

Late Breaking Results: Algorithm-Hardware Co-Design of a Sparsity-Aware Dense-Sparse Scheme for DNN Accelerators

Yueting Li*, Zhen Li*, Terry Tao Ye†, Weisheng Zhao*

* Hangzhou International Innovation Institute, Beihang University

† School of Science and Engineering, Chinese University of Hong Kong

Abstract—Deep neural networks (DNNs) in modern applications have increased the demand for energy-efficient DNN inference solutions, especially on resource-constrained platforms. However, the growing model capacity of DNNs incurs significant memory traffic and energy consumption. To address these challenges, we propose a novel solution that presents an algorithm-hardware co-design for reconfigurable DNN acceleration. This design exploits value- and bit-level sparsity to minimize memory footprint and enhance computational efficiency. To achieve this, the proposed algorithm leverages a static dense-sparse storage format, along with a dynamic bit-processing scheme that removes non-contributing bits. Building on this algorithm, a flexible processing element array is designed to perform LUT-based shift-accumulate operations, with fine-grained per-layer configurability. Experimental results show that this design yields 13–24% storage savings across the evaluated DNN models, while delivering up to 8.4× effective sparsity. Based on post-implementation FPGA results (from our RTL design), the proposed accelerator delivers 1.41× lower LUT usage than state-of-the-art design at similar throughput.

Index Terms—Algorithm-Hardware Co-design, DNN Accelerator, Energy-Efficient Computing

I. INTRODUCTION

Deep neural networks (DNNs) have evolved into high-capacity models with hundreds of millions to billions of parameters, spanning vision and language workloads [1]. The increased memory footprint and computational demand pose significant bottlenecks in deploying DNN workloads on resource-constrained platforms [2]. Although these value-level sparsity-aware accelerators can skip zero-valued operations, the limited sparsity in many DNNs restricts efficiency gains, particularly in dense scenarios [3]. To further reduce redundant computation, bit-level sparsity schemes harness zero bits within operands to eliminate non-contributing bits [4]. In practice, irregular zero-bit distributions introduce workload imbalance and limit the achievable efficiency gains [5]. The large scale of DNNs with billions of parameters presents significant challenges in maintaining inference accuracy when dealing with varying levels of sparsity.

These observations motivate an algorithm-hardware co-design that leverages value- and bit-level sparsity for energy-efficient DNN inference. The main contributions are as follows:

- 1) We propose a static dense-sparse storage format with dynamic bit processing to eliminate non-contributing bits, minimizing memory footprint while preserving precision.
- 2) We design a sparsity-aware accelerator that leverages sparsity metadata to enable fine-grained dataflow configuration, enhancing computational efficiency.

II. OVERALL DESIGN

A. Sparsity-aware Algorithm

The *static dense-sparse storage format* adopts a fixed sparsity threshold $SpTh$ to determine the storage mode for each neural network layer. For the layer sparsity, we define a mode flag δ_{ds} , where $\delta_{ds} = 1$ initiates sparse storage and $\delta_{ds} = 0$ activates dense-mapping storage. A metadata header $(S_{addr}, Idx_{addr}, E_{addr}, \delta_{ds})$ specifies the start and end addresses for the value and index blocks in memory region. The *value block* $[S_{addr}, Idx_{addr})$ stores all nonzero quantized values and records δ_{ds} depending on $SpTh$. The *index block* $[Idx_{addr}, E_{addr})$ records nonzero indices in sparse mode and zero indices in dense-mapping mode. To further exploit sparsity with modest hardware overhead, the *dynamic bit-processing scheme* (i) extracts nonzero-bit positions T_{bitpos} using LUT-based bit scanning, and (ii) computes the corresponding nonzero-bit count N_{nzb} for metadata-guided execution. This sparsity-aware algorithm generates a per-layer metadata header that enables energy-efficient execution on the DNN accelerator.

