impact on performance. The ASE consists of five major blocks:

Shift Register Bank: Consists of six parallel shift registers (SRs), with even and odd SR pairs ping-ponging to minimize the time required to write a block of data from the GLB-MEM to cache. To begin with, all even SRs are in input mode, while all odd SRs are in shift/output mode, and the functionality is inverted in the following epoch. In the input mode, we load

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

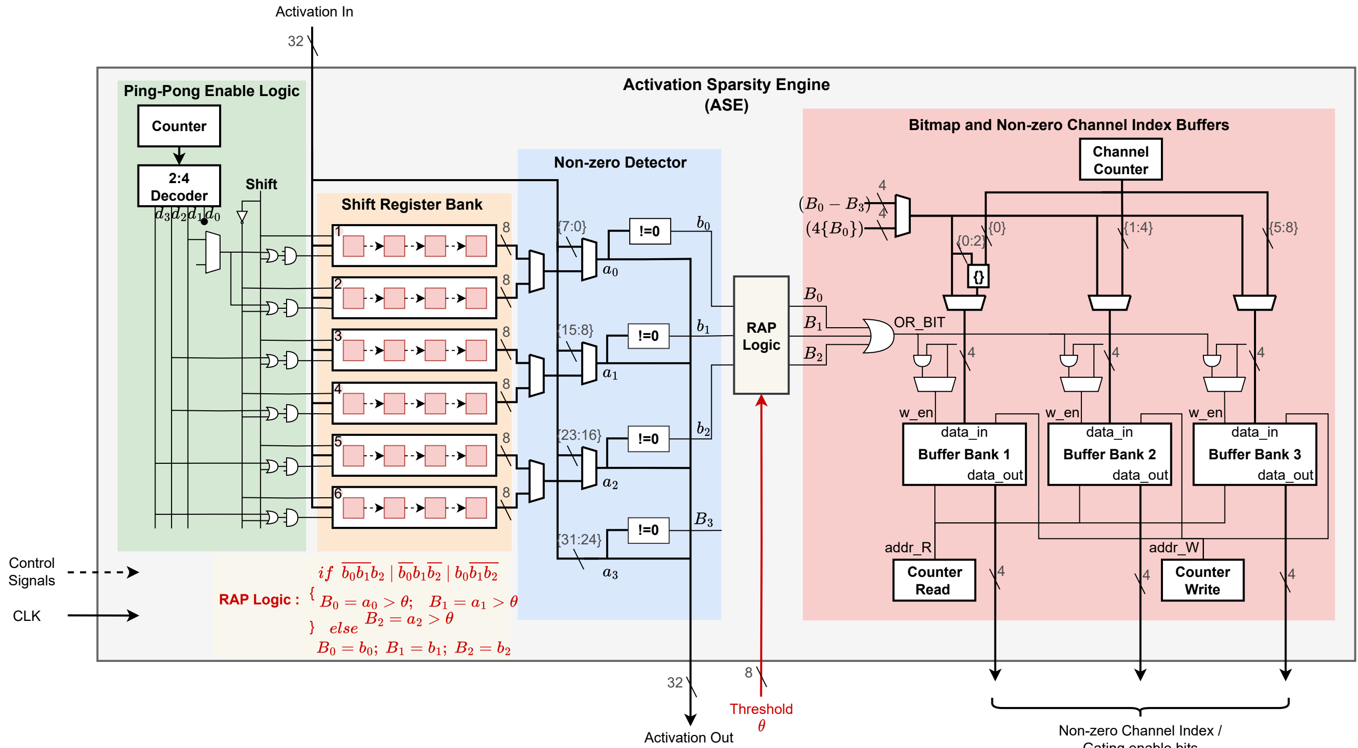


Fig. 2: Activation sparsity engine architecture.

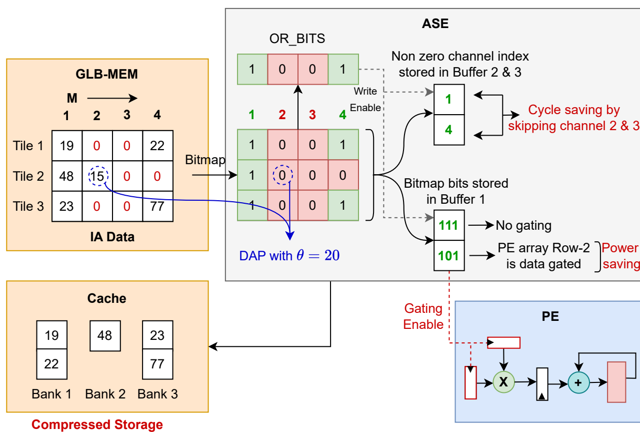


Fig. 3: ASE Illustration for the PW layer.

32b activations parallelly into four registers of an SR. In the output/shift mode, 8b data is serially shifted into the non-zero detector block.

Ping-Pong Enable Logic: Comprises a counter and 2:4 decoder to activate appropriate SRs in the shift register bank. The contents of the SRs are shifted when the $shift$ control signal is asserted.

Non-Zero Detector: The IAs read from the shift register bank are compared with zero to generate bitmap (b_0 to b_2, B_3), which is 1 when the IA value is not equal to 0 and 0 otherwise, thus recording the position of zeros. The bitmap bits (b_0 to b_2) are sent to the run-time activation pruning (RAP) logic to perform activation pruning and generate a new set of bitmap bits (B_0 to B_2). Then the bits (B_0 to B_2) obtained from RAP are 'OR'ed to generate OR_BIT, which is required for zero skipping in the PW layer. The bitmap bits, along with the OR_BIT, are fed to the succeeding bitmap and non-zero channel index buffer module.

Run-time Activation Pruning (RAP) logic: As the name suggests, the RAP performs activation pruning during inference to improve the throughput and minimize computation latency. In our implementation, IA in the PW layer is tiled with each sizing $1 \times M$, and three rows of the PE array simultaneously process three such tiles, i.e., $3 \times M$ block of IA is processed in a single epoch. However, if all activations in a column of $3 \times M$ block are zeros, that particular input channel or column is skipped entirely during the computation. Suppose there is only one non-zero activation in a column and the rest are 0; that activation value is pruned if it's below the predetermined threshold (θ) so that the column may be skipped during processing. The threshold value is precomputed during the training phase based on the input and hidden layers activation distribution. A pseudo-code of the RAP logic is shown in Fig. 2. When any one of the bitmap bits (b_0 to b_2) is 1, and the remaining are $0s$, then the activation values (a_0 to a_2) are compared with the threshold. If the value a_k is lesser than the threshold, then the corresponding bitmap bit B_k is made 0. If a column has more than one non-zero element, then the input bitmap bits to the RAP (b_0 to b_2) are retained. RAP reduces latency by 12-16%, IA cache access by 8-10%, and parameter cache reads by 13-21%, and offers accuracy vs energy/latency tradeoff which can be configured at run-time.

Bitmap and non-zero channel index buffer: It consists of

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

three buffers and a channel counter. When the OR_BIT is 1 in the PW layer, we save the corresponding input channel number obtained from the channel counter in the buffer. This information is useful during the computation as we perform computation only on input channels recorded in the buffer and skip the rest. In other layers (such as DW and CONV), we store the bitmap bits in buffers, which are later used to enable/disable the appropriate data gating registers in PEs (c.f Fig. 4).

The IAs are stored in a compressed format in the cache using the five blocks discussed above. The bitmap bits and the input channel numbers saved in buffers aid in power saving through data-gating and latency reduction through cycle skipping. Area overhead of the ASE in terms of LUTs (lookup tables), registers and memory utilization is insignificant, as demonstrated in Section IV.

