Dyn-Bitpool: A Two-sided Sparse CIM Accelerator Featuring a Balanced Workload Scheme and High CIM Macro Utilization

Xujiang Xiang*, Zhiheng Yue*, Yuxuan Li, Liuxin Lv,Shaojun Wei, Yang Hu, Shouyi Yin
School of Integrated Circuits, Tsinghua University, Beijing, China

ABSTRACT

Computing-in-memory (CIM), a promising computing paradigm, has demonstrated great energy-efficiency by integrating computing units into memory. However, previous research on CIM has rarely utilized sparsity in activation and weight concurrently. Moreover, new challenges arise when harnessing sparsity in both activation and weight (two-sided sparsity), such as imbalanced workload and low hardware substrate utilization.

To fully unleash the acceleration potential brought by two-sided sparsity, we implemented an accelerator called Dyn-Bitpool which innovates on two fronts: 1) a balanced working scheme called "pool first and cross lane sharing" to maximize the available performance benefiting from bit-level sparsity in activation; 2) dynamic topology of CIM arrays to effectively handle low CIM macro utilization issue stemming from value-level sparsity in weight. All the contributions collaborate to speed up Dyn-Bitpool by 1.89x and 2.64x on average on Googlenet, Mobilenetv3, Resnet50, ResNeXt101 and VGG19, compared with two state-of-the-art accelerators featuring CIM.

1 INTRODUCTION

Today's computing systems are built up from von Neumann architecture where data must be shuttled back and forth between memory and processing units. However, data movement is much more expansive than computation itself in terms of power consumption[1]. In data-intensive applications like deep learning, the von Neumann architecture faces a rough ride due to cooling constraints as well as the proliferation of mobile devices. So, computing-in-memory(CIM) becomes a promising alternative by performing computational tasks in place in the memory itself.

Normally, accelerators featuring CIM enable memory with certain computing capabilities and adopt bit-serial computing(BSC) style, where activation is serialized to participate in computing. Consequently, when the data precision is N bits, it should take N cycles to compute a group of data in parallel. Following the philosophy "Run,don't walk", an intuitive idea that exploits the bit-level sparsity to reduce computing latency comes to mind. However, just as Fig. 1 a) shows, there exists a troublesome problem originating from the imbalanced workload between different lanes. The lanes with light workload will be "locked" by these which have heavy workload("locked" means wait). It is a critical bottleneck when exploiting the bit-level sparsity in activation. In essence, the reason for it is that physical lanes and logical lanes are mapped statically, which is usually a one-to-one relationship. Here, a physical lane refers to the lane occupied in the physical space of a circuit, while a logical lane is a lane that performs effectual calculations logically defined by dataflow. For example, as shown in Fig. 1 a), A1xW1 is a logical lane which is statically mapped to physical lane1.

Apart from activation, weight is another operand involved in computing, which is usually stored in memory. Furthermore, Bit-Tactical[2] finds that value-level sparsity of weight is orthogonal to bit-level sparsity of activation. However, CIM macro's rigid structure is at odds with the randomly distributed zeros. So actually, some previous works such as [3][4][5][6], have adopted a new hardware-friendly sparsity pattern which features being structured at a fine granularity. Though the distribution of non-zero weight is restricted, the number of non-zero weight is not, which is usually below the predefined PE_num or the capacity of CIM macros. Because we tend to design large PE arrays or CIM macros to enhance throughput. As a result, the low hardware utilization problem may offset the speedup brought by sparsity, as stated in Fig. 1 b).

To overcome these challenges, this work presents a two-sided sparse CIM accelerator called Dyn-Bitpool. The main contributions include the following:

- We profile the bit-level sparsity in several popular neural networks and introduce **the first formal theoretical analysis** of the *load imbalance issue* and *workload sharing scheme*.
- A balanced workload scheme called ***pool first and cross lane sharing*** is proposed to approach optimal performance when exploiting bit-level sparsity in activation by regarding physical lanes as a pool and then remapping physical and logical lanes adaptively.
- ***Dynamic topology of CIM arrays*** is adopted to maximize hardware utilization by dynamically adjusting the allocation of parallelism when applying fine-grained structured pruning to weight.
- We perform extensive experiments on Dyn-Bitpool, which include sensitivity analysis and performance improvement breakdown, as well as extracting hardware characteristics from **layout and post-simulation**.

