Towards Efficient SRAM-PIM Architecture Design by Exploiting Unstructured Bit-Level Sparsity*

Cenlin Duan[1], Jianlei Yang[1], Yiou Wang[1], Yikun Wang[1], Yingjie Qi[1], Xiaolin He[1], Bonan Yan[2], Xueyan Wang[1], Xiaotao Jia[1], Weisheng Zhao[1]
[1]Beihang University, Beijing, China [2]Peking University, Beijing, China

Abstract

Bit-level sparsity in neural network models harbors immense untapped potential. Eliminating redundant calculations of randomly distributed zero-bits significantly boosts computational efficiency. Yet, traditional digital SRAM-PIM architecture, limited by rigid crossbar architecture, struggles to effectively exploit this unstructured sparsity. To address this challenge, we propose Dyadic Block PIM (DB-PIM), a groundbreaking algorithm-architecture co-design framework. First, we propose an algorithm coupled with a distinctive sparsity pattern, termed a dyadic block (DB), that preserves the random distribution of non-zero bits to maintain accuracy while restricting the number of these bits in each weight to improve regularity. Architecturally, we develop a custom PIM macro that includes dyadic block multiplication units (DBMUs) and Canonical Signed Digit (CSD)-based adder trees, specifically tailored for Multiply-Accumulate (MAC) operations. An input pre-processing unit (IPU) further refines performance and efficiency by capitalizing on block-wise input sparsity. Results show that our proposed co-design framework achieves a remarkable speedup of up to 7.69× and energy savings of 83.43%.

Keywords

Bit-level Sparsity, SRAM, PIM, Algorithm/Architecture Co-design

1 Introduction

Deep Neural Networks (DNNs) have become pervasive in numerous applications, including image recognition [1], speech recognition [2], and object detection [3]. However, the substantial memory and computing requisites pose challenges to performance and efficiency, especially for resource-constrained devices. Processing-in-memory (PIM), as an innovative computational paradigm, offers a potential solution to these challenges. Distinct from the *Von Neumann* architecture, PIM executes multiply-accumulate (MAC) operations in memory, thereby eliminating the *Memory Wall* caused by frequent data movement. Previous studies have shown that various technologies, such as SRAM [4, 5], RRAM [6, 7], and MRAM [8–10], are available candidates for PIM. Among these technologies, SRAM is widely used in academia and industry due to its faster write speed, lower write energy, and compatibility with the most advanced process technologies.

Current SRAM-PIM research focuses on sparsity support to further improve computational efficiency. The majority of these studies have been geared towards leveraging value-level sparsity. Bit-level sparsity, however, can further eliminate the redundant computation associated with zero bits in the values, which garnered significant interest. It is worth noting that although bit-level sparsity has been explored and applied in traditional digital accelerators, its efficient utilization in SRAM-PIM still presents numerous challenges.

Computational dependency issues. Unlike traditional digital accelerators, PIM-based accelerators are constrained by their rigid crossbar structure [11]. This structure enforces strict data routing for input broadcast and vertical output accumulation, thereby impeding the efficient utilization of the randomly distributed zero bits. As shown in Fig. 1(a), despite the high prevalence of zero bits, their random distribution prevents the system from efficiently bypassing them during computations, which dilutes the potential computational efficiency gains. The challenge is particularly pronounced in the highly parallel architecture of digital SRAM-PIM, where the entire array is activated and computes concurrently.

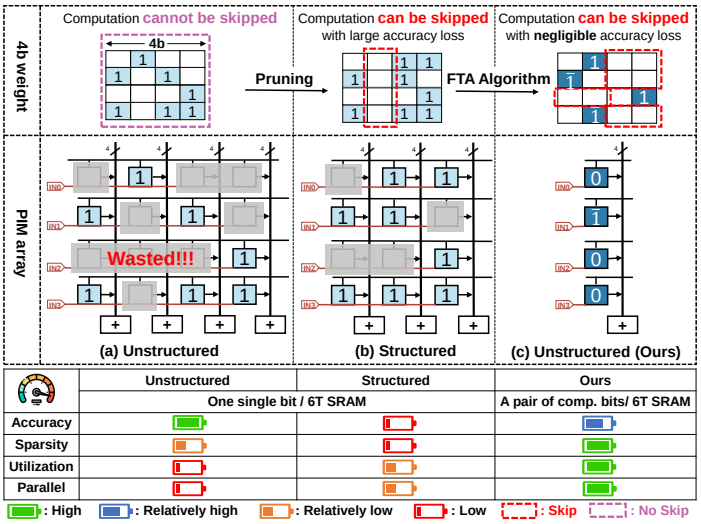

Figure 1: Exploitation of bit-level sparsity in SRAM-PIMs.

Low utilization issues. To solve the above problem, one potential method is structured bit-level pruning, as shown in Fig. 1(b). However, this method eliminates only a minimal number of zero bits, concurrently resulting in significant accuracy degradation. As a result, the retention of zero-bit mapping within the PIM array results in a multitude of ineffectual computations. Despite various studies proposing refined mapping strategies to boost array utilization, pervasive bit-level sparsity still hinders optimal utilization. To better quantify this issue, we define a particular *actual utilization* by

$$\mathcal{U}_{act} = \frac{EffectiveCompSRAMCells}{TotalCompSRAMCells} \times 100\%. \quad (1)$$

It represents the ratio of SRAM cells engaged in effective computation (computing non-zero bit) to the total SRAM cells currently involved in the computation. This metric helps assess the utilization efficiency of SRAM-PIM architectures in handling sparsity. For instance, as illustrated in Fig. 1, the utilization rates are 7/16 and 8/12, respectively. In contrast, our approach achieves a 4/4 ratio by using a 6T SRAM cell to store a pair of complementary (comp.) bits for parallelism, thus efficiently addressing the challenge of bit-level sparsity and enhancing computational efficiency.