1) ASE illustration: Fig. 3 illustrates the working of ASE for the PW layer. The three IA tiles are loaded into the shift register bank, and the non-zero detector module generates the bitmap. In Fig. 3, the second input channel of tile-2 is pruned as the value is less than the threshold value of 20, inside the RAP block. The bitmap matrix is column-wise ORed to obtain the OR_BITS vector. The columns 2 and 3 are skipped during computation as all elements are zero post RAP. The channel indices 1 and 4 are cached in buffers 2 and 3, and they are subsequently utilized as addresses to retrieve the corresponding input channel weight from the cache during computation. Additionally, the column-wise bitmap is stored in buffer 1 for non-zero channel indices (1 and 4), which is accessible during computation cycles to activate or disable the PE's data registers as depicted in Fig. 4. In the example shown, for the first input channel, all the rows of the PE array are active as the bitmap is $'111'$ for that particular column, and in the fourth channel, row-2 of the PE array is deactivated as the bitmap is $'101'$. Additionally, only the non-zero elements of the IA matrix are saved in cache banks utilizing the bitmap data. Just the bitmap is saved in buffers for other layers, not channel indices, as they only facilitate data gating and not cycle skipping.

C. PE Array

The PE array comprises 12 processing elements spatially distributed along three rows and four columns. It is coupled with the Network-on-chip to route the data among different PEs.

1) Network-on-chip: The network-on-chip handles data delivery between the cache, the PE array, and between different PEs. The following are the NoC's responsibilities: 1) Support various data delivery patterns needed by different layers; 2) Provide sufficient data bandwidth for parallel processing to keep the PEs active and improve utilization; 3) Handle various strides and padding, and 4) Leverage spatial data reuse to increase energy efficiency.

To accomplish this, we employ a network of row and column routers. The column routers distribute a cached data block across four columns of the PE array, while the row router further splits the incoming packets and delivers input to the PE in a specific row. Furthermore, the output from the PEs is directed to the post-processing module via the row routers. Section II-D goes into greater detail on the various data delivery patterns to perform different layer computations. Additionally, NOC can adapt to a wide range of bandwidth requirements wherein it can provide a high bandwidth IA data from the cache to keep the PEs busy when there is limited IA reuse (for DW layers); when the IA reuse is high (in PW and FC layers), it reduces IA data bandwidth and increases W bandwidth.

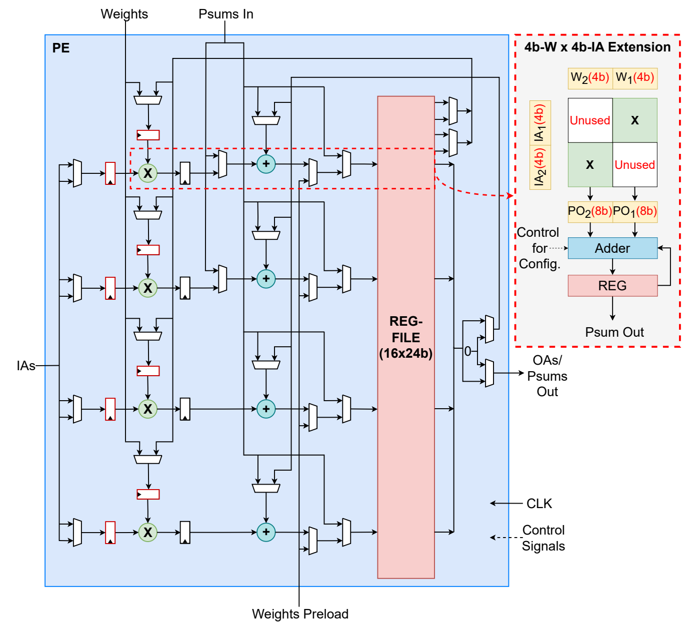


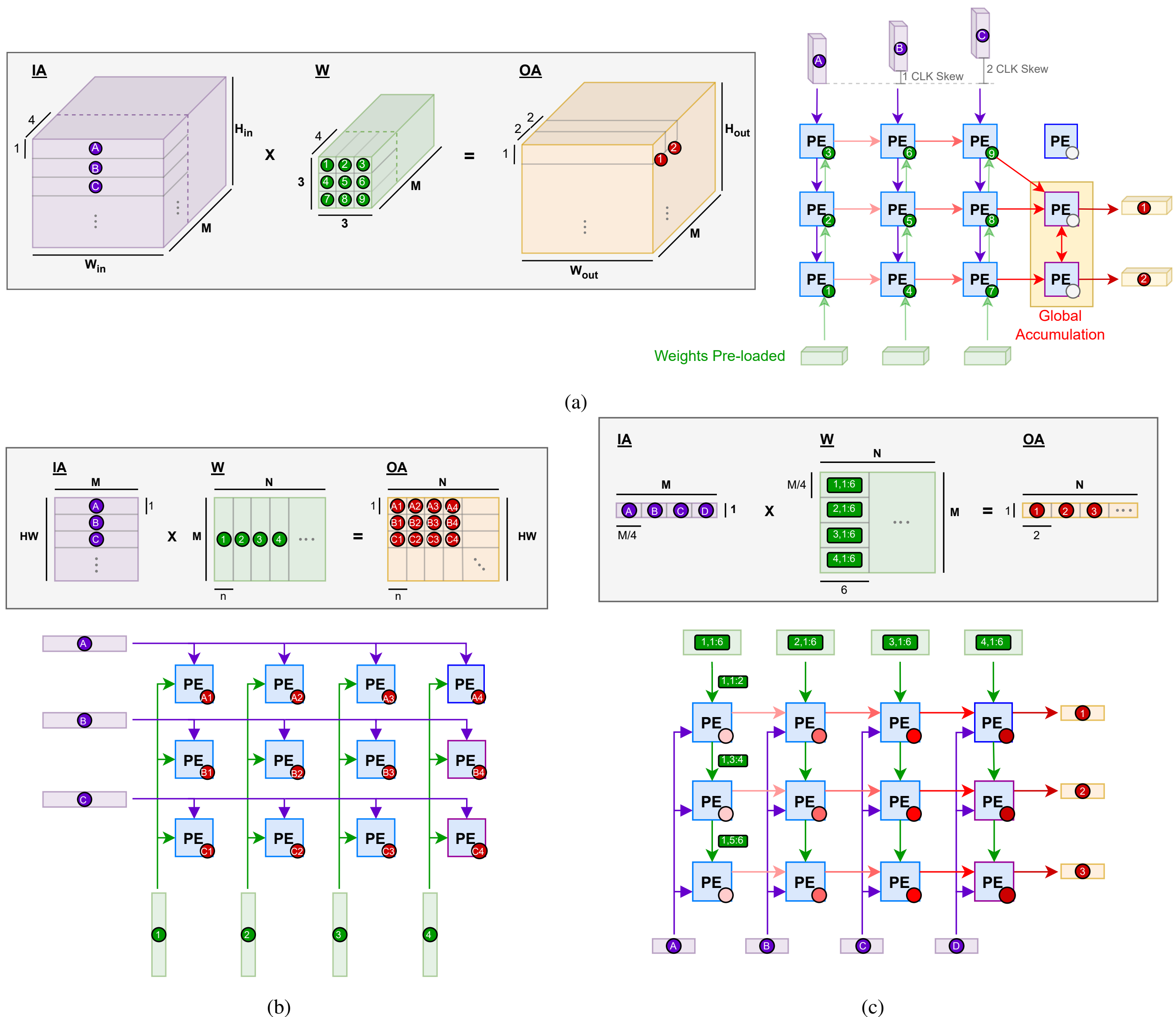
Fig. 4: Processing Element Architecture (Type-1).

2) Processing Element: Fig. 4 shows the architecture of the PE. We employ three types of PEs to support dataflow flexibility for different layer types. The architectures of the other two PE types constituting the last column of the PE array are presented in Figs. 9, 10 of the supplementary document. The fundamental elements of all PE configurations are an 8b multiplier, a 24b adder constituting MAC (Multiply and Accumulate Unit), and a reg-file. 8b W and IAs are provided as input and accumulated using the MAC unit, and a 24b Psum is saved in the RF. Offline experiments show that the Psum could fit within the 24b range. The type-2 and type-3 PEs use four-ported RF (4 input & output ports), while the RF in type-1 PE has four input and eight output ports, each with a width of 24b and a depth of 16. Thus, the RF can be addressed using 4b. The Psums are locally reduced inside the PE array, and the final result is written back to the memory after quantization in PPM, thereby reducing the writeback bandwidth and eliminating memory contention.