We evaluate the proposed Dyn-Bitpool with a dedicated reference model and implement it using Verilog and customized hand-built CIM macros. The result demonstrates that our proposed two-sided

*These authors contributed equally to this work.This work was supported in part by the NSFC under Grant 62125403, and Grant 92164301;in part by the National Science and Technology Major Project for the New Generation of Artificial Intelligence under Grant 2022ZD0115200;in part by the National Key Research and Development Program under Grant 2021ZD0114400;Beijing S&T Project Z221100007722023;in part by the project funding for the 2022 Special Project on Industrial Foundation Reconstruction and High Quality Development of Manufacturing Industry CEIEC-2022-ZM02-0245;in part by the Beijing National Research Center for Information Science and Technology;and in part by the Beijing Advanced Innovation Center for Integrated Circuits.Corresponding author: Shouyi Yin(yinsy@tsinghua.edu.cn).

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3655690

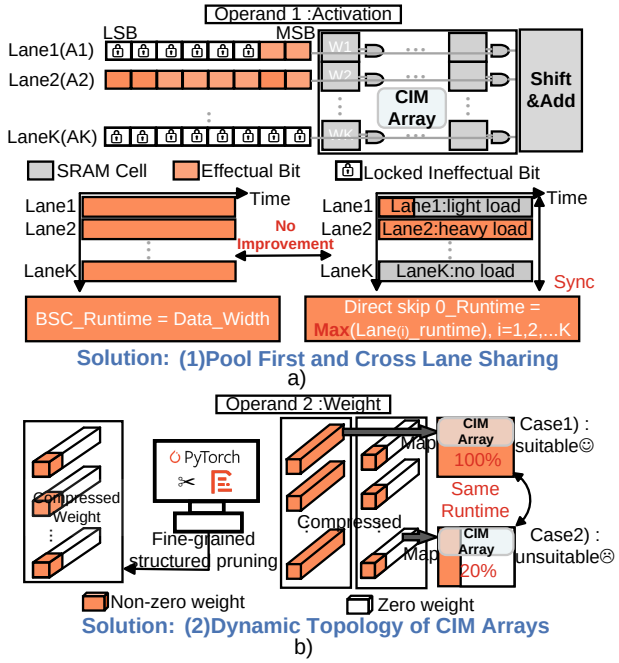


Figure 1: a) Load imbalance issue when leveraging bit-level sparsity in activation.b) Low hardware utilization problem when exploiting sparsity in weight.

sparse CIM accelerator achieves 1.89x and 2.64x speedup compared with two SOTA CIM accelerators.

2 BACKGROUND AND MOTIVATION

2.1 Bit-level Sparsity in Activation

In the realm of CIM, BSC dominates the computing paradigm. So, it will take N cycles to perform N-bit multiplication for CIM, e.g.16 cycles for INT16. But it only takes one cycle for conventional 16-bit parallel multiplier under the same frequency constraint. Some researchers have observed this "long-tail" issue and proposed their solutions. ReDCIM[7] adopts booth algorithm based multiplication which halves the computing latency. Sticker-IM[6] harnesses the bit-level sparsity in activation, since bit 0 has no contribution to the final result. Nevertheless, Sticker-IM only skips the computation when all incoming bits of activation are zero. Both works don't fully utilize the bit-level sparsity.

To figure out the acceleration potential of bit-level sparsity, we profile bit-level sparsity of activation in five popular models. As Fig. 2 a) exhibits, among these five models, the average bit-level sparsity can reach up to 0.8 and 0.91 at INT8 and INT16 respectively, which is much higher than its counterpart, i.e.value-level sparsity of activation. We point out the rationales for such high bit-level sparsity as follows: 1)activation functions(e.g.ReLU,GELU and h-swish) will convert negative activation to zero or near zero; 2)the distribution of activation is generally clustered, especially near-zero point, as shown in Fig. 2 b), thus a quantity of bit 0 will be introduced after quantization. Based on the above discussion, it would be such a waste not to fully exploit this opportunity for performance improvement. However, directly skipping bit 0 will introduce a load imbalance issue shown in Fig. 1 a), since the lane with the heaviest workload always rules the final runtime.To solve this load imbalance issue, we propose a highly efficient scheme which will be clarified in Sec. 3.

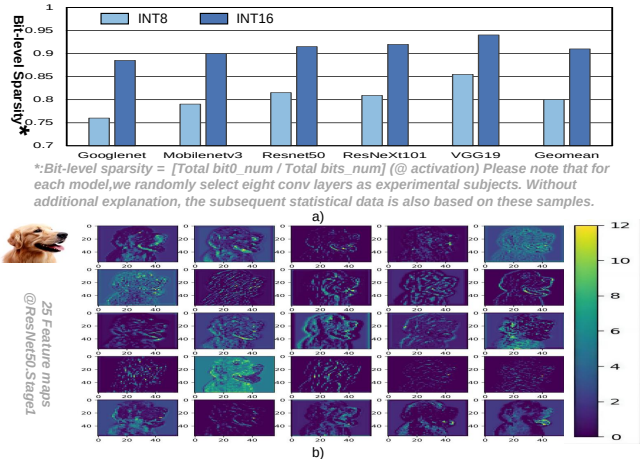


Figure 2: a)Bit-level sparsity in five widely used models.b)The data distribution of activation.

