Bit-balance: Model-Hardware Co-design for Accelerating NNs by Exploiting Bit-level Sparsity

Wenhao Sun, Zhiwei Zou, Deng Liu, Wendi Sun, Song Chen, *Member, IEEE,* and Yi Kang

Abstract—Bit-serial architectures can handle Neural Networks (NNs) with different weight precisions, achieving higher resource efficiency compared with bit-parallel architectures. Besides, the weights contain abundant zero bits owing to the fault tolerance of NNs, indicating that bit sparsity of NNs can be further exploited for performance improvement. However, the irregular proportion of zero bits in each weight causes imbalanced workloads in the Processing Element (PE) array, which degrades performance or induces overhead for sparse processing. Thus, this paper proposed a bit-sparsity quantization method to maintain the bit sparsity ratio of each weight to no more than a certain value for balancing workloads, with little accuracy loss. Then, we co-designed a sparse bit-serial architecture, called Bit-balance, to improve overall performance, supporting weight-bit sparsity and adaptive bitwidth computation. The whole design was implemented with 65nm technology at 1 GHz and performs at 326-, 30-, 56-, and 218-frame/s for AlexNet, VGG-16, ResNet-50, and GoogleNet respectively. Compared with sparse bit-serial accelerator, Bitlet, Bit-balance achieves $1.8\times{\sim}2.7\times$ energy efficiency (frame/J) and $2.1\times{\sim}3.7\times$ resource efficiency (frame/mm^2).

Index Terms—hardware accelerator, bit sparsity, quantization, neural network.

━━━━━━━━━━━━━━━━━━━━ ◆ ━━━━━━━━━━━━━━━━━━━━

1 INTRODUCTION

Nowadays, NNs have been widely applied in numerous domains, such as image recognition [1], [2], [3], [4], [5] speech recognition [6], object detection [5], and computer vision [7]. Structurally, it mainly consists of convolutional (CONV) layers and fully connected (FC) layers; the former conduct convolution, and the latter conduct matrix-vector multiplication (GEMV), both of which include massive multiplication-accumulation (MAC) operations. As higher precision and complication demands arise, the workloads of network computing continue to increase. Therefore, researchers have been striving for neural network hardware accelerators to catch up with software development.

Although NNs bring intensive computations, there exist monotonous operations; thus, many hardware architects attempt to reduce runtime by improving computation parallelism. The accelerators can be classified as four categories, bit-parallel, sparse bit-parallel, bit-serial, sparse bit-serial. DaNN [8], Eyeriss [9], TPU [10], Tianjic [11], and etc are typical bit-parallel accelerators, which primarily actualizes acceleration through improving parallelism with massive fixed-point computing units. Additionally, since the input feature maps (IFMs) and weights contain abundant zero elements owing to the fault tolerance of NNs, which can be exploited to accelerate NNs by skipping zero-elements computing. Therefore, lots of sparse bit-parallel accelerators emerge, such as EIE [12], Cambricon-X [13], Cnvlutin [14], SCNN [15], etc, further improving performance and energy efficiency further. However, these works primarily are designed for fixed-point computing, without consideration of varying weight/IFM-precision for different NNs, causing inefficient resource utilization and energy efficiency.

To adapt to varying precisions of different NNs, researchers attempt to design accelerators based on bit-serial computing. Stripe [16] presented a hardware accelerator that relied on bit-serial computing units and quantizes IFMs to the required precision, p, in each layer, which can provide $16/p\times$ speedup compared with 16-bit fixed-point baseline. Loom [17] reduced the number of both weights and IFMs bits for bit-serial acceleration. Bit-Fusion [18] proposed a bit-flexible accelerator to match the bitwidth of each individual multiply-add operand, maximizing the resource efficiency for operations of different bitwidths. UNPN [19] designed a Lookup table(LUT)-based bit-serial PE, achieving 23% 54% energy consumption reduction compared with conventional fix-point MAC units. These accelerators leverage bit-serial computing to improve performance and energy efficiency. However, they didn't exploit bit sparsity, indicating there are still room for performance improvement.

Owing to the bit sparsity of NNs, Some researchers designed sparse bit-serial accelerators to further improve performance. Pragmatic [20] skips the computing at the common zero-bit position of involved IFM elements. But the zero-bits distribution of each IFM bit sequence is irregular, resulting in only a small propotion of truncated zero-bits. Laconic [21] tears MAC operation into IFMs and weights bit-serial computing and accelerates by exploiting both IFMs and weights sparsity. However, the distribution of zero bit in IFMs and weights can be irregular, which causes imbalanced workload in the PE array and degrades performance. Bit-Tactical [22] applied a weight skipping module and an IFM-bit selection module to optimize the layout of irregular sparse weights through front-end