Power saving: PE implements data-gating logic to leverage zeros in the IAs for saving processing power in the layers with low computational density, such as DW. The red-bordered registers in Fig. 4 are data-gated, and if a zero IA value is detected (provided by the bitmap bits stored in ASE buffers), then the gating registers are disabled to prevent the MAC datapath from switching.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832



(b)

(c)

Fig. 5: Dataflow configuration across an array of PEs in RAMAN for (a) DW computation (b) PW computation and (c) FC computation.

SIMD support: PE supports SIMD processing, evident from Fig. 4, by performing four MAC operations per cycle, thereby speeding up the processing four times. In addition, SIMD processing also enables W and IA reuse, thereby reducing the number of memory accesses. RF has four write ports to simultaneously write the Psums from the four MAC units.

Variable precision: The PE datapath supports variable precision Ws and IAs. The reconfiguration of the data path for the 4b-W and 4b-IA using the same 8b datapath is shown in Fig. 4. First, two 4b-Ws and 4b-IAs are packed as an input to the 8b multiplier to compute two 4b multiplications doubling the throughput. Next, the accumulator reduces the ensuing partial outputs (PO_1 and PO_2), each 8b wide. Similarly, for the 2b-W and 2b-IA case, the multiplier simultaneously does four 2b multiplications boosting throughput by 4x.

D. Dataflow

1) DW: DW layer uses a weight stationary systolic dataflow where the W remains static during the computation. IAs of dimension $(H_{in} \times W_{in} \times M)$ are partitioned into tiles of size $(1 \times W_{in} \times 4)$ and Ws of dimension $(3 \times 3 \times M)$ are partitioned into tiles of size $(3 \times 3 \times 4)$. Then three IA tiles are streamed from the cache to the PE array from top-to-bottom, and the Psum obtained is spatially reduced along the PE columns from left-to-right. A global accumulation is performed by the last column of PEs to obtain a final accumulated result. This mapping allows the reuse of IA along a column and Psums are spatially reduced inside the PE array. Since each PE supports SIMD with four MAC operations per cycle, four input IA channels are convolved with four W channels in every cycle. The systolic data flow necessitates proper synchronization of IA and Psums, as shown in Fig. 5(a). Each processing run

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

generates a tile of OA of size $(1 \times W_{out} \times 4)$, and it takes $(H_{out} \times 1 \times M/4)$ runs to obtain all outputs. The dataflow also allows strided convolution and zero-padding.

2) PW: The PW layer execution can be represented by 2D matrix multiplication of IA with dimension $(HW \times M)$ and weight W with dimension $(M \times N)$ to produce OA with dimension $(HW \times N)$ as shown in Fig. 5(b), where M and N are number of input and output channels respectively. The IA matrix is partitioned into HW tiles each sizing $(1 \times M)$, and the W matrix is partitioned into N/n tiles each sizing $(M \times n)$. Since the PE array comprises 3×4 PEs, three IA tiles and four W tiles are passed to the PE array for computation in a single processing run. An IA tile is broadcasted to four PEs in a row, and a W tile is broadcasted to three PEs in a column. This mapping allows spatial reuse of IA along a row and W along a column. The Psums are locally accumulated and stored in the RF of individual PEs, and the final Psum is transferred to the PPM through row routers of NoC, reducing output writeback traffic to GLB-MEM. The IA and W tiles are sent to the PE array in compressed form for computation. In every processing run, $(3 \times 4n)$ tile of OA is generated, and it takes $(HW/3) \times (N/4n)$ runs to construct the entire OA matrix. n depends on the RF depth and is set to 16 in our implementation. This dataflow ensures the maximum reuse of IA and W with low OA writeback bandwidth.

3) FC: The FC layer can be represented by vector-matrix multiplication, as shown in Fig. 5(c). The IA is denoted as a vector of size $(1 \times M)$, and W is denoted as a matrix of size $(M \times N)$. The IA vector is divided into four tiles of size $(1 \times M/4)$, and each tile is broadcasted to three PEs in a column. The W matrix is divided into $4 \times N/6$ tiles of size $(M/4 \times 6)$ and is multicasted to three PEs in a column. A single PE receives two weights and a single activation per clock cycle and performs two MAC operations. The Psum reduction is made in two levels: PE-level and core-level. At the PE level, $M/4$ activations and $M/4 \times 2$ weights are streamed into each PE, and the resulting Psums are locally accumulated inside PE for $M/4$ cycles. At the core-level, the Psums stored in the RF of each PE are spatially reduced along four columns of the PE array. This two-level Psum reduction reduces the writeback traffic to GLB-MEM. The last column output provides the final result, which is sent to the PPM. Six OAs are generated in every processing run, and $N/6$ runs are required to construct the entire output vector. Each PE only uses two of the four MAC units available due to bandwidth constraints and low-weight reuse in the FC layer.

4) CONV: The standard convolutions (CONV) can be implemented by employing both DW and PW dataflows depending on the input channels (M). Specifically, RAMAN adopts the DW dataflow when the number of input channels is equal to 1 $(M = 1)$, while it employs the PW dataflow for cases where the number of input channels exceeds 1 $(M > 1)$. A detailed mapping of the CONV layer onto DW and PW dataflows depending on the input channel size is provided in Section VIII of the supplementary document.

III. RAMAN MEMORY OPTIMIZATIONS TO SUPPORT DEPLOYMENT AT THE EDGE.

The RAMAN accelerator was developed targeting tinyML edge computing applications, and the following are the features that enable RAMAN to be deployable on edge:

A. Peak activation memory reduction:

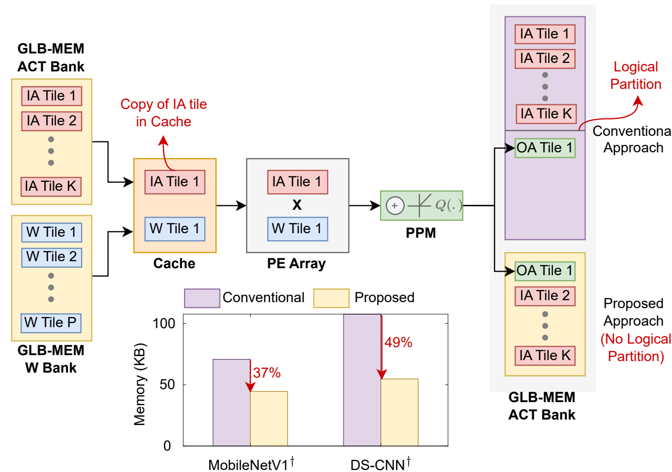


Fig. 6: Illustration of the IA and OA memory space overlapping to reduce peak activation memory.

The state-of-the-art (SOA) accelerators use two different memory spaces for storing IAs and OAs in a single memory. The IA and OA memory spaces are logically partitioned to prevent memory collision issue. This conventional approach requires a peak memory of:

$$MEM_size_{(SOA)} = max(\{sum(IA^l, OA^l)\}_{l=1}^{L}) \quad (1)$$

Where $MEM_size_{(SOA)}$ denotes the peak memory required by the state-of-the-art accelerators, L represents the total number of layers in the network, IA^l, OA^l represents the IA and OA memory sizes of a particular layer $'l'$. Effectively in the conventional approach, the activation memory size is governed by the layer with the maximum sum of IA and OA memory sizes.