2.2 Value-level Sparsity in Weight

Due to the redundancy in neural networks, pruning has been proven as an effective method to compress models by pruning weight which is not sensitive to the performance[8]. However, fine-grained pruning algorithm features randomly distributed non-zero weight, which mismatches with the rigid structure of CIM macros. As a result, computation can't be skipped even though there are plentiful zeros. In addition, though coarse-grained pruning algorithms like pruning weight of one whole output channel[9] can enable CIM arrays to skip zeros, they have a very limited compression ratio.

Thus, SRE[4] proposed a fine-grained OU-based(structured) pruning to make 0 skipped and also enjoy a high compression ratio. Similar methods(i.e.fine-grained structured sparsity) are also adopted in [3][5][6]. For example, as shown in Fig. 3, we prune or reserve weight sharing the same (x,y,c) within *Kernel_stride* kernels.Among different *Kernel_stride* kernels, there is no constraint.

Despite the relative regular zero distribution in weight, it still naturally incurs a problem——low hardware utilization, since the exact number of non-zero weight is not constrained, which is less clearly clarified in prior works. For example, the layers with shallow channels may contain only 6 non-zero weight along the channel direction after pruning, which is far less than the capacity of CIM macros(i.e. predefined computing parallelism). It will offset the benefits brought by sparsity if we just pad zeros and store them in CIM macros, as illustrated in Fig. 1 b). From the heatmap in Fig. 4 a), we find that when we statically cap the predefined parallelism at 32, almost half of the layers are processed with the macro utilization below 45% after pruning. Thus, we propose "dynamic topology of CIM arrays" to fix this problem. Details are elaborated in Sec. 4.

3 POOL FIRST AND CROSS LANE SHARING

In this section, we first conduct modeling analysis on the *load imbalance issue* and *workload sharing scheme* from the aspect of

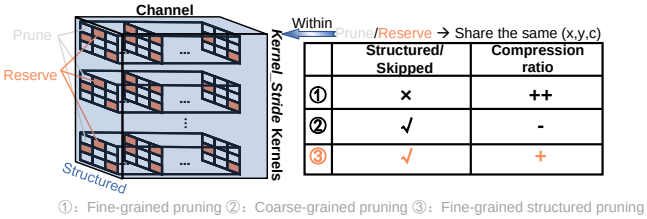


Figure 3: Fine-grained structured sparsity.

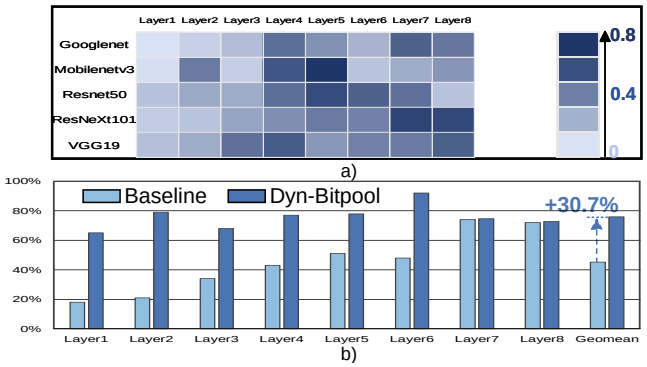


Figure 4: a) The heatmap of CIM macro utilization. b) Hardware utilization improvement by adopting dynamic topology of CIM arrays.

Probability and Statistics. Then, we will illustrate the hardware implementation of our proposed "pool first and cross lane sharing".

3.1 Theoretical Analysis

Load Imbalance Issue: we conduct modeling on this issue with two versions——**1)skip bit 0,2)skip code 0 after booth encoding**, which are denoted as ***Sp*** and ***Spbooth*** respectively.

For brevity, we assume that the data precision is N bits, the *group size* is K, the bit-level sparsity is 0.5 and the max runtime cycles are M for processing a group of activation in parallel. Inspired by the definition of driving capability, we only care the probability of processing a group of data with the constraint of using less than or equal to M cycles, which is denoted as probability @ group-level or a single letter P. The higher the probability, the higher the potential acceleration ratio. Here, for better understanding, we unify the terminology that effectual term(term={bit,code}) is the one that actually contributes to the result and ineffectual term has no contribution.