We observe that the cross-coupled structure of 6T SRAM, along with in-memory customization features, provides a unique avenue for utilizing unstructured bit-level sparsity. Based on this observation, we propose an efficient algorithm/architecture co-design framework to overcome the above two fundamental challenges. The contributions of this work are as follows:

- We propose Dyadic Block PIM (DB-PIM), the first algorithm and architecture co-design framework tailored for digital SRAM-PIM that effectively harnesses the unstructured bit-level sparsity.

*This work is supported in part by National Natural Science Foundation of China (Grant No. 62072019) and National Key Laboratory of Spintronics. Corresponding authors are *Jianlei Yang* and *Weisheng Zhao*, Email: jianlei@buaa.edu.cn, weisheng.zhao@buaa.edu.cn

DAC '24, June 23–27, 2024, San Francisco, CA, USA
2024. ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655948

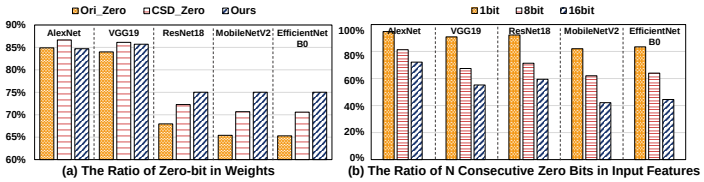

Figure 2: Bit-level sparsity existed in weights (W) and input features (I) among different models.

- At the algorithm level, we present a Fixed Threshold Approximation (FTA) algorithm, alongside a unique bit-level sparsity pattern termed dyadic block (DB), both employing the Canonical Signed Digit (CSD) encoding method. This approach maintains the random distribution of non-zero bits for accuracy while restricting the number of non-zero bits in each weight to improve regularity. It provides a foundation for solving computational dependency and low utilization issues in rigid crossbar structures.
- At the architecture level, we design a customized PIM macro that incorporates dyadic block multiply units (DBMUs) and CSD-based adder trees, specifically designed for efficient MAC operations on randomly distributed non-zero bits. Additionally, our architecture dynamically detects sparse blocks from input features, bypassing all-zero-bit blocks to enhance computational efficiency.

The rest of this paper is organized as follows. Section 2 provides background and motivation. Section 3 demonstrates the details of the proposed methodology. Experimental results are illustrated in Section 4 and the conclusion is given in Section 5.

2 Background and Motivation

2.1 Richness in Bit-level Sparsity

Limitation on Value-level Sparsity. Recent studies demonstrate that a significant proportion of zero values exist within weights and input features of neural network (NN) models. These zero values inherently contribute nothing to the final computational results. Thus, skipping the computations associated with these zero values can lead to considerable savings in computational resources, thereby boosting efficiency. However, the proportion of zero values in NN models is finite. This indicates the full potential of computational efficiency gains, achievable by bypassing unnecessary computations, has not yet been completely tapped. Consequently, research is increasingly shifting focus from the value-level sparsity to the finer granularity of bit-level sparsity. This transition not only opens new avenues for optimization but also aligns with the pressing need for more efficient computational paradigms.

Opportunity on Bit-level Sparsity. By delving into the bit-level sparsity within values, we uncover substantial untapped potential. For example, the multiplication of `INT8` weight (W) and input feature (I) can be broken down into 64 individual $1b \times 1b$ operations, as shown in Eq. (2). Only those pairings where both I_i and W_j are non-zero effectively contribute to the final results.

$$I * W = \sum_{i=0}^{7} \sum_{j=0}^{7} I_i \times W_j \ . \tag{2}$$

However, the proportion of non-zero bits in both weights and input features is relatively low, indicating that most of these computations are ineffectual. Fig. 2(a) illustrates the zero-bit proportion across various NN models. Even when employing compact NN models such as MobileNetV2 and EfficientNetB0, bit-level sparsity still reaches a significant rate of approximately 65%. Bypassing these redundant calculations can significantly enhance computational efficiency and open new avenues for model optimization. To further exploit the potential of bit sparsity, CSD encoding—a distinctive digital representation is proposed, characterized by its prohibition of consecutive non-zero bits. By employing CSD encoding, the overall sparsity is increased by an additional 5% compared to the initial level of sparsity. For compact models, our algorithm can improve by an additional 5% based on CSD encoding.

Fig. 2(b) demonstrates bit-level sparsity in input features. However, fully exploiting all non-zero bits is impractical due to substantial hardware overhead. Our analysis indicates that when input features are grouped in sets of 8 or 16, the probability of identical bit positions being zero across the group is relatively high (up to 80% in groups of 8, and around 70% in groups of 16). Bypassing columns comprised entirely of zeros can yield efficiency gains in computation.

Table 1: Sparsity Exploitation Comparison among SRAM-PIMs.

	Yue et al. [12]	SDP [11]	Liu et al. [13]	Tu et al. [14]	TT@CIM [15]	Ours
Sparsity Type	Value (V)			Bit (B)		
Weight/Input (W/I)	W	W	W	I	W	$W + I$
Digital/Analog (D/A)	A	D	D	D	A	D
Unstruct./Struct. (U/S)	S	S	U	U	U	U
Ineffectual MAC Removed	Zero $W + V$	Zero $W + V$	Zero $W + V$	Zero $I + B$	Zero $W + B$	Zero $W + B$ and Zero $I + B$

2.2 Sparsity Exploitation in SRAM-PIM

Value-level Sparsity. Value-level sparsity has been extensively explored in SRAM-PIM architectures, as summarized in Tab. 1. For instance, '$W + V$' represents skipping multiplications where the weight value is zero, while '$I + B$' represents skipping multiplications where the input feature bit is zero. Yue et al. [12] pioneered the implementation of block-wise zero-skipping in analog SRAM-PIM architectures, laying the groundwork for sparsity optimization. However, due to ADC limitation, only part of the cell array is activated simultaneously. To address this limitation, SDP [11] proposes a novel digital SRAM-PIM with a double-broadcast hybrid-grained pruning method, activating all rows simultaneously for enhanced efficiency. Building on these developments and to further mitigate accuracy loss, Liu et al. [13] present a butterfly-network-based zero skipper for unstructured NN models. However, efficiently exploiting such unstructured pruning techniques in PIM entails a significant increase in hardware overhead. Despite the significant strides made by these pioneering studies, the prevalence of zero bits in values continues to hinder the full potential of efficiency and utilization in SRAM-PIM architectures.