In our approach, as illustrated in Fig. 6, the logical partition between IA and OA memory spaces is eliminated, meaning that the OAs are overwritten in the same IA memory space. The memory size required by the proposed approach is given by:

$$MEM_size_{(RAMAN)} = max(\{max(IA^l, OA^l)\}_{l=1}^{L}) \quad (2)$$

Where $MEM_size_{(RAMAN)}$ denotes the peak memory required by RAMAN. Compared with the conventional approach, the proposed memory reduction scheme reduces the peak activation memory of the MobileNetV1† model trained for visual wake word (VWW) task by 37% and the DS-CNN† model trained for keyword spotting (KWS) application by 49% as shown in Fig. 6. We have made a couple of modifications to the original MobileNetV1 [5] and DS-CNN models [55] as

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

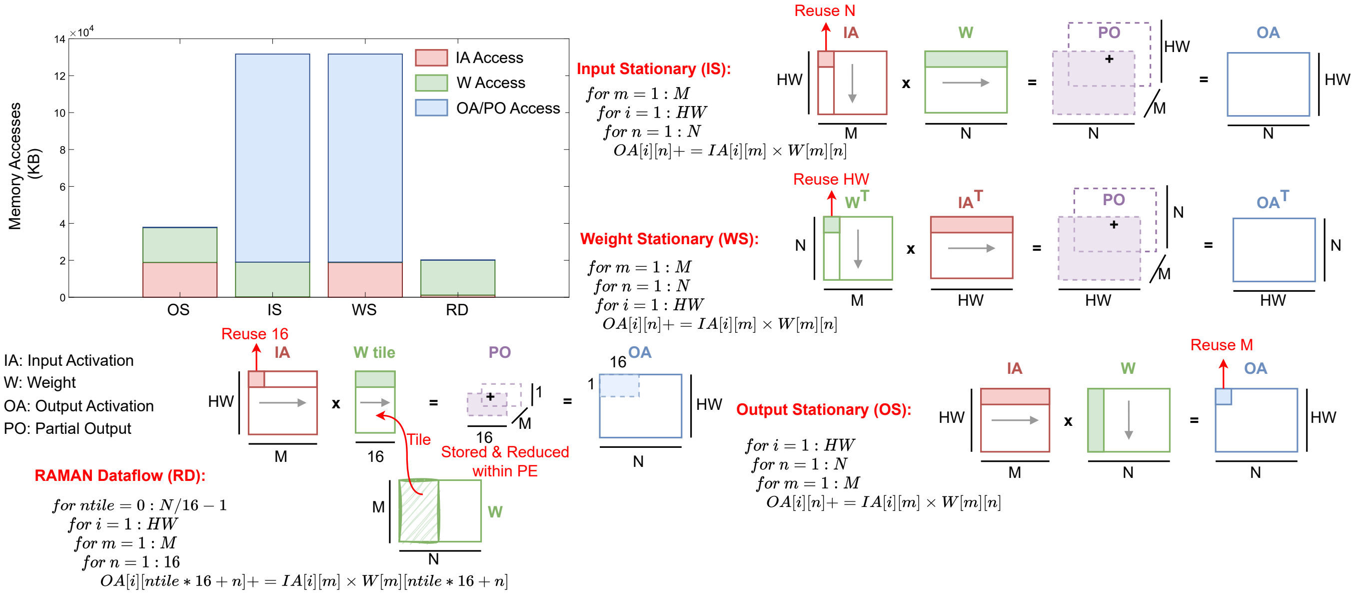


Fig. 7: Comparison of dataflows in terms of memory accesses.

per our requirements, and the modified models are denoted as MobileNetV1† and DS-CNN† models from here on.

However, the memory collision issue arises while overwriting the OA into the IA memory space if the IA is not consumed before the OA writeback. This problem can be solved by storing a copy of the IA tile in the cache and then proceeding with the computation, as shown in Fig. 6. This makes the original copy of the IA tile (e.g., IA tile-1 in Fig. 6) in the GLB-MEM activation bank redundant, allowing the OA tile to occupy that space. The disparity in tile sizes between IA and OA is another challenge with the proposed approach. If the OA tile is larger than the IA tile, it might corrupt the contents of IA tile-2 by overflowing the memory area of IA tile-1 (cf. Fig. 6) and spilling over the subsequent tile (e.g., IA tile-2 in Fig. 6). This is particularly true in the PW layer when N>M, but it's not a concern in the DW layer because OA is always less than (if stride>1) or equal (if stride=1) to IA. Intelligent data organization inside the memory and pre-fetching the IA tile-2 to the cache prior to OA tile-1 writeback aids in resolving this issue.

B. Dataflow to reduce memory accesses:

Fig. 7 shows the GLB-MEM memory accesses evaluated for a single PE for different dataflows and their corresponding abstract loop nests. The analysis was carried out on the PW layers of the MobileNetV1† model [5], whose operation can be described as a matrix-multiplication of $IA_{(HW\times M)} \times W_{(M\times N)}$, where input channel M is a shared dimension of multiplication. The output stationary (OS) dataflow, also termed inner product dataflow has the shared dimension in the innermost loop and achieves good output reuse (M times) but has poor input reuse. It computes an OA element one at a time by traversing a row of IA and a column of W. On the other hand, the Input stationary (IS) and the Weight stationary (WS) dataflows achieve good input reuse (N times) and weight reuse (HW times), respectively, but poor output reuse. It computes one partial output matrix (PO) of size ($HW \times N$) at a time by traversing a row of W and a column of IA in IS (or a column of W^T and a row of IA^T in WS) and M such matrices are generated before the final reduction. The size of the partial output matrix is massive to be stored locally inside the PE and thus has to be saved in the GLB-MEM, creating significant output data traffic evident from Fig. 7. Additionally, the bandwidth required by the partial output matrix (24b value) is much higher than the final output (8b value). Thus, moving the partial output matrix from PE to GLB-MEM is costly.

In this work, we employ a RAMAN dataflow (RD) inspired by Gustavson's algorithm to reduce the overall data-movement cost of the PW layer. It computes a row of 16 OAs at a time by traversing a row of IA, and a row of 16 elements from W. Since just 16 partial outputs are generated at a time, it is locally stored and reduced inside the PE. The dataflow is the most efficient as it avoids the two extremes of OS (by reusing IA by a factor of 16) and IS/WS (by eliminating the partial output traffic) dataflows. Only the quantized 8b accumulation result is sent to the GLB-MEM, drastically decreasing the output traffic. The W accesses from the GLB-MEM are reduced by storing a W tile in the cache and re-using them for HW times in the PW layer. Compared to the OS and IS/WS dataflows, the RAMAN dataflow significantly reduces the PW layer memory access by 1.9x and 6.5x, respectively.

C. Leveraging weight sparsity employing hardware-aware balanced pruning:

We propose a hardware-aware balanced weight pruning technique to reduce memory storage and access and improve energy efficiency and throughput for the PW layer. This strategy exemplifies the synergy between software-hardware co-optimization, where a deep neural network undergoes pruning

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

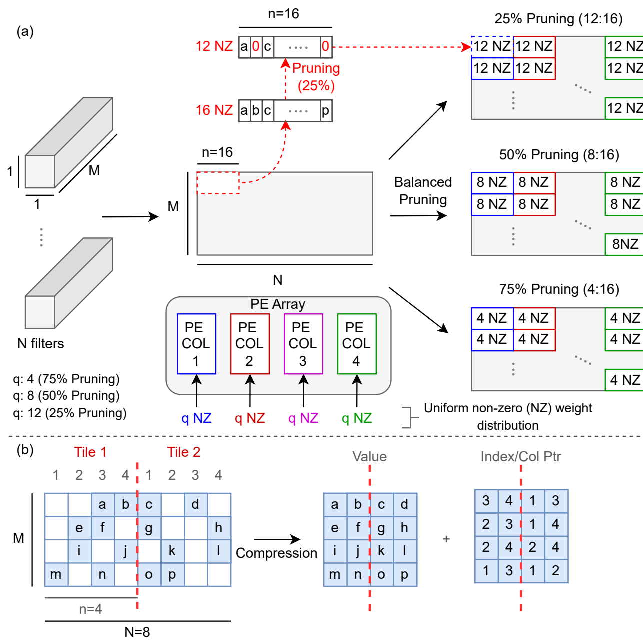


Fig. 8: Hardware-aware balanced weight pruning illustration. (a) Balanced pruning strategy for different pruning ratios. Depending on the pruning ratio, each PE receives an equal number of non-zero weight elements, leading to a uniform workload across PEs. (b) Decomposing a sparse weight matrix into a compressed, dense matrix containing only non-zero elements and an index matrix.