---

• *Wenhao Sun, Zhiwei Zou, Deng Liu, Wendi Sun, Song Chen, and Yi Kang are affiliated with the School of Microelectronics, University of Science and Technology of China, Hefei, Anhui, 30332 China.*
E-mail: wh1997@mail.ustc.edu.cn; songch@ustc.edu.cn

scheduling, thus further avoiding the calculation of zero weight elements and IFM bits. However, the independent supply of IFMs between PEs makes its data cache resources far greater than the area of computing resources, resulting in low resource utilization. Bitlet [23] proposes the bit interleaving philosophy to maximally exploit bit-level sparsity, which enforces acceleration by decreasing the number of weights involved in computing. However, the logic corresponding to bit interleaving occupies over 40% area of the entire PE module. Besides, the computing units of high precision can be idle during low precision operation, causing inefficient resource efficiency. Though these architectures exploit bit-sparsity for acceleration, they still suffer from imbalanced workload or huge overhead for implementation of sparse bit-serial processing.

This paper aims to balance sparse bit-serial workloads in the PE array while lowering the overhead of sparsity processing. A model-hardware co-design of the sparse bit-serial accelerator for NNs is proposed to improve system performance and energy efficiency. Our main contributions are as follows:

1). We proposed a systolic-array-based accelerator, called Bit-balance, supporting sparse bit-serial processing of weights and adaptive bitwidth computing, achieving high performance without extra preprocessing module by processing the sparse weights of the encoding format directly.

2). We co-designed a bit-sparsity quantization method to keep the sparsity ratio of each weight at a certain value with little accuracy loss, which can balance PE loads across the PE array and significantly improve performance.

Compared with bit-serial accelerator, Stripe [16], Bit-balance exploit weight bit sparsity for acceleration and achieved $4\times\sim7.1\times$ speedup and $3\times\sim5.6\times$ energy efficiency. Besides, Compared with sparse bit-serial accelerator, Laconic [21], we balanced the workload in the PE array and achieved $2.2\times\sim4.3\times$ speedup and $2.7\times\sim5.4\times$ energy efficiency. Further, compared with Bitlet [23], we lowered the overhead of sparse processing significantly, achieving $1.8\times\sim2.7\times$ energy efficiency and $2.1\times\sim3.8\times$ resource efficiency.

2 PRELIMINARY

Since NNs can be applied to various domains, the precisions of weights are diverse. Besides, weights of NNs contain many abundant zero bits due to the strong fault tolerance. Thus, there is great potential for architectures to gain more benefits based on bit-serial computing. Although bitwidth quantization can decrease the bit-serial computing cycles directly, only limited bitwidth can be cut for maintaining accuracy. Exploiting the bit sparsity of weights can further accelerate bit-serial computing, but the PE array suffers from imbalanced workloads. Those PE processing weights with a small number of non-zero bits (NNZB) must wait for those with large NNZB, which will degrade the performance and PE utilization. Therefore, the key challenge is to balance the sparse bit-serial computing workloads across the PE array.

2.1 Computing Process of NN

The computing process of NNs is shown in Fig.1. A CONV layer receives C_i input channels (ICs) of IFMs with a size of $H_i \times W_i$. Each OFM of C_o output channels (OCs), with a size of $H_o \times W_o$, is generated by sliding a $H_k \times W_k$ kernel window on one IFM and then accumulating across the temporary results of C_i ICs. Finally, C_o OFMs are calculated by convolving C_o groups of kernels with shared IFMs. For FC layers, the size of IFMs and OFMs are both 1×1. Set IFMs as matrix $I[C_i, H_i, W_i]$, weights as $W[C_o, C_i, H_k, W_k]$ and OFMs as $O[C_o, H_o, W_o]$, and the process can be described as Equ.1.

$$O[o,x,y] = \sum_{i=0}^{C_i}\sum_{a=0}^{H_k}\sum_{b=0}^{H_k} I[i, x+a, y+b] \times W[o, i, a, b] \tag{1}$$
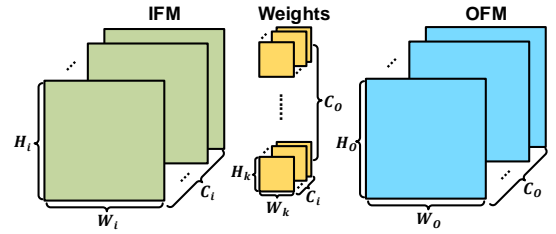$$(0 \le o < C_o, 0 \le x < H_o, 0 \le y < W_o)$$

Fig. 1: Convolution process.

Referring to Equ.1, the basic operations of NN processing are MACs. Assuming the bitwidth of weights is N, the MAC of an IFM, I_0, and a weight, W_0, can be decomposed to shift and add operations, as shown in Equ.2, which is flexible for varying bitwidth.

$$I_0 \times W_0 = \sum_{i=0}^{N} I_0 \times W_0[i] \times 2^i \tag{2}$$

2.2 Challenges of Sparse bit-serial computing

To accelerate bit-serial processing, quantizing the weight to lower bitwidth is the most straightforward method. If the weight is quantized from N-bit to N_{pb}-bit, we can achieve $N/N_{pb}\times$ speedup. An example of bit-serial computing with quantized weight is shown in Fig.2, achieving $1.3\times$ performance improvement. However, assuming the value of N-bit weight is 2^N, it can be decreased to $2^{N_{pb}}$ when the bitwidth is tailored to N_{pb}. To maintain the accuracy, the reduction of bitwidth can be limited, which constrains the performance improvement.