Bit-level Sparsity. In Sec. 2.1, we explore the substantial potential of bit-level sparsity across various NN models. Recent research has increasingly focused on this domain. Tu et al. [14] developed the Bandwidth-Balanced CIM (BB-CIM) architecture to address computational imbalances caused by input bit-level sparsity, aiming to equalize input bit-width for improving efficiency. Similarly, TT@CIM [15] seeks to enhance efficiency by analyzing and leveraging the higher sparsity ratio of zero bits in both two's complement and one's complement representations. Although numerous studies have attempted to explore the utilization of unstructured zero bits in digital SRAM-PIM, current methodologies still fall short of fully harnessing the potential of these randomly distributed zero bits. As demonstrated in Fig. 1(a), the majority of zero bits still need to be stored and processed in SRAM-PIM, resulting in a substantial number of ineffective calculations and low utilization. Fig. 1(b) illustrates that the structured bit-level pruning method can overcome the above limitations. Yet, for digital SRAM-PIM with high parallelism, a prerequisite is that a significant volume of bits must simultaneously be zero at identical locations. Such a pruning method will introduce significant accuracy loss and does not eliminate all zero bits, impeding the achievement of optimal array utilization.

Opportunity: Presently, digital SRAM-PIM is predominantly optimized for limited unstructured or structured bit-level sparsity, yielding only slight improvements. To bridge this gap, we propose a DB-PIM framework. It aims to comprehensively exploit unstructured bit-level sparsity, signifying a paradigm shift in PIM design. **By CSD encoding, FTA algorithm, and DB-PIM architecture, we selectively store and compute effective non-zero bits. This approach significantly reduces invalid computations, enhancing efficiency, and simultaneously achieves considerably higher utilization.**

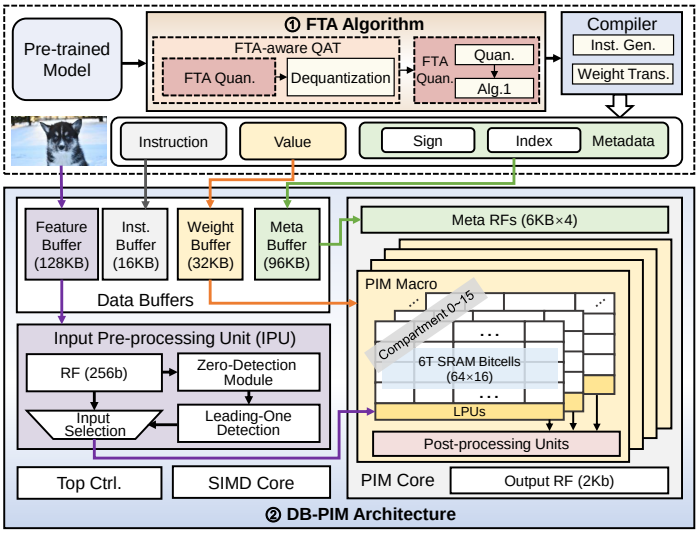

Figure 3: Overview of the proposed DB-PIM, an algorithm and architecture co-design framework.

Algorithm 1: Fixed Threshold (Φ_{th}) Approximation (FTA)

Input: Quantized filters $\mathcal{F} \doteq [f_0, \ldots, f_{n-1}]$, where $f_i \in D^{\mathcal{N}}$, n is the number of filters, $\mathcal{N}$ is the number of elements in one filter, D is determined by the quantization precision, e.g. INT8, Query Table $T(\phi^{th}) = \{t \in D \mid \phi(toCSD(t)) = \phi_i^{th}\}$

Output: Approximation filters $\mathcal{F}^{th} \doteq [f_0^{th}, \ldots, f_{n-1}^{th}]$, filter thresholds $\Phi^{th} \doteq [\phi_0^{th}, \ldots, \phi_{n-1}^{th}]$.

for i ***in*** $[0, 1, 2, \ldots, n-1]$ **do**
  for j ***in*** $[0, 1, 2, \ldots, \mathcal{N}-1]$ **do**
    $f_i^{csd}(j) \leftarrow \text{toCSD}(f_i(j))$ // CSD Conversion
    $\phi_i(j) \leftarrow \text{CountNonZeros}(f_i^{csd}(j))$ //Count Non-Zero Bit
  end
  $m_i \leftarrow Mode\ \{0 \le j < \mathcal{N} \mid \phi_i(j)\}$ //Compute Mode
  if $\forall j, \phi_i(j) == 0$ **then**
    $\phi_i^{th} \leftarrow 0$ // All Zero Filter
  else if $m_i == 0$ **then**
    $\phi_i^{th} \leftarrow 1$
  else if $1 \le m_i \le 2$ **then**
    $\phi_i^{th} \leftarrow m_i$
  else if $m_i > 2$ **then**
    $\phi_i^{th} \leftarrow 2$ // Limit Max Threshold to 2
  for j ***in*** $[0, 1, 2, \ldots, \mathcal{N}-1]$ **do**
    $f_i^{th}(j) \leftarrow \underset{t \in T(\phi^{th})}{\arg\min} |t - f_i(j)|$ // Closest Num to $f_i(j)$ in T
  end
end