during training, considering the underlying hardware architecture to have minimum accuracy degradation [56]. Initially, the weights are divided into tiles of size $M \times n$ as shown in Fig. 8(b), and for ease of explanation, we have considered $n = 4$. A fixed number of weights are pruned in each tile row based on the magnitude, leading to structured sparsity, which can be efficiently exploited in our hardware. We employ the CSR scheme with modifications to store W in compressed form. The CSR scheme uses a set of non-zero values, bounds/row pointers, and index/column pointers to represent compressed information. The bounds/row pointer determines the number of non-zero elements present in a row of the sparse matrix, and the index provides the column index of the non-zero value. In our pruning scheme, since all the rows in a sparse matrix tile have the same number of non-zero elements, we don't have to store bounds explicitly and the start location of a particular tile in memory can be easily identified. The index in our implementation provides a non-zero column index for a specific tile and not the entire sparse matrix requiring $log_2(n)$ bits instead of $log_2(N)$ in the conventional CSR approach. In our implementation, n is 16, requiring a 4b index, and the values of the weights are quantized to 8b. Thus, each non-zero weight element is represented by a 12b value-index pair. Furthermore, the balanced pruning methodology shown in Fig. 8(a) ensures uniform zero/non-zero weight distribution across the weight tiles, thereby eliminating the workload imbalance problem. Without the balanced pruning strategy, the non-zero weights would be non-uniformly distributed across different weight tiles processed by different PEs resulting in workload imbalance, and the overall performance is limited by the PE with the heaviest workload. The PEs with low non-zero weight tile distribution complete their execution faster and have to be stalled for the slowest one (the PE with high non-zero weight tile distribution).

IV. Implementation Results

This section assesses the RAMAN's performance on Efinix FPGA [57].

A. Efinix FPGA Implementation Results

We evaluate RAMAN's performance on popular networks aimed at tinyML edge computing applications: MobileNetV1 and DS-CNN. The input dimensions are resized as per the requirement: 96×96 for the MobileNetV1† and 30×32 for the DS-CNN† model. The DS-CNN† model was trained on the google speech command dataset for the keyword spotting (KWS) application [58]. The input audio with each 1s duration was sampled at 16KHz and fed to the cascade of asymmetric resonators [59] to generate a cochleagram. RAMAN used the output cochleagram of size 30×32 as an input to infer the keyword. The MobileNetV1† model was trained on the Visual Wake Words (VWW) dataset [60] with input image converted to gray-scale. The specific MobileNetV1† and DS-CNN† network architectures deployed on RAMAN are provided in Section VII of the supplementary document.

RAMAN was optimized and implemented on an Efinix Ti60 FPGA, with parameters, instructions, and pre-processed inputs written into corresponding memories. The specifications of the RAMAN architecture are shown in Table II. Fig. 9(a) shows the register breakdown of the RAMAN architecture. The PE array utilizes 52% of the registers since each PE comprises 16×24b RF to store Psums. Furthermore, RAMAN provides the flexibility to downsize the PE RF width to 20b (or lower) depending on the application to reduce register utilization. The PPM stores post-processing parameters in registers leading to 32% register utilization. The LUT breakdown of the RAMAN architecture is shown in Fig. 9(b). It is evident that the controller and the PE array consume most of the LUTs in FPGA fabric, accounting for 86% of LUTs, and the ASE utilization is insignificant. The LUTs and registers are inferred as the eXchangeable Logic and Routing (XLR) cells in Efinix FPGA. The power distribution of the RAMAN architecture is presented in Fig. 9(c-d), where logic and clock account for 80%, while memory and DSP constitute the remaining 20%.

The RAMAN architecture comprises 48 MAC units operating at 75 MHz, which theoretically translates to a throughput of 7.2 GOp/s. However, since operations involving zero are skipped in the PW layer, we achieve an effective throughput of 13.5 GOp/s and 10.5 GOp/s for the MobileNetV1† and DS-CNN† models, respectively by exploiting both activation and weight sparsity. The power consumption of the RAMAN architecture on Efinix Ti60 FPGA is estimated to be 136.96 mW (89.37 mW dynamic power + 47.6 mW static power) and 131.77 mW (84.39 mW dynamic power + 47.38 mW static

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

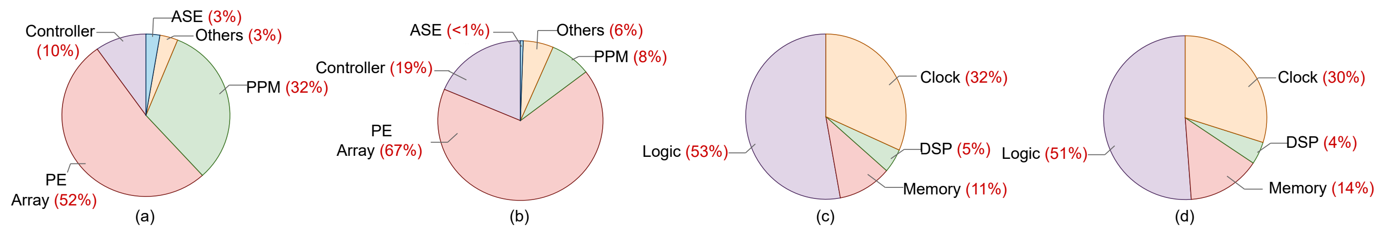


Fig. 9: Resource utilization breakdown of (a) Registers, (b) LUTs and power breakdown of RAMAN for (c) MobileNetV1† model, (d) DS-CNN† model for 75% weight pruning in the PW layers.

TABLE II: RAMAN Specifications.

Platform	Efinix Ti60
Layers Supported	CONV, DW, PW, FC and Max/Average pooling.
Number of PEs	12 (4 MACs/PE)
Reg-file Memory	PE Array: 0.576 KB PPM: 0.32 KB
Clock Rate	75 MHz
Arithmetic Precision	W & IAs: 2b, 4b or 8b fixed point, Psums: 24b fixed point.
Power	137 mW for MobileNetV1† 132 mW for DS-CNN†
XLR cells	52261 (85.96% util.)
DSPs	61 (38.12% util.)
Memory Blocks	168 (65.62% util.) for MobileNetV1† 118 (46.09% util.) for DS-CNN†
Theoretical Throughput	7.2 GOp/s (3.6 GMACS)
Effective Throughput	13.5 GOp/s for MobileNetV1† 10.5 GOp/s for DS-CNN†
Energy Efficiency	2355 Inferences/J for MobileNetV1† 6609 Inferences/J for DS-CNN†

power) for the MobileNetV1† and DS-CNN† models, respectively. Therefore, the effective power efficiency of RAMAN at 75 MHz and 75% PW weight sparsity is 98.47 GOp/s/W (or equivalently 2355 Inferences/J) for the MobileNetV1† and 79.68 GOp/s/W (or equivalently 6609 Inferences/J) for the DS-CNN† model. A detailed power and memory breakdown of the RAMAN architecture is provided in Section IX of the supplementary document.

The average MAC utilization of the DW and PW layers is around 59% and 86%, respectively, with an overall utilization of 78%. DW layers have a lower MAC utilization due to limited IA re-use and the time spent to fetch the IAs and Ws from the GLB-MEM memory. On the other hand, memory access latency of the PW layers in RAMAN is completely hidden with MAC operations; however, the latency introduced due to data fetch from the RF of PEs cannot be completely hidden as the next tile's MAC operation can be started only after completely evicting the contents of the RF of the current tile. The MAC units remain idle while fetching the contents of RF and transferring them to the post-processing module. This problem can be solved by double buffering the reg-file in the PEs, increasing the PW layer utilization to 97%.