To further improve performance, the bit sparsity of weights can be exploited for acceleration by skipping zero-bits computing. Assuming the NNZB in each weight is N_{nzb} with the largest value being N_{nzb_max}, weights are compressed and loaded in PE array, with N_{nzb_max} computing cycles. An example of bit-serial computing by exploiting sparsity is shown in Fig.3(a), achieving $1.6\times$ performance. However, since the sparsity ratio of weights is randomly distributed, the workloads across the PE array is imbalanced. The PEs with small NNZB must wait for those with large NNZB to finish. Thus, PE_1 will be idle for 2

cycles, indicating inefficient PE utilization. Moreover, once the N_{nzb_max} is equal to original weight bitwidth, there can be no performance improvement.
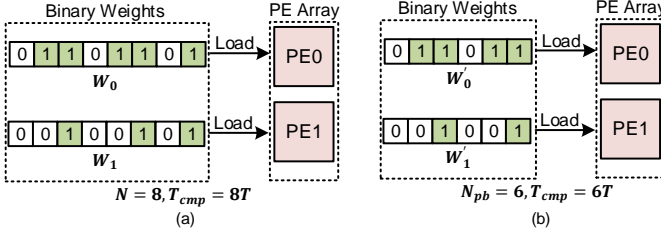

Fig. 2: Bit-serial computing with original weights and quantized weights in the PE array. (a)Original weights. (b)Quantized weights.
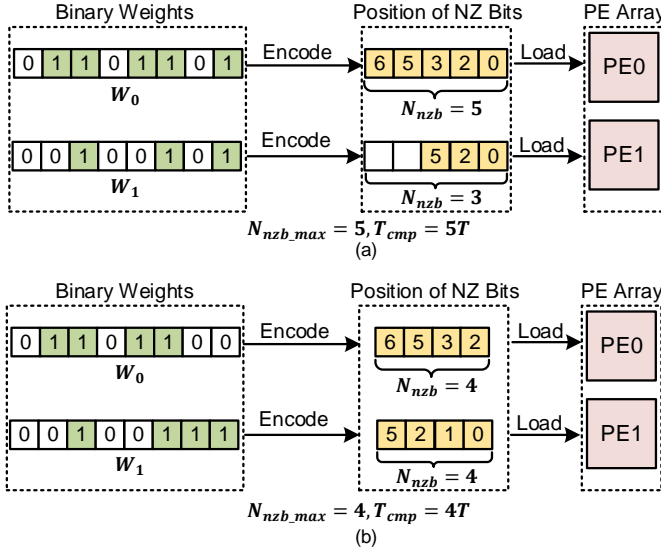

Fig. 3: Sparse bit-serial computing with imbalanced and balanced workloads of non-zero (NZ) weight bits in the PE array. (a)Imbalanced workloads. (b)Balanced workloads.

Although there are some previous works of bit-serial computing acceleration, such as Stripe [16], Laconic [21] and Bitlet [23], they all suffer from performance degradation or inefficient resource efficiency owing to imbalanced workloads. Thus, in this work, we aim to balance the workloads of sparse weight bits in the PE array.

3 METHODOLOGY

To deal with the imbalanced workloads in the PE array, we quantize the weights based on bit sparsity and maintain the NNZB in each weight to no more than a certain value, achieving significant improvement of PE utilization and performance. To fit with bit-serial computing, we store the weights in encoding format, which can be used directly for shift-add operation.

3.1 Bit-Sparsity Quantization

Since NNs have the nature of strong fault tolerance, the weights allow redundant bits. However, quantizing weights to lower bitwidths directly will significantly decrease the

numeric range of weight value and exploiting the randomly distributed bit sparsity of weights will suffer from imbalance workloads. Thus, to maintain the numeric range of weight value and balance computing loads across the PE array, we proposed a bit-sparsity quantization method, which set several weight bits as zero and maintain the maximum NNZB, N_{nzb_max}, in each weight to no more than a certain value. Assuming the bitwidth of original weight is N, its numeric range can be calculated as $\sum_{i=0}^{N_{nzb\_max}} \binom{N}{i}$. Compared with quantizing the weights to N_{pb}-bit directly, the numeric range of weight value in our method is more abundant, as shown in Tab.1. The case of $N_{nzb\_max} = 3$ in bit-sparsity quantization can be competitive with $N_{pb} = 9$ in bitwidth quantization.

Compared with exploiting the randomly distributed bit sparsity of weights, the performance based on our quantization method is higher owing to the more balanced workload of sparse weight bits across the PE array. An example of the computing flows toward bit-sparsity quantization is shown in Fig.2(b). The NNZB in W_0 and W_1 are both four, eliminating the idle PEs. Therefore, we obtained the total computing time of $4T_w$ with $1.25\times$ speedup compared with the imbalanced workload case.