3 Methodology

3.1 Overall Framework

Fig. 3 illustrates the overview of the DB-PIM, an algorithm and architecture co-design framework that effectively resolves the challenges discussed earlier. We first leverage the ① FTA algorithm and a bit-level sparsity pattern to obtain a weight matrix with a fixed number of non-zero bits. This approach maintains model accuracy by preserving the randomness of the non-zero bits and enhances data regularity by limiting the number of non-zero bits in each weight. It forms the foundation for our framework to handle unstructured non-zero bits efficiently. Subsequently, we develop a dedicated ② DB-PIM architecture, designed to accelerate these patterns. This cohesive integration of the FTA algorithm with the DB-PIM architecture effectively addresses computational dependencies typically found in rigid crossbar structures, enabling efficient computation of randomly distributed non-zero bits. Consequently, this enhances hardware efficiency and optimizes array utilization. Specifically, in the training procedure, we first apply a modified Quantization-Aware Training (QAT), known as FTA-aware QAT, to the pre-trained model. This step is crucial for obtaining quantization parameters that reflect the impact of our algorithm on model accuracy. Following this, we perform FTA quantization to obtain approximation models based on our FTA algorithm. We then transform these models into values and metadata (including signs and indices) and generate corresponding instructions in the compilation phase. This compilation is conducted offline and stores the above information in off-chip memory. Finally, DB-PIM performs bitwise AND operations with these values and recovers the final results based on the metadata in the customized PIM macros.

3.2 FTA Algorithm

CSD Representation for Weight. The CSD encoding, utilized in our algorithm, is a binary number representation system that employs three possible digit values: 1, 0, or −1 (with −1 often represented by $\bar{1}$). A hallmark characteristic of CSD is the constraint that adjacent bits cannot be both non-zero bits (1, $\bar{1}$). This restriction ensures each number has a unique representation with a minimized count of non-zero bits. On average, the CSD representation contains 33% fewer non-zero bits than their binary equivalents, enhancing its bit-level sparsity and potentially reducing computational overhead [16]. For example, the binary number 0111_1101_2 would be encoded in CSD as $1000\_0\bar{1}01_{CSD}$. Incorporating CSD encoding into our framework is driven by two primary factors. First, CSD encoding significantly improves bit-level sparsity, potentially improving computational efficiency. Secondly, its inherent property of preventing the occurrence of consecutive non-zero bits is crucial for our DB-PIM architecture. This attribute aligns seamlessly with the hardware constraints and optimizes the architecture's operational efficiency, making it an ideal choice for our DB-PIM framework.

Bit-level Sparsity Pattern. Based on CSD encoding, we propose a novel bit-level sparsity pattern termed the dyadic block (DB). This pattern partitions an 8-bit binary number into four DBs, each consisting of a pair of bits. As a fundamental unit of our encoding scheme, the DB is allocated a distinctive index to denote its position, which is crucial for accurately locating and processing non-zero bits. In our CSD encoding, the DB can be divided into two categories: Zero Pattern block (00) and Complementary Pattern (Comp. Pattern) blocks, which include 01, 10, $0\bar{1}$, and $\bar{1}0$. To effectively quantify sparsity within our framework, we introduce a symbol ϕ to represent the number of non-zero bits, which ranges from 0 to 4 and corresponds to sparsity levels from 100% down to 50%. As depicted in Fig. 4, the 8-bit binary number $f_1^{th}(0) = 0\bar{1}00\_0010_{CSD}$ decomposes into four DBs: '$0\bar{1}|00|00|10$', assigning each pair an index, '$DB\#3|DB\#2|DB\#1|DB\#0$'. $\phi_1(0) = 2$ reflects the two non-zero bits in this value and 75% sparsity.

The rationale for adopting this pattern is our discovery of high compatibility between the Comp. Pattern block and the cross-coupled structure in 6T SRAM cell. Specifically, a Comp. Pattern block can be stored within a single 6T SRAM cell and computed simultaneously. For each 8-bit value, we could eliminate all Zero Pattern blocks and only store and compute the Comp. Pattern blocks. This approach not only preserves the unstructured characteristic of bit-level sparsity but also capitalizes on the cross-coupled structure to ensure the utilization efficiency of the SRAM cell array. In this system, all SRAM cells involved in the computation are effectively utilized, addressing **the low utilization issues**, as shown in Fig. 1. However, merely exploiting this pattern falls short of resolving **the computational dependency issues**. The reason lies in the variability of ϕ values across the weights in each filter. Indiscriminate removal of all Zero Pattern blocks might result in computational irregularity. Such irregularity conflicts with the structured computation requirement of a rigid crossbar architecture. To handle the computational dependency issues effectively, we propose the FTA algorithm.

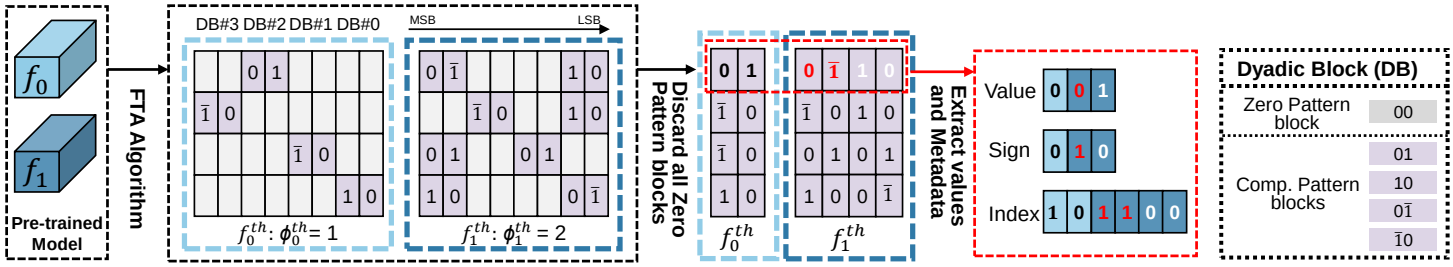

Figure 4: Extraction and representation of bit-level sparsity patterns.