B. Sparsity Results

1) Leveraging sparsity in latency reduction: Fig. 10 shows the accuracy vs. latency trade-off at compile-time and run-time. Figs. 10(a) and 10(b) demonstrate the RAMAN processing latency and model accuracy as a function of weight sparsity for the MobileNetV1† and DS-CNN† models, respectively. The required degree of weight sparsity is pre-set at compile time using the hardware-aware balanced weight pruning technique described in Section III-C. The performance is assessed for four weight sparsity levels. It is evident that the latency reduction is almost linear with the degree of weight sparsity, and the accuracy degradation is minimal. Additionally, the latency distribution for the PW layers of the MobileNetV1† and DS-CNN† models are shown in Figs.10(c) and 10(d). There is a significant reduction in latency by leveraging IA sparsity. The latency gains are substantial in the final layers of the MobileNetV1† model due to an increase in IA sparsity and number of operations. In contrast, the DS-CNN† model is computationally intensive in the initial layers. Furthermore, we compare the latency after run-time activation pruning for different thresholds. Thresholds are set based on the input data distribution. For the MobileNetV1† model, RAP with pruning threshold 40 reduces latency by an additional 16% compared to no pruning case, with accuracy degradation of ≈2%. For the DS-CNN† model, RAP reduces total latency by 12% with the pruning threshold set at 20. However, the possibility of minimizing latency with RAP is only confined to the initial layers since the final layers of DS-CNN† have low computational intensity.

2) Leveraging sparsity in memory access reduction: Table III shows the activation cache access breakdown for different layer types of the MobileNetV1† and DS-CNN† models. It is evident from the table that the majority of the cache accesses happen in the DW-PW layers, and the activation cache reads are more than the cache writes, leading to effective cache reuse. Additionally, leveraging IA sparsity reduces the cache accesses by $40-45\%$. Finally, Table IV presents the weight cache access breakdown for different sparsity or pruning ratios. Again, a similar trend is observed where reads dominate writes, indicating effective cache reuse and the cache accesses reduce with increased sparsity ratio. In addition, it is observed that the weight cache reads are further reduced by 30% with IA sparsity since when the IA value is zero, the corresponding weight is not read from memory. However, the weight cache writes remain the same since all weight values are loaded to the cache initially, irrespective of IA sparsity. In addition, run-time activation pruning reduces IA cache access by $8-10\%$, and parameter cache reads by $13-21\%$.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

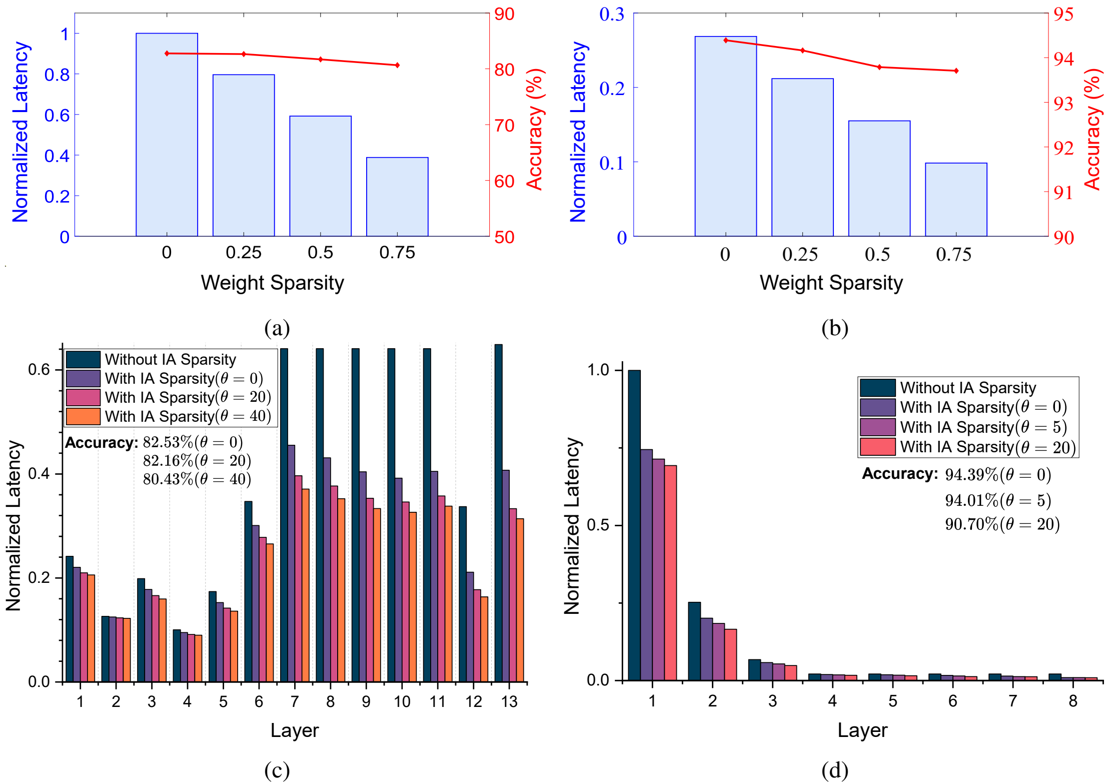


Fig. 10: Leveraging sparsity in latency reduction: Compile-time and run-time latency vs. accuracy trade-off for MobileNetV1† model (left-panel) and DS-CNN† model (right-panel). Top-panel: Latency of RAMAN for different weight sparsity (pruning) ratios set at compile time. Bottom-panel: Latency distribution of PW layers in RAMAN with run-time activation pruning at different activation pruning thresholds (θ) with accuracy highlighted. The accuracy and latency estimates are for two tasks: keyword spotting using the DS-CNN† model trained on the Google speech command dataset and image classification using the MobileNetV1† model trained on the VWW dataset.

TABLE III: Activation cache access breakdown for (a) DS-CNN† and (b) MobileNetV1† model.

Layer	With IA sparsity (in KB)				Without IA sparsity (in KB)			
	Read		Write		Read		Write	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
CONV	0.5	9.8	0.2	6.7	2.7	13.5	0.96	9.2
DW	171	356	77	144	281	630	129	246
PW	47	173	47	173	76	335	76	335
Pool	0	0	0	0	0	0	0	0
FC	0.13	0.26	0.06	0.26	0.13	0.26	0.06	0.26
Total	**219**	**539**	**124**	**323**	**360**	**979**	**206**	**590**

TABLE IV: Weight cache access breakdown for (a) DS-CNN† and (b) MobileNetV1† model.

Weight Sparsity	Read (in KB)				Write (in KB)	
	With IA sparsity		Without IA sparsity		-	
	(a)	(b)	(a)	(b)	(a)	(b)
0%	1727	6344	2451	9425	49	295
25%	1295	4908	1839	7250	37	222
50%	863	3472	1226	5075	25	149
75%	432	2036	613	2900	12	76

3) Leveraging sparsity in storage reduction: The global memory requirements of the design are tabulated in Table V. The peak activation memory needed is obtained by overwriting OAs in the same IA memory space. The parameter memory needed is the sum of memory needed to store weights and post-processing parameters of all the layers. Taking pruning into account, the parameter memory reduces with an increase in pruning percentage, as shown in Table V.

TABLE V: Global memory requirements.

Memory (in KB)	Activation Memory	Parameter Memory with different pruning ratios			
		0	0.25	0.5	0.75
DS-CNN†	54.72	61.968	49.656	37.392	25.104
Mobile -NetV1†	44.56	324.288	251.328	178.368	105.384

C. Comparison with prior works

We compare RAMAN with prior works quantitatively in Table VI. [20]–[26], [31] leverage sparsity in either activation or weights or both. NullHop [20] introduces an architecture that exploits activation sparsity to accelerate computation through zero skipping and reduces storage demands. However, it's worth noting that this architecture does not exploit weight sparsity or incorporate activation pruning at run-time to increase the sparsity of activations. [22]–[25] exploit sparsity exclusively in weights and not activations. [21], [26] exploit weight sparsity by zero skipping and activation sparsity by clock gating. In contrast, RAMAN optimizes both power and

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

TABLE VI: Quantitative comparison of RAMAN with prior implementations.