The specific flow of bit-sparsity quantization is shown in Fig.4. First, we set an initial maximum NNZB of weights, N_{nzb_max}. Then, for weights with NNZB larger than N_{nzb_max}, we set the less significant none-zero bits as zero and maintain the total NNZB to N_{nzb_max}. Next, the previously quantized weights were retrained and testified if the accuracy dropped out of boundary. If not, we continued to decrease N_{nzb_max} and train the weights; otherwise, we saved the final quantized weights and recorded N_{nzb_max}. Fig.5 shows a quantization example of two 8-bit weights, W_0 and W_1. We set the NNZB in each weight to less than 4. Although the distribution of zero bits is irregular, the workloads of each weight are balanced, thereby achieving $8/4 = 2\times$ speedup.
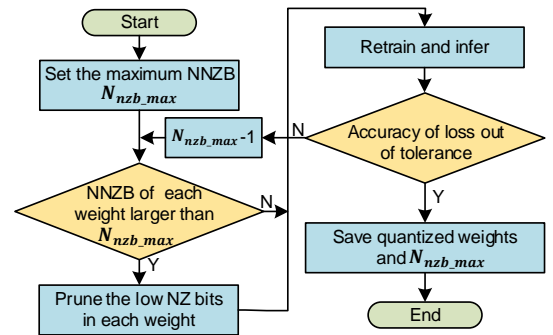

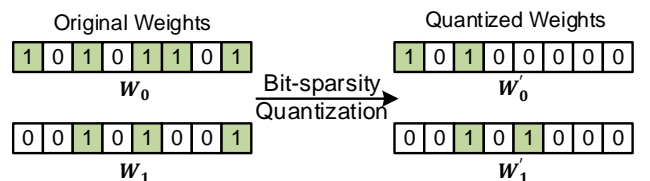Fig. 4: The flow of bit-sparsity quantization.

Fig. 5: Example of bit-sparsity quantization.

TABLE 1: Numeric Range Comparison of Weight Values

Bitwidth	N_{pb}	13	12	11	10	9	8	7	6	5	4	3
Quantization	$2^{N_{pb}}$	8192	4096	2048	1024	512	256	128	64	32	16	8
Bit-sparsity	(N_{nzb_max},N)	(13,16)	(12,16)	(11,16)	(10,16)	(9,16)	(8,16)	(7,16)	(6,16)	(5,16)	(4,16)	(3,16)
Quantization	$\sum_{i=0}^{N_{nzb\_max}}\binom{N}{i}$	65339	64839	63019	58651	50643	39203	26333	14893	6885	2517	697

3.2 Sparse Bit-serial Processing

To reduce the overhead of sparsity processing, the weights are encoded with length, sign, bit position, and bitmap, as shown in Fig.6. The length, N_{nzb_max}, represents the maximum NNZB of all weights, which only needs to be stored once for all weights in the current layer. The sign, W_s, indicates the positivity or negativity of one weight. The bit position, W_p, determines how many bits IFM should shift. The bitmap, W_b, indicates whether the bit position is valid or not since the NNZB of some weights is smaller than N_{nzb_max}.
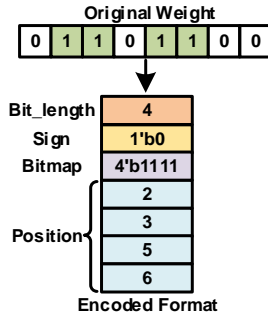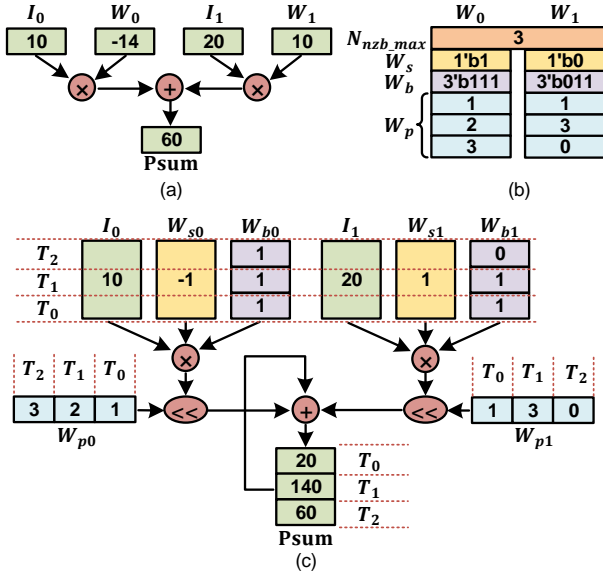

Fig. 6: Encoding format of weights.

Fig. 7: An example of sparse bit-serial computing. (a)Original bit-parallel computing. (b)Encoded Weights. (c)Process of sparse bit-serial computing.