Detailed Procedure of FTA Algorithm. The core idea of our algorithm lies in setting a uniform threshold, denoted as ϕ^{th} for each filter. This threshold is pivotal in ensuring that weights in each filter maintain a uniform count of non-zero bits. Despite the random distribution of these non-zero bits, this consistency guarantees the elimination of the same number of Zero Pattern blocks. As a result, it upholds the regular structure of each compressed filter, as shown in Fig. 4. Specifically, the processing of this algorithm is shown in Alg. 1. First, the entire layer is grouped based on the number of filters, with the weights in these groups subsequently converted into CSD representation. Next, we determine a threshold ϕ^{th} for each filter by analyzing the distribution of the number of non-zero bits across all weights within the filter. Weight distribution analysis in various NN models reveals that a ϕ^{th} value of 2 is the most prevalent across the filters. Therefore, to enhance sparsity while avoiding a significant accuracy drop, we confine the ϕ^{th} from 0 to 2. The FTA algorithm sets the value of $f_i^{th}(j)$ as the closest value to $f_i(j)$ in set $T(\phi^{th})$. In conclusion, combining the FTA algorithm and sparsity pattern lays the foundation for addressing the prevalent computational dependencies and low utilization issues in SRAM-PIM.

Our proposed FTA algorithm demonstrates significant potential in alleviating the bit-level irregularity by setting the uniformed ϕ^{th}. However, this potential remains underutilized in current SRAM-PIM architectures due to several limitations. First, the current PIM macro cannot execute parallel computations on the complementary states, denoted as $Q/\overline{Q}$, stored in the cross-coupled structure of 6T SRAM. Second, the existing adder trees cannot directly accumulate outputs with randomly distributed non-zero bits. Third, the current PIM macro does not have the functionality to identify and bypass zero bits in input features during runtime. As such, the DB-PIM architecture is specifically designed to capitalize on the novel opportunities offered by the FTA algorithm to enhance computation efficiency.

3.3 Architecture Design of DB-PIM

Top Level Architecture. Fig. 3 ② illustrates the overall architecture of DB-PIM, which is composed of a top controller, an input pre-processing unit (IPU), data buffers, a PIM core, and a SIMD core. The PIM core consists of metadata register files (RFs) for storing sign and index information fetched from meta buffer, four customized PIM macros, and an output RF. The PIM macro is an extension of the ADC-less SRAM PIM macro proposed in [17]. The top controller first processes instructions fetched from the instruction buffer (Inst. Buffer) and dispatches corresponding control signals to the whole system. The input features, stored in the feature buffer, are accessed by IPU for converting into bit-serial form. IPU identifies and eliminates all-zero sequences, subsequently broadcasting all non-zero sequences along with their index information to the PIM core. The PIM core, receiving weights from the weight buffer and input features from the IPU, executes bitwise AND operations. Subsequently, the results of these operations are accumulated in a customized CSD-based adder tree, guided by metadata derived from the meta RFs. These MAC results are then shifted and accumulated based on their respective bit position obtained from IPU for producing the partial sum (Psum). Then, we accumulate Psum, and the final results are written back in the feature buffer. The SIMD core is responsible for performing other element-wise operations.

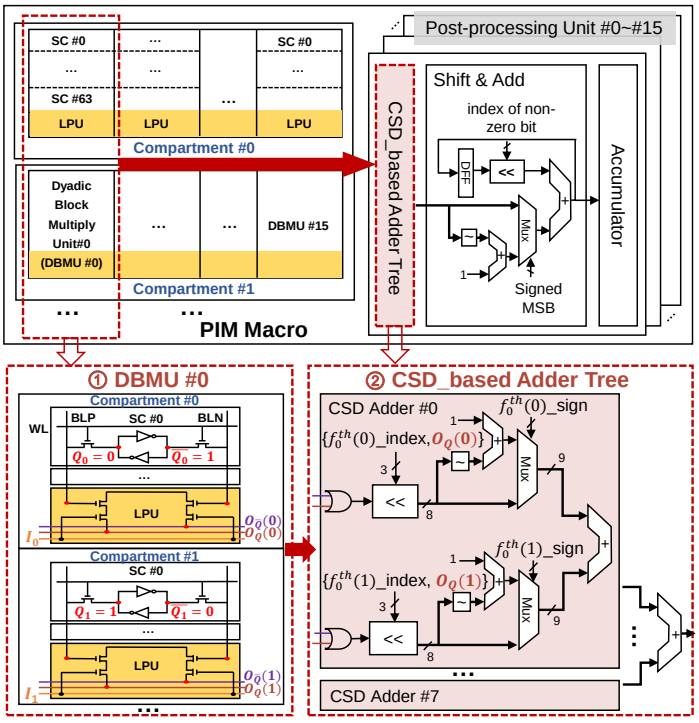

Figure 5: Circuit design of customized SRAM-PIM macro.

Customized SRAM-PIM Macro. Fig. 5 showcases the circuit design of our customized SRAM-PIM macro, which consists of 16 compartments, 16 post-processing units, and other peripheral circuits. Each compartment comprises 16 DBMUs, including sixty-four 6T SRAM cells (SC #0 ~ SC #63) and one local processing unit (LPU). The allocation of weights per row in each compartment is determined by both compartment width and the ϕ^{th} of each filter, equating to 8 for $\phi^{th} = 2$ and 16 for $\phi^{th} = 1$. In this setup, each LPU within a DBMU acts as a fundamental dyadic block multiplier, utilizing four transistors for two independent multiplications, $IN \times Q$ and $IN \times \overline{Q}$. Specifically, a Comp. Pattern block, stored in a cross-coupled structure (Q and $\overline{Q}$) within the 6T SRAM cell of DBMU, executes two individual bitwise AND operations with identical input.