First, at value-level, P(density=i) is denoted as the probability of containing i effectual terms. Basically, the derivation process is a simple permutation and combination problem. It's noteworthy that only 000 and 111 can be encoded as 0 according to booth algorithm.

$$\begin{aligned} Sp: P(density=i) &= C_N^i (0.5)^{N-i}(0.5)^i, (i=0->N) \\ Spbooth: P(density=i) &= C_{N/2}^i (0.25)^{N/2-i}(0.75)^i, (i=0->N/2) \end{aligned} \tag{1}$$

Then, at group-level, owing to the load imbalance issue, every single data should contain M effectual terms at most. Thus, the final probability should be multiplied by K times.

$$\begin{aligned} Sp: P &= (\sum_{i=0}^{min(M,N)} P(density)=i)^{\mathbf{K}} \\ Spbooth: P &= (\sum_{i=0}^{min(M,N/2)} P(density=i))^{\mathbf{K}} \end{aligned} \tag{2}$$

From Eq. 2, we can clearly find that the bottleneck lies in $x^{\mathbf{K}}$. When K increases, the probability @ group-level will severely decrease.

Workload Sharing Scheme: If we can share the workload across lanes (i.e.effectual terms can be computed in any lanes), as a consequence, the probability @ group-level circumvents $x^{\mathbf{K}}$. Instead, the final probability is

$$P = P(\frac{1}{K}\sum_{j=1}^{K} X_j \le M) \tag{3}$$

Here, variable X_j is j^{th} activation(lane) containing i effectual terms. However, we can not derive the precise value of Eq. 3, since variable X_j is a random variable. Consequently, it seems that we cannot quantitatively compare the probability @ group-level between two cases. However, if K approaches ∞, *Central Limit Theorem* will take effect, which means that we can derive the approximate value of Eq. 3 via *normal function*. A variant of the *Central Limit Theorem* lies here: if $X_j (j \in [1, K])$ are independent and identically distributed and K approaches ∞, we have

$$\frac{\frac{1}{K}\sum_{j=1}^{K} X_j - \mu}{\sigma/\sqrt{K}} \sim N(0,1),\ N(0,1)\ is\ standard\ normal\ function \tag{4}$$

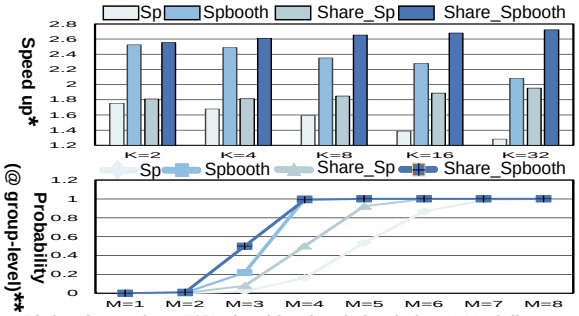
So, we must firstly derive the expectation μ and variance σ^2 of X_j to get the approximate value of Eq. 3. We notice that Eq. 1 satisfies the *Binomial distribution*, thus we can very easily obtain the expectation and variance of X_j. Then, substituting μ and σ^2 of X_j into Eq. 4 and then substituting Eq. 4 into Eq. 3, we have

$$\begin{aligned} Share\_Sp: P &= P(\frac{1}{K}\sum_{j=1}^{K} X_j \le M) \approx P(N(0,1) \le \frac{M-\mu}{\sigma/\sqrt{K}}) \\ &= P(N(0,1) \le \frac{2M-N}{\sqrt{N/K}}),\ similarly \\ Share\_Spbooth: P &\approx P(N(0,1) \le \frac{8M-3N}{\sqrt{6N/K}}) \end{aligned} \tag{5}$$

Here, the prefix "Share" means after sharing workload. To visualize the probability value in different scenarios, we record the probability @ group-level varying along with M according to Eq. 5 and Eq. 2. In addition, we apply different *group size* K to these four cases and compare the runtime with the conventional BSC equipped with the same *group size*. Both results are shown in Fig. 5.

From Fig. 5, we find that the theoretical analysis and experimental results are in good agreement. In addition, we can clearly find that as the *group size* K increases, the speedup of *Sp* and *Spbooth* declines, **whereas the opposite is true for *Share_Sp* and**

Share_Spbooth, which manifests the superiority of workload sharing scheme. In the next section, we'll talk about the hardware implementation of workload sharing scheme.



*:Bit-level sparsity = 50%,algorithm-level simulation **.Modeling results:N=8 and K=4.(When K is large, the probability value is too small to distinguish.Similar conclusions can be drew when K becomes larger.)

Figure 5: Average speedup compared with conventional BSC (top).Probability @ group-level varies along with parameter M(bottom).

