Efficient Sparse Matrix Acceleration for Deep Learning *via* Two-Step Bitmap Tensor Architecture

Jia-Hong Weng [1], Yuan-Jin Lin [2], Wan-Hsun Tsai [1], Yu-Jie Yang [2] and Wei-Chen Tu [1, 2, 3*]
[1] Institutes of Microelectronics, National Cheng Kung University, Tainan City 701401, Taiwan
E-mail: q16121088@gs.ncku.edu.tw, q16134196@gs.ncku.edu.tw, wctu@gs.ncku.edu.tw
[2] Program on Semiconductor Manufacturing Technology, Academy of Innovative Semiconductor and Sustainable Manufacturing, National Cheng Kung University, Tainan City 701401, Taiwan
E-mail: m28121562@gs.ncku.edu.tw, m26131066@gs.ncku.edu.tw, wctu@gs.ncku.edu.tw
[3] Department of Electrical Engineering, National Cheng Kung University, Tainan City 701401, Taiwan
Correspond author: Wei-Chen Tu, email:wctu@gs.ncku.edu.tw

***Abstract*—As deep learning models continue to scale in complexity, their computational, memory, and energy demands become increasingly burdensome, especially in applications such as computer vision and natural language processing. To address these challenges, this paper proposes a novel two-step bitmap compression method tailored for column-balanced, block-wise pruned convolutional neural networks. The proposed scheme effectively encodes both structured sparsity introduced by pruning and irregular sparsity induced by activation functions, thereby reducing memory overhead and data movement. To support this format, we design a dedicated hardware accelerator for bidirectional sparse matrix multiplication, leveraging the Gustavson dataflow and preprocessing modules to maximize throughput and efficiency. The system demonstrates significant improvements: at 50% sparsity, data transfer time is reduced by 50% and computation time by 80%; at 75% sparsity, data transfer time is reduced to 25% and computation time by 85%. These results validate the effectiveness and scalability of the proposed solution for accelerating sparse inference in deep learning applications.**

I. INTRODUCTION

Convolutional neural networks (CNNs) are widely used in fields such as image classification, object detection, and Natural Language Processing (NLP). However, larger deep learning models inherently require more computational power and memory resources, especially in convolution layer and fully connected layer due to the intensive nature of matrix operations[1]. Moreover, redundant computations are prevalent in deep learning models. Take VGG-16 as an example, VGG16 up to 90% of multiply-accumulate operations involve zeros due to pruning and ReLU activations[2]. Thus, eliminating such operations is critical to improving computational efficiency.

In recent works, matrix compression techniques have been widely adopted to reduce the overhead of matrix data movement and arithmetic operations, particularly in the context of deep learning accelerators. These techniques aim to exploit the inherent sparsity in neural network weights and activations to minimize memory bandwidth usage and improve computational efficiency. Among them, widely used sparse matrix formats such as coordinate format (COO) and compressed sparse row (CSR) are effective in reducing storage requirements by storing only nonzero elements along with their corresponding indices.

However, while COO and CSR formats offer notable storage savings, they introduce indexing overhead that requires additional memory lookups and control logic. This complexity poses challenges in hardware implementations, where irregular access patterns hinder parallelism and reduce overall performance. Therefore, there is a growing need for compression formats that preserve data regularity to enable hardware-friendly access and efficient computation. To tackle these limitations, we propose a novel sparse matrix compression method for accelerating sparse inference in deep learning applications. A block-pruned, two-level bitmap encoding scheme that reduces memory traffic and improves storage efficiency. A high-performance sparse matrix multiplication method optimized for FPGAs, achieving reducing latency. An HLS-based hardware implementation that efficiently maps the design to the target platform, balancing speed and accuracy.

II. RELATED WORK

During the CNNs training process, forward propagation is used for feature extraction through layers such as convolution, pooling, normalization, and ReLU. However, the presence of redundant weights and activations leads to excessive computation and memory overhead. Various pruning strategies have been developed to address this problem. For example, post-training pruning simplifies inference and accelerates processing [3]. In general, pruning methods can be categorized into two types. Fig. 1 (a) depicts an unstructured pruning architecture, where individual weights are removed,

979-8-3315-7206-8/25/$31.00 ©2025 IEEE

resulting in irregular sparsity patterns that are less favorable for hardware acceleration. In contrast, Fig .1 (b) and (c) show structured pruning architectures, they removes entire groups of weights for rows, columns or blocks, producing more regular sparsity patterns and suited for efficient hardware implementation, Commonly structure such as N:M pruning [4] and column-balanced block-wise pruning [5].