Consider $f_0^{th}(0) = 0001\_0000_{CSD}$ and $f_0^{th}(1) = \overline{1}000\_0000_{CSD}$, as shown in Fig. 4 and Fig. 5. We eliminate all Zero Pattern blocks and keep Comp. Pattern blocks with their corresponding indices and signs. Then, we store 01 (DB #2 of $f_0^{th}(0)$) into $Q_0/\overline{Q_0}$ with $sign = 0$ and $index = 10$, and 10 (DB #3 of $f_0^{th}(1)$) into $Q_1/\overline{Q_1}$ with $sign = 1$ and $index = 11$ in Fig. 5 ① DBMU #0. Subsequently, I_0 and I_1 are sent to Compartment #0 and #1 for bitwise AND operations. $O_Q(0) = Q_0 \& I_0$, $O_{\overline{Q}}(0) = \overline{Q}_0 \& I_0$, $O_Q(1) = Q_1 \& I_1$, $O_{\overline{Q}}(1) = \overline{Q}_1 \& I_1$. However, it is noteworthy that directly adding $\{O_Q(0), O_{\overline{Q}}(0)\}$ and $\{O_Q(1), O_{\overline{Q}}(1)\}$ would yield an incorrect results. For instance, if I_0 and I_1 are both 1, the sum would incorrectly

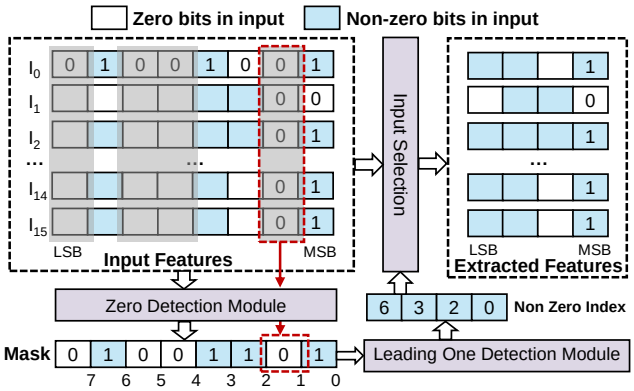

Figure 6: Bit-level sparsity utilization for input features.

Table 2: Top-1 Accuracy Comparison on CIFAR100 Dateset.

Models	W/I Precision	Ori. Accu.	FTA Accu.	Accu. Drop
AlexNet	8b/8b	70.62%	69.64%	0.98%↓
VGG19	8b/8b	76.78%	76.14%	0.64%↓
ResNet18	8b/8b	79.65%	79.09%	0.56%↓
MobileNetV2	8b/8b	82.20%	82.04%	0.16%↓
EfficientNetB0	8b/8b	72.19%	71.67%	0.52%↓

be 11. However, the correct result of $f_0^{th}(0) + f_0^{th}(1) = 0001\_0000_{CSD} + \overline{1}000\_0000_{CSD} = 1\_1001\_0000_2$. Thus, we have engineered a specialized ② CSD_based adder tree, adept at handling randomly distributed non-zero bits. This innovation, integrated into our customized PIM macro and combined with our algorithm, allows us to exploit unstructured bit-level sparsity in digital SRAM.

Input Pre-processing Unit. Recognizing that the number of non-zero bits varies across different input features, it is impractical to bypass all zero bits directly within a rigid crossbar structure. However, the data analysis presented in Fig. 2(b) reveals that block-wise zero bits still account for a large proportion. To harness the bit sparsity in the input feature, we propose IPU, designed to dynamically detect block-wise zero bits, as shown in Fig. 6. Initially, the IPU identifies columns consisting entirely of zero bits and generates a corresponding mask. Then, we detect the first non-zero bit in the above mask to select input features for calculations, thereby enhancing computational efficiency.

4 Evaluation Results

4.1 Experimental Setup

Hardware Implementation. DB-PIM is evaluated on 28 nm technology, with a 128 KB feature buffer, a 16 KB instruction buffer, a 32 KB weight buffer, a 96 KB meta buffer, four 6 KB meta RFs and four 16 Kb PIM macros. The power consumption, latency, and area of PIM macros are extracted from the post-layout of customized design extension from [17]. The remaining digital modules are implemented with Verilog HDL and synthesized by Design Compiler for area evaluation, while PrimeTime PX obtains power consumption. Aiming to evaluate the performance of DB-PIM, we implement a simple compilation tool for dataflow mapping and a customized cycle-accurate C++ simulator for functionalities validation.

Dense Digital PIM Baseline. To evaluate the benefits and associated overhead of our proposed techniques, we establish a dense digital PIM baseline for comparison. This baseline, obtained by removing all the sparsity support from the DB-PIM architecture, consists of buffers, the SIMD core, PIM macros, an output RF, and IPU. Within this baseline, the PIM macro is similar to the state-of-the-art digital PIM [17], while all other hardware configurations are the same as in the DB-PIM.

4.2 Evaluation of FTA Algorithm

To demonstrate the generality of the FTA algorithm, we conduct evaluations across various NN models on the CIFAR100 dataset. These models include standard NN models such as AlexNet [1], VGG-19 [18] and ResNet-18 [19], along with compact NN models such as MobileNetV2 [20] and

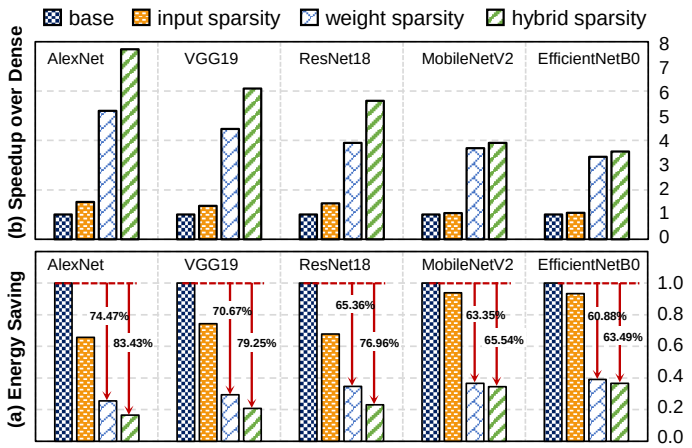

Figure 7: Speedup and energy savings of different sparsity exploration approaches over the dense PIM baseline.

EfficientNetB0 [21]. As detailed in previous research [11], compact NN models exhibit markedly less redundancy compared to standard models. This inherent attribute suggests that traditional value-level sparsity methods are effective only on certain layers to maintain accuracy. Thus, these methods, while effective in standard NN models, do not yield significant acceleration when applied to compact NN models. Conversely, Tab. 2 details the accuracy loss associated with our algorithm. Importantly, the accuracy loss remains below 1% even when applying the FTA algorithm to all layers of compact NN models. This underscores the effectiveness of our algorithm in various NN models with negligible accuracy loss.