3.2 Hardware Design

We develop our proposed workload sharing scheme with two versions corresponding to ***Share_Sp*** and ***Share_Spbooth*** respectively. Fig. 6 elucidates this scheme of *Share_Spbooth*. It is worth noting that the hardware of *Share_Spbooth* is a superset of *Share_Sp*, implying it is straightforward to deduce the corresponding circuit of *Share_Sp* from Fig. 6. For example, booth encoders in stage ① is not existent for *Share_Sp*. After stage ①, every term contains four fields —— Sign, Valid, SF and ID. Notably, Valid field indicates whether the term itself is effectual. Basically, stage ② consists of *sub_group_size* trailing zero counters(TZC) in serial connection. Here, we must emphasize that we only conduct cross-lane sharing within the scope of a sub_group for the consideration of the trade off between performance and hardware overhead. Additionally, there may be multiple parallel and independent sub_groups inside a group. TZCs detect non-zero bits in the Valid vector and return their positions which will be used to clear the detected non-zero bit and retrieve information from the same position in the remaining three fields. Stage ③ is cross-lane sharing computation where different fields act on corresponding components, e.g the id field acts as select signals to MUX for the right match between activation and weight. After stage ③, partial products are generated, which should be further added to obtain the final result.

The crux of this solution is a strategic remapping of physical and logical lanes. Here, physical lanes are considered as a resource pool. This allows us to dynamically allocate a specific number of physical lanes to logical lanes, based on the workload distribution. The ID field maintains this mapping relationship, while the early termination signal indicates the release of physical resources. For example, as shown in stage ③, there are two logical lanes and four physical lanes at cycle0. Since A2xW2 and A3xW3 is invalid logically, physical lane 1 and 2 are mapped as logical lane 1(A1xW1). Likewise, physical lane 3 and 4 are mapped as logical lane 2(A4xW4). As a result, computing latency is reduced by 4x. In Sec. 6.3, we will comprehensively validate the efficacy of this scheme and its impact on hardware when altering *sub_group_size*.

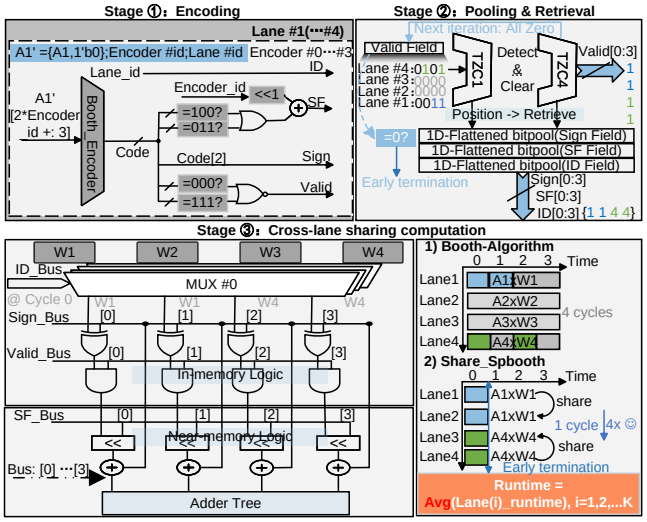


Figure 6: Illustration of "pool first and cross lane sharing" scheme of *Share_Spbooth*.Here, N=8, K=sub_group_size=4.

4 DYNAMIC TOPOLOGY OF CIM ARRAYS

In this section, we introduce the scheme of "dynamic topology of CIM arrays", which is guided by a crucial insight——***The total computing parallelism of hardware is static but we can alter the allocation of parallelism during runtime***. Besides, there are two operands sharing the total parallelism, i.e.activation and weight.

The first step is to let hardware know the exact number of incoming weight(i.e. needed weight_parallelism). Firstly, as shown in Fig .7 a), the host will inform Dyn-Bitpool the tail address of the weight of any (x,y) coordinate by configuring relevant registers, which is the most indispensable step. Then, Dyn-Bitpool will initiate Runtime Table through simple parsing procedure. At last, as long as this round of computing is done, the controller will update and maintain the Runtime Table by firstly changing the head address, which is transparent to the host. The second step is topology transformation. For a CIM-core, incoming effectual_weight_num = tail_addr-head_addr+1, which is used to select the corresponding topology. Here, we assume that one CIM core consists of four CIM arrays, each of which can accommodate eight effectual weight(i.e.computing parallelism=8 @ one CIM array).

For a clear understanding, we establish a fixed scene as depicted in Fig. 7 b). Here, CIM arrays contributing to the same output activation form a cluster. As shown in Fig. 7 b), the incoming effectual_weight_num is 6, indicating that the needed weight_parallelism (6) is well below the total parallelism(32). This leaves a substantial amount of spare parallelism that can be utilized by the input activation. In this case, one CIM array constitutes one cluster and weight is broadcasted to four CIM arrays. Thus, IN_Parallelism is quadruplicated, so does CIM macros' utilization. Likewise, the remaining two topology forms work in a similar way. Strictly speaking, the topology of CIM arrays remains a mesh at a coarse granularity, but the geometry constituting the mesh has changed, as illustrated in Fig. 7b) left. Finally, as shown in Fig. 4 b), CIM macros' utilization is increased by an average of 30.7% when adopting *dynamic topology of CIM arrays*, which is based on ResNeXt101.