	NullHop [20]	McDanel et al. [21]	SpWA [22]	Lu et al. [23]	Yin et al. [24]	Xie et al. [25]	Zhu et al. [26]	Lu et al. [31]	Wu et al. [36]	FitNN et al. [33]	Hao et al. [34]	Synetgy [35]	RAMAN (Ours)
Platform	Xilinx Zynq-7100	Xilinx VC707	Xilinx ZC706	Xilinx ZCU102	Xilinx ZCU102	Intel Arria10	Xilinx ZCU102	Xilinx ZC706	Xilinx ZU3EG	Xilinx Zynq-7020	Xilinx Pynq-Z1	Xilinx ZU3EG	Efinix Ti60
Model	VGG16	Custom	VGG16	VGG16	MobileNetV2	MobileNetV2	ResNet-50	1-D CNN	ResNet-50	iSmart2	Custom	DiracDeltaNet	MobileNetV1†, DS-CNN†
Precision	16b	N/A	16b	16b	16b	8b	16b	16b	8b	N/A	N/A	W:1b Act.:4b	2, 4 or 8b
LUTs	229k	239k	155.2k	132.34k	194.6k	102.6k	390k	3.238k	40.78k	39.19k	43.9k	24.13k	37.2k◇
Registers	107k	201k	153.02k	68.8k	95.68k	N/A	278k	N/A	45.25k	49.5k	40k	29.9k	8.6k
DSPs	128	112	768	364	884	512	1352	48	257	220	202	37	61
Freq. (MHz)	60	170	166	200	190	170	200	200	150	150	100	250	75
Pwr. Efficiency (GOp/s/W)	27.4	N/A	N/A	12.33	N/A	18.69	N/A	45.05	44.95	N/A	N/A	8.56	98.47[a]⋆ 79.68[b]⋆
Power (W)	1.1	2.2	N/A	23.6	13.32	4.6	15.4	0.506	1.4	1.97	2.2	5.5	137mW

◇4-input LUTs. ⋆Estimated for 8b precision. [a]VWW trained on MobileNetV1†. [b]KWS trained on DS-CNN†.

TABLE VII: Quantitative comparison of RAMAN with prior works for VWW and KWS tasks.

Platform	Task	Acc. (%)	Perf. (inf./s)	Power (mW)	Energy (uJ/inf.)
EK-RA6M4 [61]	VWW	85.4[a]	6.24	74	11.85k
	KWS	90.1[b]	19.69	75	3.79k
RX65N-Cloud-Kit [61]	VWW	85.4[a]	4.07	54.24	13.32k
	KWS	90.1[b]	12.37	54.7	4.42k
Nucleo-L4R5ZI [61]	VWW	85.4[a]	1.65	40.31	24.31k
	KWS	90.1[b]	5.54	40.83	7.37k
xG24-DK2601B [61]	VWW	84.7[a]	8.97	10.22	1.14k
	KWS	90.3[b]	27.45	11.03	401.86
XC7K410T [62]	KWS	88.8[c]	N/A	68	N/A
XC7A200T [63]	KWS	90.1[c]	680	470	700
Efinix TI60 RAMAN	**VWW**	**80.7[d]**	**322.53**	**136.96**	**424.63**
	KWS	**93.7[e]**	**869.65**	**131.77**	**151.31**

Model employed: [a]MobileNetV1(0.25x), [b]DS-CNN, [c]Custom model, [d]MobileNetV1†, [e]DS-CNN†.

latency by gating and skipping processing cycles for both activations and weights.

[31]–[35] propose hardware architecture for the DNNs targeted for edge computing. Nevertheless, among these works, only Lu et al. [31] incorporate sparsity optimizations. RAMAN stands out with its exceptional power efficiency and efficient resource utilization, including LUTs, registers, and DSPs, when compared to previous edge implementations. It's worth noting that RAMAN utilizes 4-input LUTs on Efinix FPGA, in contrast to other Xilinx-based implementations which typically utilize 6-input LUTs. A comparison of recent FPGA-based CNN accelerators is provided in Section X of the supplementary document.

Additionally, we have conducted a comparative analysis with other implementations listed in the MLPerf [61] benchmark, specifically within the Inference-tiny category, as well as prior FPGA-based implementations [62], [63] for VWW and KWS tasks. The results, as presented in Table. VII, highlight RAMAN's notable superiority in terms of energy efficiency and latency, surpassing the majority of previous efforts. This underscores RAMAN's exceptional performance in the realm of tinyML applications. An extended ablation study showcasing our approach across various sparsity levels for both Visual Wake Word (VWW) and Keyword Spotting (KWS) tasks is presented in Fig. 11. Additionally, the depiction includes comparisons with prior implementations listed in the MLPerf inference-tiny category. The MLPerf benchmark establishes quality targets, specifying a minimum accuracy requirement of 80% for VWW and 90% for KWS tasks. Notably, our models consistently meet these quality standards even after undergoing aggressive pruning. Furthermore, owing to weight pruning and compressed storage of model parameters, we see a significant reduction of storage requirements in RAMAN compared to the base model.

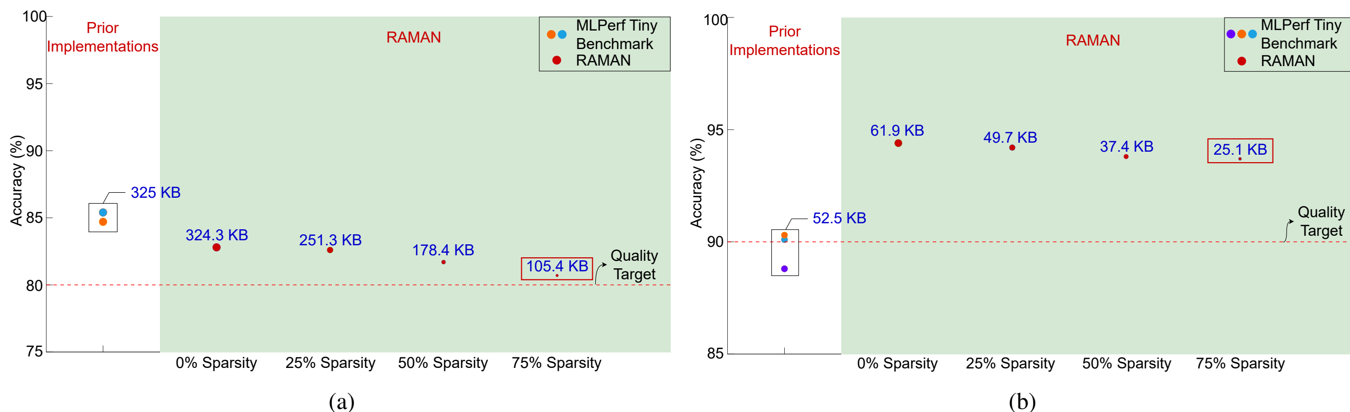


Fig. 11: Ablation study of our approach for different sparsity levels compared with prior implementations listed in MLPerf Benchmark within the Inference-tiny category for (a) Visual Wake Word task and (b) Keyword spotting task. The size of the dots is proportional to the model size. The model used for comparison in Table VII is highlighted with a red bounding box.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

V. Conclusions

Deep neural networks (DNNs) introduce weight and activation sparsity, enabling deep learning applications to operate more efficiently on hardware platforms with constrained resources and energy. However, these sparse models need specialized hardware architectures to fully benefit from the sparsity for storage, latency, and energy gains. In this work, a reconfigurable and sparse neural network accelerator exploiting both weight and activation sparsity is proposed for tinyML applications. RAMAN uses an activation sparsity engine to leverage unstructured activation sparsity and a hardware-aware balanced pruning to exploit structured weight sparsity. We propose a novel dataflow inspired by Gustavson's algorithm that enables the Psum reduction with the PE array and significantly reduces the writeback traffic. The dataflow reduces the PW layer memory accesses by 1.9x compared to output stationary dataflow and 6.5x compared to input/weight stationary dataflow. Furthermore, we propose a technique to lower peak memory activation by overlaying IA and OA on the same memory space, which can reduce storage requirements by up to 50%. These memory optimizations in terms of memory accesses and memory storage enable RAMAN to be deployable on edge with a small form factor.

RAMAN supports a wide range of DNN topologies from standard CNN layers to modern DS-CNNs and can be configured to support accuracy vs. power/latency tradeoffs using techniques deployed at compile time and run time. RAMAN architecture was implemented on Efinix FPGA with 37.2K LUTs using 48 MAC units distributed across 3×4 PEs. The design achieves an overall energy efficiency of 2355 and 6609 Inference/J for MobileNetV1† and DS-CNN† models at 75 MHz on Efinix FPGA. The effective power efficiency of the system is estimated to be 98.4 and 79.68 GOp/s/W for MobileNetV1† and DS-CNN† models, respectively. A demonstration video of the proposed RAMAN accelerator on the Efinix Ti60 FPGA board for the keyword spotting task, where we control the maze game using the keywords uttered by the user, can be found here https://youtu.be/sCksj7nlBY8.