4.3 Evaluation for Hybrid Sparsity

The accuracy comparison in Sec. 4.2 demonstrates that the accuracy degradation of our FTA algorithm is minimal compared to the original quantization model. To further explore the hardware benefits of our methods, as depicted in Fig. 7, we present a detailed analysis of speedup and energy saving over the digital PIM baseline. Fig. 7(a) illustrates the significant speedup achieved by DB-PIM across various NN models, due to its capability to exploit bit-level sparsity. Specifically, DB-PIM achieves a speedup of about 5.20× for AlexNet and 4.46× for VGG19 by utilizing weight sparsity, respectively. Driven by high redundancy, the filter thresholds (ϕ^{th}) of most convolutional (Conv) layers and fully connected (FC) layers in AlexNet can be set to 1 through our FTA algorithm. Consequently, this setting allows each PIM macro within the DB-PIM to process 16 filters in parallel. In contrast, the acceleration in VGG19 is slightly lower because the ϕ^{th} for most of its Conv layers is set to 2, while for the FC layers, it remains at 1. Despite the FC layer having more parameters, their computational complexity is significantly less than that of the Conv layers. When considering input sparsity, the speedup for these two models increases to about 7.69× and 6.10×, respectively.

For compact NN models, such as MobileNetV2 and EfficientNetB0, we still achieve a noteworthy acceleration of 3.90× and 3.55×, respectively. As detailed in [11], limiting sparsity to partial layers at 80% in EfficientNetB0 results in just a 1.85× acceleration. This demonstrates the exceptional effectiveness of our method, particularly in EfficientNetB0 with limited redundancy. As shown in Fig. 7(b), the energy saving is also improved by 63.49% to 83.43% within five classical NN models.

4.4 Area Breakdown of DB-PIM

Tab. 4 provides a detailed area breakdown analysis for DB-PIM. The DB-PIM can be divided into the digital PIM baseline and additional logic introduced by our techniques. The additional logic mainly includes extra DFFs and routing resources in PIM macro, extra post-processing units with CSD-based adder tree, and meta RFs, all customized for our co-design. In our work, the complementary states ($Q/\overline{Q}$) in the cross-coupled structure of 6T SRAM cell represent two individual bits (the Comp. Pattern block) for parallel computations. Hence, compared to the counterpart in [17],

Table 3: Detailed Comparisons with Related Works.

	[12]	[11]	[13]	[14]	[15]	This Work				
Technology (nm)	65	28	28	28	28	28				
Die Area (mm^2)	12	6.07	3.93	14.36	8.97	1.15				
Supply Voltage (V)	0.62~1	1	0.64~1.03	0.60~1	0.60~0.90	0.72~0.90				
Frequency (MHz)	25~100	500	20~320	85~275	125~216	500				
Power (mW)	18.60~84.10	1050	8.27~250.65	29.83~153.62	11.40~45.10	1.45~11.65				
SRAM Size (KB)	294	384	96	192	114	272				
PIM Size (KB)	8	128	144	128	128	8				
Number of PIM Macro	4	512	96	128	16	4				
Dataset	CIFAR10 /ImageNet	ImageNet	Enwik8	VQA	CIFAR10	CIFAR100				
Actual Utilization ($\mathcal{U}_{act}$)	ResNet18	ResNet-50	Adaptive-Span Transformer	ViLBERT-base	ResNet20	AlexNet	VGG19	ResNet18	MobileNetV2	EfficientNetB0
	32.04%	**48.64%**	/	/	**<50%**	**91.95%**	**97.69%**	**98.42%**	**97.82%**	**94.41%**
Peak Throughput (TOPS) (8b/8b)	**0.10**	**26.21**	**3.33**	**3.55**	**0.40**	**0.31**				
Peak Throughput/Macro (GOPS) (8b/8b)	**24.69**	**51.19**	**34.68**	**27.73**	**25.1**	**77.5**				
Energy Efficiency (TOPS/W) (8b/8b)	**0.09~2.37**	**25 (dense)~107.60**	**1.96~25.22**	**48.40~101**	**5.99~13.75**	**18.14~45.20**				
Peak Energy Efficiency per Unit Area (TOPS/W/mm^2) (8b/8b)	**2.97**	**17.73**	**6.42**	**7.03**	**1.53**	**39.30**				

Table 4: DB-PIM Area Breakdown Analysis.

Modules	Area (mm^2)	Breakdown
PIM Baseline	1.00809	87.32%
Meta-RFs	0.07829	6.78%
Extra Post-processing Units	0.06259	5.42%
DFFs and Routing Resources	0.00550	0.48%
Input Sparsity Support	0.00007	~0.00%
Total	1.15453	100%

DB-PIM requires extra storage units (DFFs) and routing resources for processing the additional information. The overhead incurred by these units is relatively minor, approximately 0.48%. The majority of the extra overhead stems from the additional post-processing units and meta RFs, a trade-off for enhanced parallel processing capabilities. Typically, in standard configurations, a macro processing two 8-bit precision filters simultaneously requires only two post-processing units. However, our design is tailored for high-level parallel processing, allowing the concurrent computation of up to 16 filters based on the provided metadata. This necessitates one post-processing unit for each filter, leading to an overhead that scales with the number of filters processed in parallel. Although this approach results in a slight increase in area consumption, it significantly enhances the parallelism of the PIM array, thereby accelerating computation.