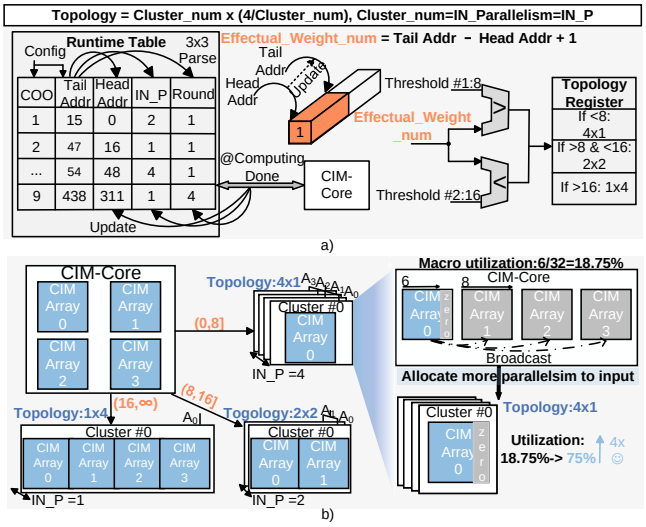


Figure 7: A general case a) and particular case b) of "dynamic topology of CIM arrays".

5 IMPLEMENTATION

Fig. 8 provides the top view of Dyn-Bitpool. We have implemented Dyn-Bitpool using 28nm CMOS technology, equipped with a 64KB global input buffer, a 144KB weight buffer, a 16KB output buffer, a 256b on-chip bus, a 1D-engine, a 2D-engine, a top controller, a scheduler and 16 CIM Cores. One CIM core encompasses four CIM arrays and other necessary modules mentioned above. We conduct cross lane sharing scheme within the scope of one CIM array, i.e.*sub_group_size*=8. The scheduler is utilized to balance workload at the inter-core level. Additionally, we choose run-length encoding to compress the weight for its relatively simple decoding procedure. As regards pruning, we opt *Kernel_stride* = 16 to preserve Quality-of-Service.

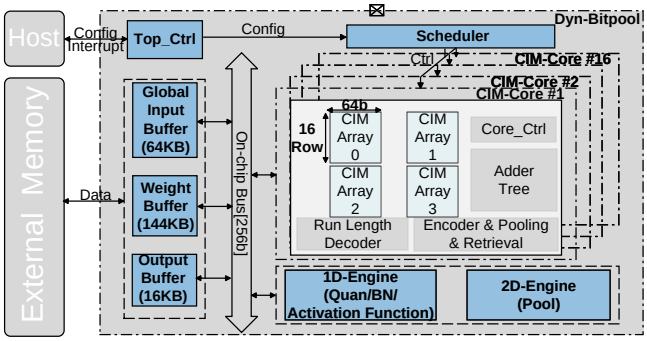


Figure 8: Top view of Dyn-Bitpool

6 EVALUATION

6.1 Measurement Setup

Firstly, we develop a dedicated reference model using MATLAB, which is comprehensively verified. In addition, we use Pytorch to train, prune and quantize five aforementioned models on the Imagenet-1K_V2 dataset. Subsequently, we employ VCS and PrimeSim to conduct digital/analog mixed signal simulation to verify the functionality of the circuits. Upon functional verification, the digital portion of Dyn-Bitpool is synthesized by Design Compiler for area and frequency estimation, while the corresponding information about CIM macro is derived from the layout. Finally, PrimeTime PX is used to measure power consumption of the digital portion based on model-level granularity, and the counterpart of CIM macro is derived from post-simulation in Virtuoso. In addition, digital buffers are synthesized by Memory compiler for better area-efficiency but the CIM macros are hand-built starting from the transistor level.

	Dyn-Bitpool*	Sticker-IM[6]**	ReDCIM[7]*
Technology node	28nm CMOS	65nm CMOS	28nm CMOS
Memory library	UHDSPSRAM-HVT	N/A	N/A
Frequency	500MHz	100MHz	220MHz
Data precision	INT8	INT2/4/6/8	INT8/16,BF16,FP32
Total digital SRAM	224KB	164KB	128KB
Macro capacity	64Kb	16Kb	96Kb
CIM type	Digital-CIM	Analog-CIM	Digital-CIM
Cell area	4.675um^2	N/A	N/A
Total area	2.924mm^2	9mm^2	6.69mm^2
Power/digital portion	69.2mW(@Resnet50) 65.6mW(@Mobilenetv3) 66.2mW(@ResNeXt101)	21.79mW	25.55mW
Power/CIMmacro	38.4~71.3mW***	34.95mW	43.8mW
Peak performance	3.5TOPS	0.2TOPS	1.35TOPS
Area effeciency	1.19TOPS/mm^2	0.022TOPS/mm^2	0.2TOPS/mm^2
Power effeciency	28.2TOPS/W****	3.56TOPS/W	19.5TOPS/W
Sparsity support	A&W	A&W*****	Not support