Fig.7 shows an example of sparse bit-serial computing. Fig.7(a) shows the original MAC operation, $Psum = I_0 * W_0 + I_1 * W_1$. Fig.7(b) shows the encoding format of two weights, with the maximum NNZB, $N_{nzb\_max} = 3$. The computing flow is shown in Fig.7(c). At T_0, we fetch

$W_{p0} = 1$, $W_{p1} = 1$, and then I_0 and I_1 both shift left 1-bit, calculating $Psum = 20$. Similarly, at T_1, the Psum is calculated as 140. Since the last bitmap of W_{b1} is zero, the operation of W_{p1} is invalid and the final result is 60. The entire process takes 3 cycles, which is $2.67\times$ faster than 8 cycles of the original process.

4 BIT-BALANCE ARCHITECTURE

To achieve high speedup and energy efficiency at low hardware cost, we chose the systolic array [24] as the mainstay for high bandwidth saving and simplified inter-PE connection. To further reduce the overhead of sparse processing, the weights were encoded at software level, without introducing the corresponding preprocessing module. The control of sparse bit-serial computing is mastered by top controller based on the maximum NNZB. Thus, the PEs in our architecture only requires the shift-add units. The design of the overall architecture aims to balance the workloads of sparse weight bits at a low hardware cost, collaborating with bit-sparsity quantization.
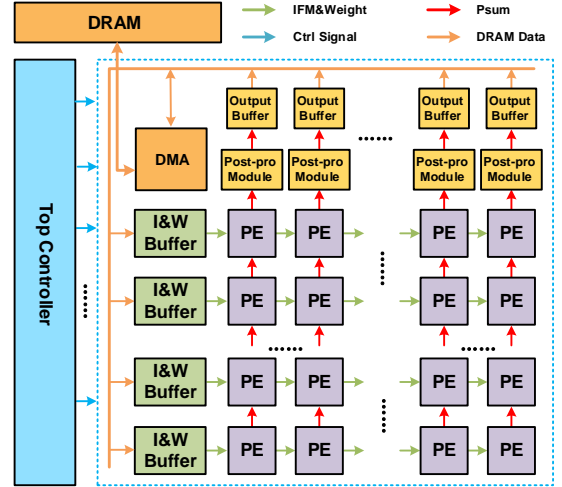

Fig. 8: Top-level architecture.

4.1 Overview

As shown in Fig.8, the top-level architecture mainly consists of input-weight buffers (I&W buffer), a systolic-based PE array, post-processing modules (Post-pro module), output buffers, a top controller, and a DMA interface. Initially, the weights in encoding format and control parameters were generated at software level. These data and input images were directly loaded into DRAMs. Next, the IFMs and weights were fetched by I&W buffers and then transferred to the PE array. After massive bit-serial operations by PEs, post-pro modules take in the outputs to perform ReLU and pooling for final OFMs, which were buffered in the output buffers temporarily. Finally, the OFMs of the current layer

were stored to DRAMs for the next layer computing. We executed layer-wise, and the OFMs of the last layer were classified to output the final results. The top controller mastered the entire computing flow and the on/off-chip data were transported through the DMA interface.

4.2 Systolic-based PE Array

The systolic-based PE array, in charge of bit-serial computing, is composed of $N_{PE} \times N_{PE}$ PEs, among which PEs of the same row share IFMs of one IC, and PEs of the same column process OFMs of one OC. Each PE contains a complement processing unit, a shift unit, and an accumulation unit, as shown in Fig.9. First, the sign of weight, W_s, chose the complement or the original value of the IFM. Then, the temporary result was shifted left based on the weight-bit position, W_p. Finally, the shifted result was accumulated with adjacent PE. To reduce power consumption, the complement processing unit and shift unit are gated when the weight bitmap, W_b, is zero. Moreover, to adapt to different bitwidths of IFMs and weights, we merged the 8-bit operation with 16-bit operation for higher resource utilization. The 16-bit IFMs with the corresponding 32-bit temporary results can be divided as two pairs of 8-bit IFMs and 16-bit temporary results, achieving $2\times$ peak throughput in 8-bit operation compared with the 16-bit.
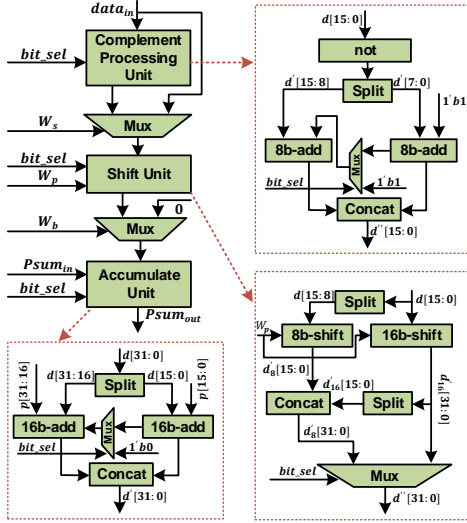

Fig. 9: PE unit.