VI. Acknowledgements

The authors would like to express their sincere gratitude and appreciation to their colleagues, Madhuvanthi Srivatsav, Hitesh Pavan, Anand Chauhan, and Shankaranarayanan, for their invaluable help throughout this work.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[3] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[5] 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*, 2017.

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv*, 2019.

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

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

[9] S. Han, H. Mao, and W. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[10] X. Ding, G. Ding, J. Han, and S. Tang, "Auto-Balanced Filter Pruning for Efficient Convolutional Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12262

[11] H. Tanaka, D. Kunin, Y. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *Advances in neural information processing systems*, vol. 33, 2020. [Online]. Available: https://par.nsf.gov/biblio/10291300

[12] A. Yousefzadeh and M. Sifalakis, "Training for temporal sparsity in deep neural networks, application in video processing," *arXiv*, 2021.

[13] 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, Jan 2017.

[14] 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)*, Oct 2016, pp. 1–12.

[15] 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," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 27–40.

[16] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1–13.

[17] 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, Feb 2020.

[18] 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," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 243–254, jun 2016. [Online]. Available: https://doi.org/10.1145/3007787.3001163

[19] 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, Feb 2021.

[20] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019.

[21] B. McDanel, S. Q. Zhang, H. T. Kung, and X. Dong, "Full-Stack Optimization for Accelerating CNNs Using Powers-of-Two Weights with FPGA Validation," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 449–460. [Online]. Available: https://doi.org/10.1145/3330345.3330385

[22] L. Lu and Y. Liang, "SpWA: An Efficient Sparse Winograd Convolutional Neural Networks Accelerator on FPGAs," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[23] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 17–25.

[24] X. Yin, Z. Wu, D. Li, C. Shen, and Y. Liu, "An Efficient Hardware Accelerator for Block Sparse Convolutional Neural Networks on FPGA," *IEEE Embedded Systems Letters*, pp. 1–1, 2023.

[25] X. Xie, J. Lin, Z. Wang, and J. Wei, "An Efficient and Flexible Accelerator Design for Sparse Convolutional Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 7, pp. 2936–2949, 2021.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2024.3386832

[26] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An Efficient Hardware Accelerator for Structured Sparse Convolutional Neural Networks on FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 1953–1965, 2020.

[27] W. Sun, D. Liu, Z. Zou, W. Sun, S. Chen, and Y. Kang, "Sense: Model-Hardware Codesign for Accelerating Sparse CNNs on Systolic Arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 470–483, 2023.

[28] Y. Meng, C. Yang, S. Xiang, J. Wang, K. Mei, and L. Geng, "An Efficient CNN Accelerator Achieving High PE Utilization Using a Dense-/Sparse-Aware Redundancy Reduction Method and Data–Index Decoupling Workflow," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[29] G. Zhang, R. Zhang, R. Wang, and S. Zhu, "A Systolic Array-Based Scheduling Strategy for Sparse CNN Accelerators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[30] X. Yin, Z. Wu, D. Li, C. Shen, and Y. Liu, "An efficient hardware accelerator for block sparse convolutional neural networks on FPGA," *IEEE Embedded Systems Letters*, 2023.

[31] J. Lu, D. Liu, X. Cheng, L. Wei, A. Hu, and X. Zou, "An Efficient Unstructured Sparse Convolutional Neural Network Accelerator for Wearable ECG Classification Device," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 11, pp. 4572–4582, 2022.

[32] K. Choi and G. E. Sobelman, "An Efficient CNN Accelerator for Low-Cost Edge Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 4, aug 2022. [Online]. Available: https://doi.org/10.1145/3539224

[33] Z. Zhang, M. A. P. Mahmud, and A. Z. Kouzani, "FitNN: A Low-Resource FPGA-Based CNN Accelerator for Drones," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21 357–21 369, 2022.

[34] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317829

[35] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek, and K. Keutzer, "Synetgy: Algorithm-Hardware Co-Design for ConvNet Accelerators on Embedded FPGAs," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 23–32. [Online]. Available: https://doi.org/10.1145/3289602.3293902

[36] B. Wu, T. Yu, K. Chen, and W. Liu, "Edge-Side Fine-Grained Sparse CNN Accelerator With Efficient Dynamic Pruning Scheme," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2024.

[37] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, June 2019.

[38] N. Shah, P. Chaudhari, and K. Varghese, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Co-processor for Deep Convolutional Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5922–5934, 2018.

[39] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 16–25. [Online]. Available: https://doi.org/10.1145/2847263.2847276

[40] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 152–159.

[41] C. Zhang and V. Prasanna, "Frequency Domain Acceleration of Convolutional Neural Networks on CPU-FPGA Shared Memory System," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 35–44. [Online]. Available: https://doi.org/10.1145/3020078.3021727

[42] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection," *Electronics*, vol. 10, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/282

[43] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[44] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[45] L. Bai, Y. Zhao, and X. Huang, "A CNN Accelerator on FPGA Using Depthwise Separable Convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[46] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A High-Performance CNN Processor Based on FPGA for MobileNets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 136–143.

[47] S. Yan, Z. Liu, Y. Wang, C. Zeng, Q. Liu, B. Cheng, and R. C. Cheung, "An FPGA-based MobileNet Accelerator Considering Network Structure Characteristics," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 17–23.

[48] Y. Yu, C. Wu, T. Zhao, K. Wang, and L. He, "OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 35–47, 2020.

[49] X. Xie, M. Zhu, S. Lu, and Z. Wang, "Efficient Layer-Wise N: M Sparse CNN Accelerator with Flexible SPEC: Sparse Processing Element Clusters," *Micromachines*, vol. 14, no. 3, p. 528, 2023.

[50] H.-A. Rashid, U. Kallakuri, and T. Mohsenin, "TinyM2Net-V2: A Compact Low Power Software Hardware Architecture for M ulti m odal Deep Neural Networks," *ACM Transactions on Embedded Computing Systems*, 2023.

[51] G. Zhang, N. Attaluri, J. S. Emer, and D. Sanchez, "Gamma: Leveraging Gustavson's Algorithm to Accelerate Sparse Matrix Multiplication," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 687–701. [Online]. Available: https://doi.org/10.1145/3445814.3446702

[52] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney, and K. Keutzer, "HAWQV3: Dyadic Neural Network Quantization," in *ICML*, 2021.

[53] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[54] M. M. Strout, L. Carter, J. Ferrante, and B. Kreaseck, "Sparse Tiling for Stationary Iterative Methods," *The International Journal of High-Performance Computing Applications*, vol. 18, no. 1, pp. 95–113, 2004. [Online]. Available: https://doi.org/10.1177/1094342004041294

[55] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers," *arXiv*, 2017.

[56] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 1135–1143.

[57] Efinix, "TI60 Data Sheet," Online, 2023. [Online]. Available: https://www.efinixinc.com/docs/titanium60-ds-v2.7.pdf

[58] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv*, 2018.

[59] Y. Xu, C. S. Thakur, R. K. Singh, T. J. Hamilton, R. M. Wang, and A. van Schaik, "A FPGA Implementation of the CAR-FAC Cochlear Model," *Frontiers in Neuroscience*, vol. 12, p. 198, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00198

[60] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual Wake Words Dataset," *arXiv*, 2019.

[61] M. Perf., "ML Commons," https://mlcommons.org/en/, 2023, [Online]. Available: https://mlcommons.org/en/.

[62] M. Wang and A. P. Chandrakasan, "Flexible Low Power CNN Accelerator for Edge Computing with Weight Tuning," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2019, pp. 209–212.

[63] A. N. Mazumder and T. Mohsenin, "A Fast Network Exploration Strategy to Profile Low Energy Consumption for Keyword Spotting," *arXiv*, 2022.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on April 24,2024 at 04:57:13 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.