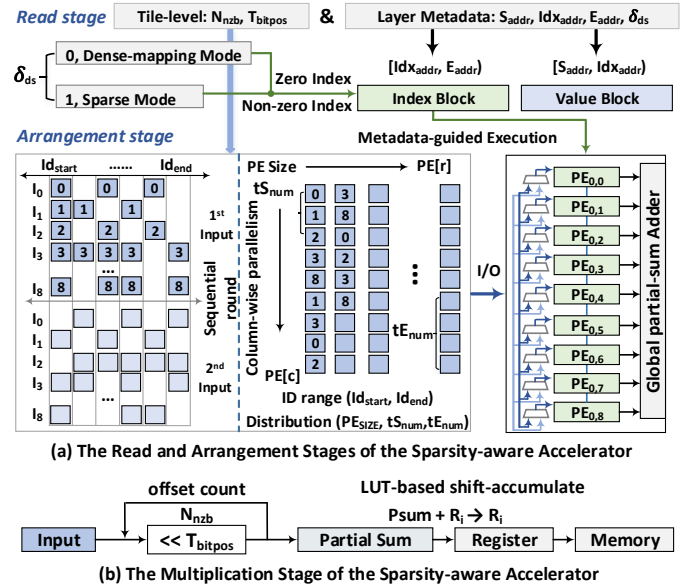


Fig. 1. The Proposed Sparsity-aware Accelerator: (a) Read and Arrangement Stages and (b) Multiplication Stage.

B. Sparsity-aware Accelerator

The sparsity-aware accelerator comprises *read*, *arrangement*, and *multiplication* stages, and supports a configurable dataflow for DNN inference. In the *read* stage, the accelerator

fetches weights and their corresponding inputs under the control of the layer metadata header ($S_{addr}, Idx_{addr}, E_{addr}, \delta_{ds}$). It then applies the *dynamic bit-processing scheme* to the nonzero weights and generates $[T_{bitpos}, N_{nzb}]$ for metadata-guided execution. In the *arrangement* stage, the accelerator organizes the $[T_{bitpos}, N_{nzb}]$ metadata of layer-level weights, which are fed into the accelerator on a per-PE-column basis, as shown in Fig. 1(a). A global configuration record, *tile_info*, specifies the tile ID range (Id_{start}, Id_{end}), which updates the executed weight block and provides a distribution tuple ($PE_{size}, tS_{num}, tE_{num}$). Based on *tile_info*, the accelerator loads T_{bitpos} and N_{nzb} into PE-aligned buffers and accordingly partitions the corresponding input arrays. The edge PE columns are parameterized by tS_{num}, tE_{num} (valid tile spans in the first and last columns) to accommodate incomplete tiles, while the remaining columns are assigned based on the tile size.

With this column allocation, the proposed accelerator iterates over $Id \in [Id_{start}, Id_{end}]$ to pack ($\delta_{ds}, T_{bitpos}, N_{nzb}$) and dispatch input tiles in sequential rounds, while local re-ordering ensures data alignment. The *multiplication* stage implements LUT-based shift-accumulate on a weight-stationary systolic array. During execution, (T_{bitpos}, N_{nzb}) are updated on the fly to reflect the active nonzero-bit positions. For each tile, the corresponding *inputs* are loaded according to the range $[Id_{start}, Id_{end}]$ and broadcast to the PE row (PE[r]) and column (PE[c]). As shown in Fig. 1(b), each PE consumes the input stream and the bit-metadata stream (T_{bitpos}, N_{nzb}) to compute partial products and accumulate them into a local register R_i (i.e., $P_{sum} + R_i \rightarrow R_i$). In the proposed design, the N_{nzb} controls the offset count, simplifying dataflow control. The accumulated results are then streamed from the output buffers to external memory.

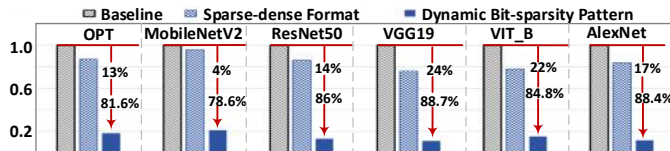


Fig. 2. The significant-bit ratio in the static dense-sparse storage format and dynamic bit processing scheme across OPT, MobileNetV2, ResNet50, VGG19, ViT_B, and AlexNet models, the blue bars correspond to effective bit-level sparsity (up to 8.4x).