4.3 On-chip Buffer

To reduce energy consumption of the on/off-chip data movement, we need to buffer IFMs, weights, and OFMs for data reusing. Thus, the on-chip buffers mainly consist of I&W buffers and output buffers. To avoid performance bottleneck of data loading, we utilized each group of two memories as Ping-Pong buffers, one for DRAM access and the other for PE transporting, which is similar with other systolic-array based accelerators [25]. To support adaptive bitwidth computing, the bitwidths of I&W buffers and output buffers are both 16-bit, which can store one 16-bit IFM/OFM or two 8-bit IFMs/OFMs per address. For weights in encoding format, we store 16 weight signs, 16 weight bitmaps, 4 weight positions of 16-bit precision, or 5 weight positions of 8-bit precision per address.

5 DATAFLOW OF SPARSE BIT-SERIAL PROCESSING

To reduce the energy consumption of DRAM access, we applied a dataflow to enhance data reusing with limited on-chip buffer resources. Since Bit-balance is a systolic-array-based accelerator, we merge the sparse bit-serial computing with systolic dataflow [25]. Furthermore, a mapping algorithm was designed to map various networks on our architecture, based on provided network parameters.

5.1 Data Partition

For limited on-chip buffer resources and PE array size, we need to partition IFMs and weights into independent sub-tiles in terms of edge size and input/output channel, referring to the tiling method [25]. Considering the resources of Psum storage, the size of IFM sub-tiles is set no larger than $W_{IS} \times H_{IS}$, and the sizes of IFM tiles on the width and height dimension are T_{WI} and T_{HI}, respectively. For limited PE array size, we only process N_{PE} input and output channels concurrently, with tiling numbers of T_{IC} and T_{OC}. The specific tiling parameters are shown in Tab.2. After data partition, each tile of IFMs and weights can be processed with sparse bit-serial computing independently.

TABLE 2: Parameters of Data Partition

Parameter	Explanation	Formula
N_{PE}	PE array size	-
N_{IC}	# of input channel	-
N_{OC}	# of output channel	-
W_K	Width of a kernel	-
H_K	Height of a kernel	-
W_I	Width of a IFM	-
H_I	Height of a IFM	-
W_{IS}	Width of a IFM tile	-
H_{IS}	Height of a IFM tile	-
T_{IC}	Tiling # of input channel	N_{IC}/N_{PE}
T_{OC}	Tiling # of input channel	N_{OC}/N_{PE}
T_{WI}	Tiling # of IFM width	W_I/W_{IS}
T_{HI}	Tiling # of IFM height	H_I/H_{IS}

5.2 Mapping Algorithm

Considering dataflow and partition of IFM and weight, to map various networks on Bit-balancing, we designed a network mapping algorithm. The computing flow description is shown in Tab.3. The 1st and 4th rows decide the dataflow according to Adaptive Dataflow Configuration [25]. For "reuse IFM first (RIF)", we finish the OFM computing of all OCs for one output tile before switching to the next output tile; otherwise, for "reuse weight first (RWF)", we finish computing of all output tiles for one OC before switching to the next OC. The 2nd and 3rd rows describe the partition of IFM, with a row number of T_{ofm_row} and column number of T_{ofm_col}. The 5th row drives the accumulation of input channels to obtain the final Psums. The 6th and 7th rows represent the element number of one tile of IFM and one kernel. The 8th row indicates the maximum NNZB of the weights, where N_{nzb_max} is loaded as a parameter because the weights from training are fixed. The 9th row indicates that the PE is performing the sparse bit-serial computing with IFM and the weight bit position.

TABLE 3: Computing Flow

Input: parameters of architecture configuration; IFMs; weights; **Output**: OFMs

Noting: When RIF, $T_{OC\_RWF} = 1, T_{OC\_RIF} = Toc$; otherwise, $T_{OC\_RWF} = Toc, T_{OC\_RIF} = 1$
1   $for(a = 0; a < T_{OC\_RWF}; a = a + 1)$
2    $for(b = 0; b < T_{WI}; b = b + 1)$
3     $for(c = 0; c < T_{HI}; c = c + 1)$
4      $for(d = 0; f < T_{OC\_RIF}; d = d + 1)$
5       $for(e = 0; e < T_{IC}; e = e + 1)$
6        $for(f = 0; f < W_K \times H_K; f = f + 1)$
7         $for(g = 0; g < W_K \times H_K; g = g + 1)$
8          $for(h = 0; h < N_{nzb\_max}; h = h + 1)$
9           $Psum+ = I_{NZ} \times 2^{W_P}$

6 EXPERIMENTS

6.1 Implementation

Bit-balance Implementation: To measure the area, power, and critical path delay, we implemented Bit-balance in Verilog and conducted a functional simulation on Cadence Xcelium 2019.2. Then, the Verilog code was synthesized with Synopsys Design Compiler (DC). The on-chip SRAM buffers are generated by Memory Compiler. The on-chip power is calculated by Synopsys Prime Suite 2020.09 with the Switching Activity Interface Format(SAIF) file generated from testbench. The reports of resource utilization and power breakdown are shown in Tab.4.