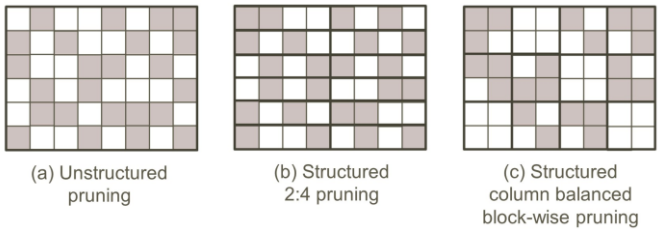

Fig. 1. Structured and Unstructured sparse matrix pruning method. (a)Unstructured pruning (b)Structured 2:4 pruning (c)Structured column balanced block-wise pruning

Moreover, a significant portion of weights and activations in CNNs are zero. To eliminate redundant computations and reduce memory overhead, recent studies have proposed various sparse matrix compression strategies. Common formats include COO, CSR, bitmap encoding, all of which have been employed to improve memory efficiency and computational performance. Beyond these standard techniques, more advanced schemes have been proposed to better adapt to specific sparsity patterns and further improve efficiency. Examples include compressed interleaved sparse row (CISR) [6], block compressed sparse row (BCSR) [7], and pointer-based bitmap encoding [8] . According to Li et al. [9], bitmap encoding generally offers better space efficiency than CSR, except when the sparsity level exceeds 88%.

Based on the above discussion, structured pruning is generally more hardware-friendly than unstructured methods. In this work, we adopt column-balanced block-wise pruning to evenly distribute non-zero elements across columns and partition weights into uniform blocks. Moreover, we employed bitmap encoding to enhance storage efficiency, which typically offers the smallest storage footprint, making it especially suitable for sparse matrices in computation workloads. Our method simplifies indexing, improves memory access patterns, and ensures efficient utilization of hardware resources by balancing sparsity with performance.

III. METHODOLOGY

A. *Two-step Bitmap Format*

We propose a novel two-step bitmap compression scheme specifically designed for weight matrices subjected to column-balanced block-wise pruning. This method captures both the structured sparsity introduced by pruning and the irregular sparsity resulting from activation functions. As illustrated in Fig. 2 , the compression process is divided into two steps. In the first step, a bitmap is used to record the locations of nonzero blocks, reflecting the regular sparsity pattern imposed by block pruning and enabling efficient hardware mapping. In the second step, an additional bitmap is applied to further compress the nonzero elements that remain after activations (such as ReLU) introduce additional zeros. This step effectively encodes the remaining irregular sparsity and improves data access efficiency.

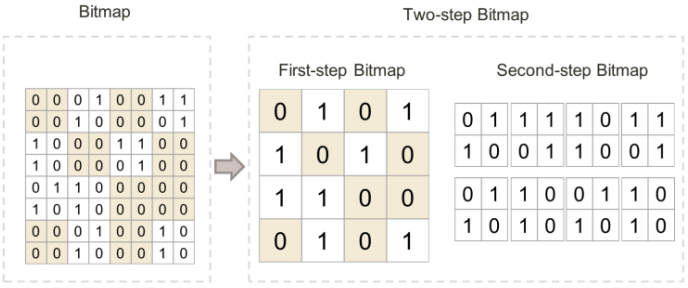

Fig. 2.Two-step bitmap compressed format.

A comparison of various compression schemes, including COO, CSR, and bitmap-based methods, is provided in Table 1. We assume 32 bits per nonzero value and 8 bits for each index or pointer entry, and define k as the block size, X as the number of rows in the matrix, Y as the number of columns, and S as the sparsity level. The analysis of storage requirements reveals that bitmap encoding typically offers superior compression relative to COO and CSR. Furthermore, the pointer-bitmap merges feature from both CSR and bitmap schemes delivers intermediate efficiency. Our two-step bitmap format developed in this study is guided by the condition $[(1/k)+(1-s)]>1$. This formula signals when our proposed scheme surpasses the traditional bitmap in compression ratio.