III. EXPERIMENTAL RESULTS

In this section, we present FPGA post-synthesis results for the RTL design at 200 MHz, focusing on resource usage under matched throughput. Fig. 2 shows that the proposed sparsity-aware algorithm achieves 13–24% storage savings across the evaluated models, while delivering up to 8.4 \times effective sparsity (defined as $\text{BitOps}_{full}/\text{BitOps}_{exec}$). The sparsity-aware algorithm eliminates non-contributing bits, demonstrating its effectiveness across various DNN models.

Table I reports the per-module resource breakdown of the sparsity-aware accelerator for the *read*, *arrangement*, and *multiplication* stages. The *read* stage primarily accesses the *value*

TABLE I
HARDWARE RESOURCE UTILIZATION OF THE PROPOSED DESIGN

Module	BRAM	DSP	FF (K)	LUT (K)	Cycle
Read	22	0	1.6	3.2	5
Arrangement	0	4	7.2	72.7	14
Multiplication	0	0	12.1	53.2	10

TABLE II
COMPARISON WITH STATE-OF-THE-ART DESIGNS

Metric	Amoeba [6]	Ours	Saving
BRAMs	1036	98	10.57 \times
DSPs	522	11	47.45 \times
LUTs (K)	143	101.4	1.41 \times
FFs (K)	N/A	20.1	N/A
Throughput (GOPS)	169.7	170.84	-

and *index* blocks in the static dense-sparse storage format, utilizing 22 BRAMs for weights, indices, and corresponding inputs. Amoeba is a flexible FPGA-based inference accelerator that supports CNNs with arbitrary kernel sizes through a configurable dataflow. In contrast, Table II reports post-integration end-to-end resources, which include shared control and interconnect overhead. Table II compares this design with the state-of-the-art Amoeba [6], showing significant resource savings (10.57 \times for BRAM, 47.45 \times for DSPs, and 1.41 \times for logic resource usage) while maintaining comparable throughput performance.

IV. CONCLUSION

This design presents a novel algorithm-hardware co-design solution that exploits both value-level and bit-level sparsity. Compared with a state-of-the-art design, it achieves a 10.57 \times reduction in BRAM usage, a 47.45 \times reduction in DSPs, and a 1.41 \times reduction in LUTs. This design remains lightweight and is suitable for integration into highly energy-efficient cores.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (62404016), the China Postdoctoral Science Foundation (2024M754061), the Zhejiang Key Laboratory of Intelligent Electromagnetic Control and Advanced Electronic Integration (ZJUAEI3009), and the Hangzhou International Innovation Institute of Beihang University (2024BKZ005).

REFERENCES

- [1] M. Xu et al., “Resource-efficient Algorithms and Systems of Foundation Models: A Survey,” *ACM Comput. Surv.*, vol. 57, no. 5, pp. 1-39, 2025
- [2] F. Masar et al., “FPGA-Based Emulation and Fault Injection for CNN Inference Accelerators,” *IEEE DATE*, pp. 1-2, 2025
- [3] W. Wang et al., “SparDR: Accelerating Unstructured Sparse DNN Inference via Dataflow Optimization,” *IEEE DATE*, pp. 1-7, 2025
- [4] J. Yang et al., “SparSynergy: Unlocking Flexible and Efficient DNN Acceleration Through Multi-Level Sparsity,” *IEEE DATE*, pp. 1-7, 2025
- [5] Y. Li et al., “ASP: Adaptive Sparse LUT-based Bit-Slice Accelerator for Efficient CNN Inference,” *IEEE ISCAS*, pp. 1-4, 2025
- [6] X. Wu et al., “Amoeba: An Efficient and Flexible FPGA-Based Accelerator for Arbitrary-Kernel CNNs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.32, no.6, pp. 529-551, 2024