.@ peak performance A&W:8b. **: @ peak performance A:4b, W:8b . *:Due to the very slow post-simulation speed, we do not conduct power analysis based on model-level granularity. Instead, we choose to inject random numbers. ****: digital portion@ Resnet50, CIM macro @ (min+max)/2. *****: sparsity in activation: skip style; sparsity in weight: power gating style*

Table 1: Hardware characteristics of Dyn-Bitpool,Sticker-IM and ReDCIM.

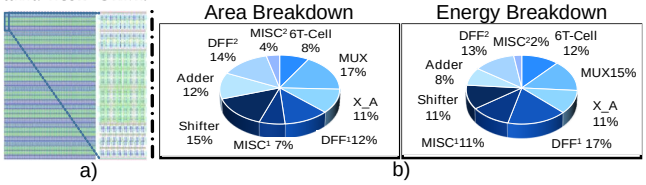


Figure 9: a) Layout image of one cim array(left) and eight cells(right) in it.b)Area and energy breakdown. X_A =XOR_AND gate. $XX^1 : in-memory \ \ XX^2 : near-memory$

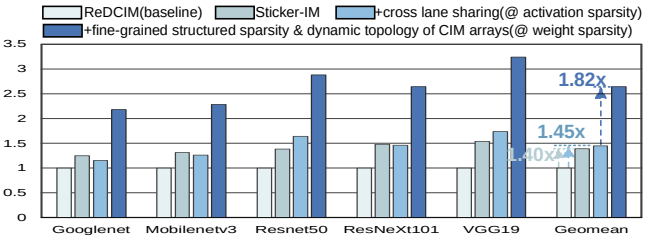
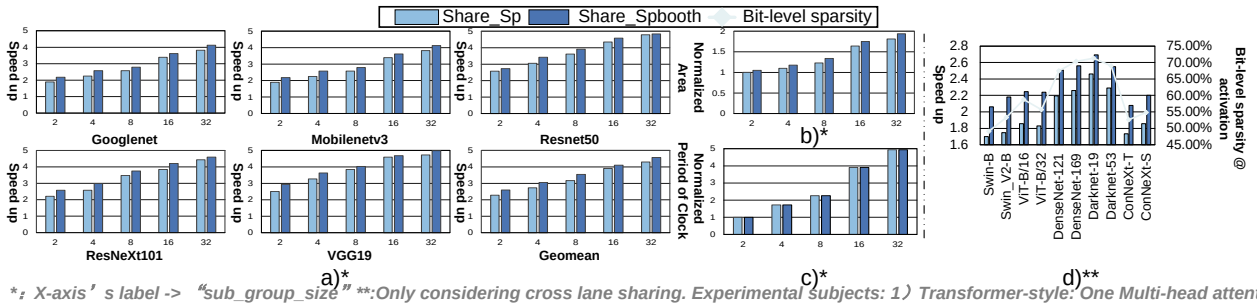


Figure 10: Performance comparison with ReDCIM and Sticker-IM @ CIM cores.

6.2 Hardware Summary

Table. 1 summarizes the hardware characteristics of Dyn-Bitpool. Fig. 9 presents the layout image of one cim array, along with its area & energy breakdown derived from post-simulation. For area-efficiency, in/near-memory logic primarily relies on transmission gate logic or pass transistor logic rather than static complementary CMOS logic. The area and energy breakdown reveal that DFF and MUX dominate the overhead. Fig. 10 provides a performance comparison between Dyn-Bitpool and two other SOTA CIM accelerators. We only compare performance because both SOTA CIM



*：X-axis’s label -> “sub_group_size”**:Only considering cross lane sharing. Experimental subjects: 1）Transformer-style: One Multi-head attention (Matrix-matrix multiplication,mat A x mat B, activation -> Mat A) ;2) Convolution-style: Two conv layers (Direct convolution). Dataset: ImageNet-1K. Framework:Pytorch. Sub_group_size = 8 Group_size=32.