Table.1 Comparison of storage space across different compression formats.

Format	Storage overhead	
	Non-zero data	Index
COO	32XY(1-S)	$20 \cdot XY \cdot (1-S)$
CSR	32XY(1-S)	$10 \cdot XY \cdot (1-S) + 10 \cdot Y$
Bitmap	32XY(1-S)	XY
Pointer-bitmap[8]	32XY(1-S)	$XY + 10 \cdot Y$
Two-step bitmap	32XY(1-S)	$\frac{XY}{k} + XY \cdot (1-S) = \left[\frac{1}{k} + (1-S)\right] \cdot XY$

B. *Hardware Architecture*

We propose dedicated hardware architecture to support the proposed two-step bitmap compression format. The design offers high computational flexibility by dynamically detecting and bypassing multiplications involving zero elements, allowing higher sparsity to directly translate into improved computational efficiency. Fig. 3 illustrates the overall architecture. The blue blocks represent buffers that temporarily store data fetched from DRAM. Data is loaded into these buffers by the load module, and after computation, the results are retained until they are sequentially written out by the store module. The yellow blocks highlight the core processing unit responsible for performing sparse matrix

multiplications. In addition, two preprocessing modules are placed before the main computation to decode the two-step bitmap format. These modules ensure that the compressed data is efficiently interpreted, enabling streamlined and effective downstream processing.

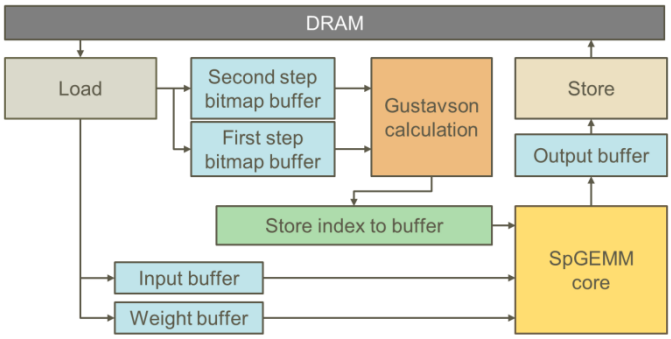

Fig. 3. Hardware architecture of two-step bitmap accelerator.

C. *Hardware Algorithm*

Our design adopts bidirectional sparse matrix multiplication based on the Gustavson dataflow [10], enabling efficient handling of sparse data through bitmap-guided filtering. First, bitmap entries of 0 and 1 are used to perform lightweight AND operations, allowing for the quick elimination of zero elements and reducing unnecessary computation. An output bitmap is generated during processing, and an index unit identifies positions that require multiplication. Nonzero values are then selectively fetched based on available processing elements (PEs) and computed, and the calculated results would be stored in the output buffer.

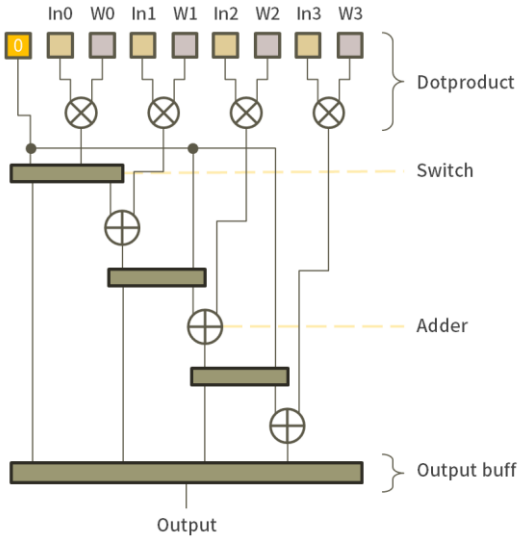

Fig. 4. Dot-product module architecture.

The computation proceeds as follows: The load module stores two-step bitmaps with corresponding nonzero activations and weights in the buffer. The first-step bitmap filters activations, which are then ANDed with the second-step bitmap to finalize valid positions. A sorting step shifts 1s downward, updates indices, and stores them in the buffer. According to available PEs, required indices and values are fetched, multiplied, and results written back sequentially. Fig. 4 shows the Dot-product module, which fetches data, performs MAC operations, and handles irregular positions. Switch and output buffer units ensure correct timing and utilization. In Fig. 4, In0 to In3 denote activations and W0 to W3 denote weights, selectively fetched and aligned with bitmap-guided indices.