4.5 Comparison with Prior Works

Tab. 3 provides a comprehensive comparison with existing state-of-the-art (SOTA) PIM-based accelerators. These studies represent the two predominant sparsity approaches in SRAM-PIM architectures: value-level sparsity [11–13] and bit-level sparsity [14, 15]. We mainly focus on four critical aspects: utilization, peak throughput per macro, energy efficiency, and energy efficiency per unit area. The actual utilization $\mathcal{U}_{act}$ as illustrated in Eq. (1) in most of these studies is equal to the ratio of non-zero bits across various NN models. Fig. 2 shows that this ratio is extremely low for most NN models, indicating that the actual utilization in most studies is below 50%. In contrast, DB-PIM effectively tackles the challenge of low utilization by employing CSD encoding, FTA algorithm, and a cross-coupled structure to store two individual bits. Compared with previous works, $\mathcal{U}_{act}$ in DB-PIM improves up to about 3×. This innovative co-design boosts not only storage efficiency but also significantly enhances the peak throughput of each macro and energy efficiency. The peak throughput of each macro in DB-PIM gains up to 3.14× improvements compared to other works. Meanwhile, we compare the energy efficiency and energy efficiency per unit area with previous works. The results show that DB-PIM reaches up to 45.20 TOPS/W in system-level energy efficiency, surpassing most of the earlier works. It also shows that DB-PIM achieves the highest energy efficiency per unit area at 39.30, much higher than previous works.

5 Conclusion

Bit-level sparsity holds significant potential for enhancing computational efficiency. However, traditional digital SRAM-PIM struggles to effectively leverage randomly distributed bit-level sparsity due to their rigid crossbar structure. Our paper presents a breakthrough in this area through an algorithm-architecture co-design framework. DB-PIM framework adeptly utilizes unstructured bit-level sparsity in digital SRAM-PIM, overcoming the limitations of conventional designs. The results show that DB-PIM achieves a remarkable 5.20× speedup and a 74.47% improvement in energy saving by utilizing unstructured weight bit sparsity. Moreover, when combined with input bit sparsity, our framework attains even more remarkable results, with a 7.69× speedup and a 83.43% increase in energy efficiency. These results validate our co-design approach and highlight its potential to utilize the bit-level sparsity. In the future, we aim to combine our approach with value-level sparsity to maximize the exploitation of sparsity in NN models.

References

[1] Alex Krizhevsky et al. Imagenet Classification with Deep Convolutional Neural Networks. In *Proceedings of the NIPS*, 2012.

[2] Yu Zhang, William Chan, and Navdeep Jaitly. Very Deep Convolutional Networks for End-to-End Speech Recognition. In *Proceedings of the ICASSP*, 2017.

[3] Ekim Yurtsever, Jacob Lambert, et al. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE ACCESS*, 2020.

[4] Yu-Der Chih, Po-Hao Lee, et al. An 89TOPS/W and 16.3 TOPS/mm^2 All-Digital SRAM-based Full-Precision Compute-in Memory Macro in 22nm for Machine-Learning Edge Applications. In *Proceedings of the ISSCC*, 2021.

[5] Cenlin Duan, Jianlei Yang, et al. DDC-PIM: Efficient Algorithm/Architecture Co-Design for Doubling Data Capacity of SRAM-based Processing-in-Memory. *IEEE TCAD*, 2024.

[6] Tzu-Hsien Yang et al. Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks. In *Proceedings of the ISCA*, 2019.

[7] Fangxin Liu, Wenbo Zhao, et al. Bit-Transformer: Transforming Bit-Level Sparsity into Higher Preformance in ReRAM-based Accelerator. In *Proceedings of the ICCAD*, 2021.

[8] Xueyan Wang, Jianlei Yang, et al. TCIM: Triangle Counting Acceleration with Processing-in-MRAM Architecture. In *Proceedings of the DAC*, 2020.

[9] Xuhang Chen et al. Accelerating Graph-Connected Component Computation with Emerging Processing-in-Memory Architecture. *IEEE TCAD*, 2022.

[10] Yinglin Zhao, Jianlei Yang, et al. NAND-SPIN-Based Processing-in-MRAM Architecture for Convolutional Neural Network Acceleration. *SCIS*, 2023.

[11] Fengbin Tu, Yiqi Wang, et al. SDP: Co-Designing Algorithm, Dataflow, and Architecture for In-SRAM Sparse NN Acceleration. *IEEE TCAD*, 2022.

[12] Jinshan Yue, Xiaoyu Feng, et al. A 2.75-to-75.9 TOPS/W Computing-in-Memory NN Processor Supporting Set-Associate Block-Wise Zero Skipping and Ping-Pong CIM with Simultaneous Computation and Weight Updating. In *Proceedings of the ISSCC*, 2021.

[13] Shiwei Liu, Peizhe Li, et al. A 28nm 53.8 TOPS/W 8b Sparse Transformer Accelerator with In-Memory Butterfly Zero Skipper for Unstructured-Pruned NN and CIM-based Local-Attention-Reusable Engine. In *Proceedings of the ISSCC*, 2023.

[14] Fengbin Tu, Zihan Wu, et al. MuITCIM: A 28nm 2.24 uJ/token Attention-Token-Bit Hybrid Sparse Digital CIM-based Accelerator for Multimodal Transformers. In *Proceedings of the ISSCC*, 2023.

[15] Ruiqi Guo, Zhiheng Yue, et al. TT@ CIM: A Tensor-Train in-Memory-Computing Processor Using Bit-Level-Sparsity Optimization and Variable Precision Quantization. *IEEE JSSC*, 2022.

[16] Sampatrao L Pinjare et al. Implementation of Artificial Neural Network Architecture for Image Compression Using CSD Multiplier. In *Proceedings of the ERCICA*, 2013.

[17] Bonan Yan, Jeng-Long Hsu, et al. A 1.041-Mb/mm^2 27.38-TOPS/W Signed-INT8 Dynamic-Logic-based ADC-Less SRAM Compute-in-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications. In *Proceedings of the ISSCC*, 2022.

[18] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] Kaiming He, Xiangyu Zhang, et al. Deep Residual Learning for Image Recognition. In *Proceedings of the CVPR*, 2016.

[20] Mark Sandler, Andrew Howard, et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the CVPR*, 2018.

[21] Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the ICML*, 2019.