Figure 11: Sensitivity analysis of *sub_group_size* to a)speedup over conventional BSC, b)area and c)frequency.Please note that *group size* is fixed to 32 and data precision is fixed to INT8. In addition, area and frequency analysis is based on one CIM core(see in Sec. 5). d)Speedup brought by cross lane sharing in dense applications.

accelerators have been silicon-validated, making it inappropriate to compare area and power consumption with them. But for quicker reference by other researchers, basic hardware characteristics of Sticker-IM and ReDCIM are also included in Table. 1. In addition, to guarantee sufficient fidelity in comparing performance, we develop a dedicated cycle-accurate model for each accelerator's core data path. The result demonstrates that *cross lane sharing* and *fine-grained structured sparsity & dynamic topology of CIM arrays* enhance performance by 1.45x and 1.82x respectively. It's important to note that although fine-grained structured sparsity is also adopted by Sticker-IM[6], zeros in weight are still stored in CIM macros and are only used to save power, rather than significantly improving performance like Dyn-Bitpool. All these contributions collectively enable Dyn-Bitpool to outperform Sticker-IM and ReDCIM by 1.89x and 2.64x, respectively.

6.3 Sensitivity Analysis

Sub_Group_Size: To evaluate the performance improvement brought by cross-lane sharing scheme and its impact on hardware, we derive the result from our reference model testing on the five NN models mentioned in Sec. 2.1 and collect data on the area and frequency fluctuating with *sub_group_size*. From Fig. 11 a), we can clearly find that the average runtime cycles for a group of data is approximately **1.98-2.87 cycles** when *sub_group_size* = 8 and the data precision is **INT8**. This indicates that with minor hardware overhead, cross-lane sharing scheme can achieve at most 2x reduced computing latency compared with booth algorithm based BSC. From Fig. 11 b) and c), we observe that as *sub_group_size* increases, CIM core's area grows almost linearly but frequency drops significantly. We can insert more DFFs to cut off the long combinational logic chain to fix this but the extra overhead is usually non-trivial except for scenarios demanding high performance. Meanwhile, from Fig. 11 a),b) and c), it is evident that *Share_Spbooth* outperforms *Share_Sp* to some extent but with a slight hardware penalty. This is why Dyn-Bitpool adopts *Share_Spbooth*.

Dense Application: Combining the results from Fig.2 a) and Fig. 11 a), we can find that bit-level sparsity affects the final speedup benefiting from cross lane sharing, which is rather intuitive. So, we choose some other models featuring relatively low bit-level sparsity in activation to check the efficacy of cross lane sharing.Results are shown in Fig. 11 d). For example, bit-level sparsity in Swin-B is roughly 49%, as a result, the speedup is around 2(i.e. similar performance benefit of booth algorithm based BSC). In other words, this also manifests one of the superiority of two-sided sparsity over one-sided sparsity, since when sparsity in one side does not work well(e.g. low sparsity), the other side may work well.

7 CONCLUSION

In this paper, we propose Dyn-Bitpool, a CIM accelerator leveraging two-sided sparsity——bit-level sparsity in activation and value-level sparsity in weight. Dyn-Bitpool employs a unique balanced workload scheme termed "pool first and cross lane sharing" to efficiently address the load imbalance issue, thereby fully unleashing the acceleration potential brought by bit-level sparsity in activation. Besides, to counter the low hardware utilization brought by value-level sparsity in weight, we propose an ingenious strategy called "dynamic topology of CIM arrays". Extensive experiments demonstrate that on average, Dyn-Bitpool achieves 1.89x and 2.64x speedup compared with two SOTA CIM accelerators respectively.

REFERENCES

[1] Mark Horowitz. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

[2] Alberto Delmas Lascorz et al. "Bit-Tactical: A Software/Hardware Approach to Exploiting Value and Bit Sparsity in Neural Networks". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 749–763. ISBN: 9781450362405. DOI: 10.1145/3297858.3304041. URL: https://doi.org/10.1145/3297858.3304041.

[3] Chaoqun Chu et al. "PIM-Prune: Fine-Grain DCNN Pruning for Crossbar-Based Process-In-Memory Architecture". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218523.

[4] Tzu-Hsien Yang et al. "Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks". In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 2019, pp. 236–249.

[5] Xuda Zhou et al. "Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach". In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, pp. 15–28. DOI: 10.1109/MICRO.2018.00011.

[6] Jinshan Yue et al. "STICKER-IM: A 65 nm Computing-in-Memory NN Processor Using Block-Wise Sparsity Optimization and Inter/Intra-Macro Data Reuse". In: *IEEE Journal of Solid-State Circuits* 57.8 (2022), pp. 2560–2573. DOI: 10.1109/JSSC.2022.3148273.

[7] Fengbin Tu et al. "ReDCIM: Reconfigurable Digital Computing- In -Memory Processor With Unified FP/INT Pipeline for Cloud AI Acceleration". In: *IEEE Journal of Solid-State Circuits* 58.1 (2023), pp. 243–255. DOI: 10.1109/JSSC.2022.3222059.

[8] Yu Cheng et al. "A survey of model compression and acceleration for deep neural networks". In: *arXiv preprint arXiv:1710.09282* (2017).

[9] Wei Wen et al. "Learning structured sparsity in deep neural networks". In: *Advances in neural information processing systems* 29 (2016).