D. *Experimental Flow*

We selected the Xilinx ZCU104 platform to leverage its programmability and flexibility for our hardware implementation. The experiment follows a hardware–software co-design methodology, where C-level simulation is first employed to predict inference behavior and accelerate development. As shown in Fig. 5, the complete experimental flow spans software-based training to FPGA-based inference. First, the CNN model is trained in software and converted into an HLS-compatible format using hls4ml. Next, Vitis HLS is used to translate the C code into RTL, generating CNN hardware IP cores. These cores are then integrated with the proposed sparse matrix multiplication engine and supporting modules such as AXI, DMA, and decoder interfaces. Finally, emulation is performed on the FPGA to validate computational performance and functional correctness.

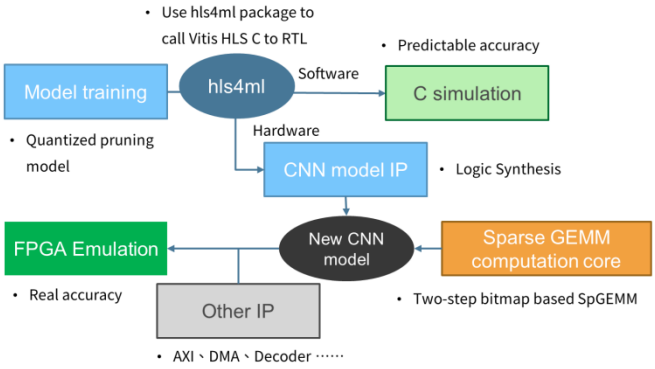

Fig. 5. Experimental flow chart of hardware and software design.

IV. RESULTS

Table 2 presents a comparison of inference accuracy between the pruning method provided by the TensorFlow Model Optimization Toolkit (TFMOT) [11] and the proposed row-balanced block pruning method, evaluated across two CNN architectures: VGG16-Light and VGG16-Large. The results indicate that our method achieves comparable accuracy to TFMOT in both software and hardware implementations. For VGG16-Light, the proposed method yields a software accuracy of 70.9% (float) and 71.5% (quantized), closely matching TFMOT's 71.8% and 72.5%. In hardware, our method maintains 71.3% (float) and 72.5% (quantized), slightly trailing TFMOT. For VGG16-Large, the accuracy differences are minimal across both methods, with the proposed pruning scheme achieving 87.1% (software float), 84.1% (software quantized), 87.0% (hardware float), and

86.1% (hardware quantized), all closely aligned with TFMOT's results. These findings demonstrate that the proposed row-balanced pruning strategy preserves inference accuracy effectively, while offering compression benefits and maintaining high sparsity, thereby confirming its practical applicability in deep learning deployments.

Table. 2 Software and hardware comparison in pruning methodology.

Method		TFMOT pruning[11]	Our study
VGG16-Light			
Software	Float	71.8 %	70.9 %
	Quantized	72.5 %	71.5 %
Hardware	Float	72.6 %	71.3 %
	Quantized	72.9 %	72.5 %
VGG16-Large			
Software	Float	87.9 %	87.1 %
	Quantized	85.0 %	84.1 %
Hardware	Float	87.1 %	87.0 %
	Quantized	86.1 %	86.1 %

Additionally, the performance of the proposed two-step bitmap compression format and its dedicated hardware architecture was evaluated in the context of sparse matrix multiplication, and compared against conventional dense matrix multiplication. By skipping unnecessary computations involving zero elements and reducing memory access, the proposed design achieves significant improvements in both computation time and memory efficiency. As illustrated in Fig. 6 and Fig. 7, the proposed sparse matrix multiplication accelerator outperforms a traditional dense accelerator, achieving computation time reductions of up to 80% and 85% at weight sparsity levels of 50% and 75%.