The size of the PE array is 32×32, achieving a peak throughput of $1024GOP/s$ and $2048GOP/s$ for 16b and 8b shift-add operations respectively. The IFMs and weights are both encoded in 16 bits or 8bit, and their Psum are correspondingly truncated to 32 bits and 16 bits respectively. Since the PEs are implemented with shift-add units, the area of computing core (CC), containing PE array, post processing module, and controller, is only $2.91mm^2$. To balance the data reuse efficiency and resource overhead, we set the IFM tiling unit as 8×8. Each I&W buffer contains two $1K \times 16b$ and two $256 \times 16b$ single-port static random-access memories (SRAMs); each output buffer and post processing module contain two $64 \times 16b$ single-port Register Files (RFs) and one $64 \times 16b$ dual-port RF, respectively. Thus, the total on-chip memories are 176KB. The power consumption mainly comes from the computing core, taking up 85% power of full chip.

TABLE 4: Resource and Power Proportion

Module	Area (mm2)	16b-Power (mW)	8b-Power (mW)
I&W Buffer	1.81(36.3%)	90(11%)	86(10%)
PE array	2.34(46.9%)	582(71%)	609(71%)
Post-pro Buffer	0.49(9.8%)	82(10%)	95(11%)
Output buffer	0.27(5.4%)	41(5%)	42(5%)
Controller	0.08(1.6%)	25(3%)	25(3%)
Computing core	2.91(58.3%)	689(84%)	729(85%)
Total	4.99	820	857

To verify the superiority of Bit-balance, we set some baseline accelerators as shown in Tab.5. The brief descriptions of each accelerator are presented as below:

1) Eyeriss [9] is a typical bit-parallel accelerator based on the 16-bit fixed-point MAC, without considering the sparsity of IFMs or weights. By comparison, Bit-balance achieves better performance through exploiting weight bit sparsity and higher energy efficiency by replacing MAC units with shift-add units.

2) Cambricon-X [13] accelerates NNs by skipping zero-weight elements computing. However, the whole architecture is designed for accelerating NNs of 16-bit precision, resulting in inefficient resource utilization in lower-bit operation. By comparison, Bit-Balance achieves higher resource efficiency through bit-serial computing.

3) Stripe [16] allows per-layer selection of precision instead of fixed-point values, which improves performance proportionally with the bitwidth of IFMs based on bit-serial computing. But it only truncates the bitwidth while ignoring the bit sparsity of IFMs. By comparison, Bit-Balance achieves higher performance by skipping the zero bits of weights.

4) Laconic [21] tears MAC operation into IFMs and weights bit-serial computing and achieves great speedup by exploiting both IFM and weight sparsity. However, the distribution of zero bits in IFMs and weights can be irregular, which causes imbalanced workload in the PE array and degrades performance. By comparison, Bit-Balance achieves higher performance through bit-sparsity quantization.

5) Bitlets [23] proposes a bit interleaving philosophy to maximally exploit bit-level sparsity, which enforces the acceleration by decreasing the number of weights involved in computing. However, the logic corresponding to the bit-interleaving occupies 40% area of the entire PE module. Besides, the computing units for the high-bit can be idle during low-bit operation, causing inefficient resource efficiency. By comparison, we achieved higher energy efficiency and resource efficiency with model-hardware co-design and adaptive bitwidth computing.

As for benchmarks, we conducted inference on AlexNet [1], VGG-16 [2], ResNet-50 [4], GoogleNet [3], Yolo-v3 [5] and compared our design with each baseline. The first four NNs are testified on ImageNet [1] and Yolo-v3 is testified on CoCo [26].

6.2 Performance

To show the advantages of sparse processing methods in Bit-balance, we compared our design against Eyeriss [9], Cambricon-X [13], Stripe [16], Laconic [21], and Bitlets [23] on normalized performance, achieving $1.6\times\sim8.6\times$, $1.1\times\sim2.4\times$, $4\times\sim7.1\times$, $2.2\times\sim4.3\times$, and $1.1\times\sim1.9\times$ speedup, respectively, as shown in Fig.10. In AlexNet, VGG-16, ResNet-50, GoogleNet, and Yolo-v3, we set the maximum NNZB, N_{nzb_max}, as 3~4 in 16-bit precision and 4~5 in 8-bit precision, respectively, as shown in Tab.6. The columns of Top 1 and Top 5 accuracy show the inference accuracy of each NN and its corresponding accuracy loss in the brackets, which is referred to the training structures of Pytorch [27]. The performance is calculated by the ratio of frequency and total cycles of inference computing, achieving $4\times\sim8\times$ speedup compared with the basic bit-serial computing of 16-bit precision. Though the accuracy of 16-bit precision is higher than the 8-bit and its maximum NNZB is also smaller, the peak throughput of the 8-bit is twofold, resulting in higher performance naturally. The normalized performance, R_p, is calculated by the ratio of our performance and other