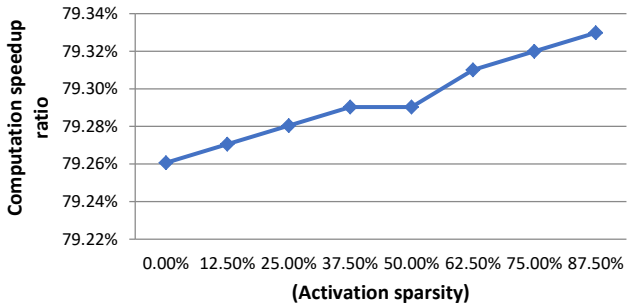

Fig. 6. Speedup ratio comparison with GEMM in weight sparsity 50%.

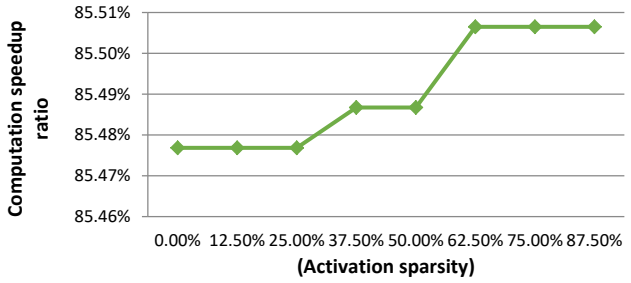

Fig. 7. Speedup ratio comparison with GEMM in weight sparsity 75%.

V. CONCLUSION AND DISCUSSION

In summary, this work combines column-balanced block pruning with two-stage bitmap compression to speed up sparse matrix multiplication and improve deep learning inference efficiency. The method reduces memory usage while maintaining accuracy and can be applied to CNNs and Transformers. Future work includes further optimizations and evaluating scalability on ASICs, GPUs, and cloud AI accelerators for efficient AI hardware.

REFERENCES

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Feb. 2017, doi: 10.1109/JPROC.2017.2761740.

[2] J. Li *et al.*, "SqueezeFlow: A Sparse CNN Accelerator Exploiting Concise Convolution Rules," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1663–1677, Nov. 2019, doi: 10.1109/TC.2019.2924215.

[3] M. Shao, A. Basit, R. Karri, and M. Shafique, "Survey of Different Large Language Model Architectures: Trends, Benchmarks, and Challenges," *IEEE Access*, vol. 12, pp. 188664–188706, 2024, doi: 10.1109/ACCESS.2024.3482107.

[4] C. Fang, W. Sun, A. Zhou, and Z. Wang, "CEST: Computation-Efficient N:M Sparse Training for Deep Neural Networks," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Apr. 2023, pp. 1–2. doi: 10.23919/DATE56975.2023.10137121.

[5] H. Peng *et al.*, "Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, Apr. 2021, pp. 142–148. doi: 10.1109/ISQED51717.2021.9424344.

[6] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, "Tensaurus: A Versatile Accelerator for Mixed Sparse-Dense Tensor Computations," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2020, pp. 689–702. doi: 10.1109/HPCA47549.2020.00062.

[7] N. S. Sattar, H. Lu, and F. Wang, "BCSR on GPU: A Way Forward Extreme-scale Graph Processing on Accelerator-enabled Frontier Supercomputer," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2024, pp. 280–289. doi: 10.1109/SCW63240.2024.00044.

[8] S. Xu, J. Jiang, J. Xu, and X. Qian, "Efficient SpMM Accelerator for Deep Learning: Sparkle and Its Automated Generator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 3, p. 38:1-38:30, Sep. 2024, doi: 10.1145/3665896.

[9] Y. Chen, A. Louri, S. Liu and F. Lombardi, "A Balanced Sparse Matrix Convolution Accelerator for Efficient CNN Training," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 71, no. 10, pp. 4638-4651, Oct. 2024, doi: 10.1109/TCSI.2024.3430831.

[10] S. Li, S. Huai, and W. Liu, "An Efficient Gustavson-Based Sparse Matrix–Matrix Multiplication Accelerator on Embedded FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4671–4680, Dec. 2023, doi: 10.1109/TCAD.2023.3281719.

[11] TensorFlow Model Optimization Team, "TensorFlow Model Optimization Toolkit: Pruning API," TensorFlow, 2020. [Online]. Available:https://www.tensorflow.org/model_optimization/guide/pruning/comprehensive_guide