TABLE 5: Overall Comparison of Each Accelerator

Accelerator	Technology (nm)	# of PEs	Area (mm²)	Frequency (MHz)	Peak Throughput (GOP/s)	Power (mW)	Classification
Eyeriss [9] JSSCC-2017	65nm	168	12.25@full chip	200	33.6@16b MAC	278@AlexNet 234@VGG-16	Bit Parallel
Cambricon-X [13] MICRO-2016	65nm	16	6.38@full chip	1000	256@16b MAC	278	Sparse Bit Parallel
Stripe [16] CAL-2017	65nm	4096	122.1@full chip	980	64225@16b shift-add	-	Bit Serial
Laconic [21] ISCA-2019	65nm	512	4.1@CC	1000	8192@1b shift-add	-	Sparse Bit Serial
Bitlet [23] MICRO-2021	65nm	32	5.80@CC	1000	768@24b shift-add	1390@16b 1199@ 8b	Sparse Bit Serial
Bit-balance	65nm	1024	4.99@full chip 2.91@CC	1000	1024@16b shift-add 2048@8b shift-add	820@16b 857@ 8b	Sparse Bit Serial

TABLE 6: IFM and Weight Sparsity Ratios and Accuracy Loss of Each NN

NN Types	16-bit Precision				8bit-precision			
	N_{nzb_max}	Performance (frame/s)	Top-1 Accuracy(%)	Top-5 Accuracy(%)	N_{nzb_max}	Performance (frame/s)	Top-1 Accuracy(%)	Top-5 Accuracy(%)
AlexNet	3	270.5	55.6(-0.9)	78.8(-0.3)	5	326.2	55.6(-0.9)	78.8(-0.3)
VGG-16	3	20.4	70.8(-0.8)	89.9(-0.5)	4	30.1	71.2(-0.4)	90.1(-0.3)
GoogleNet	4	136.2	69.1(-0.7)	89.3(-0.3)	5	218.4	69.1(-0.7)	89.3(-0.3)
ResNet-50	3	46.8	75.2(-0.9)	92.4(-0.5)	5	56.3	74.9(-1.2)	92.4(-0.5)
Yolo-v3	3	10.9	61.9(-0.7)	-	4	16.4	60.3(-2.3)	-

accelerators. Since Cambricon-X, Stripe, and Laconic didn't provide their performance in the paper, the normalized performance is calculated based on the comparison with Bitlet and Eyeriss in our benchmarks.

Compared with bit-parallel architectures, we leveraged bit-serial computing instead of 16-bit MAC unit. For Eyeriss, since it has been taped out, we assume the frequency of Eyeriss can reach to 1GHz for a fair comparison (Eyeriss-S in Fig.10). Though we consumed multiple cycles for one MAC operation in a PE, our PE array size is larger than Eyeriss. Besides, the execution time of one MAC can be shrunk to N_{nzb_max} cycles by skipping zero bits. In the best case, we managed to achieve $30.1/(3.5) = 8.6\times$ speedup with 8-bit precision at VGG-16. For Cambricon-X, it exploits the weight element sparsity to accelerate computing. However, the sparsity of weights engaged in each PE can be imbalanced, causing performance degradation. Considering both sparsity exploiting and PE array size, we can achieve $1.1\times \sim 2.4\times$ performance improvement.

Compared with bit-serial architectures, we accelerate the NNs cooperated with bit-sparsity quantization. We only quantized the NNZB instead of the bitwidth in Strip. Therefore, our quantization method can accommodate wider numeric range of weight values and the NNZB in Bit-balance is smaller than the bitwidth in Stripe. Thus, we obtained $4\times \sim 7.1\times$ speedup. Comparing with Laconic, we balanced the workloads across PE array by constraining the NNZB to a certain value. Though Laconic exploits sparsity of both IFMs and weights, its computing time is determined by the longest non-zero bit sequence, causing performance degradation once there exists a very long bit sequence involved in computing. Thus, we obtained $2.2\times \sim 4.3\times$ performance improvement through load-balancing. For Bitlet, its performance improved by the bit-interleaving is similar with our method at the 16-bit precision. But we applied the adaptive datawidth computing for higher performance in low-bit operation, while in Bitlet, some computing units remain

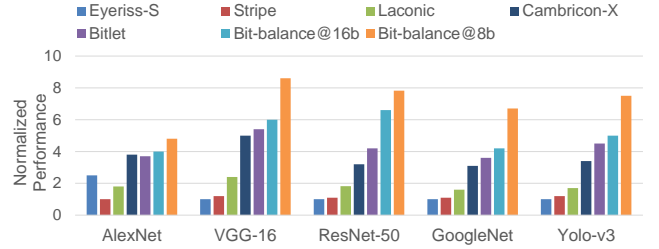idle. In the best case, we obtained $56.3/29 = 1.9\times$ speedup in 8-bit precision at ResNet-50.

Fig. 10: Performance comparison with Eyeriss, Cambricon-X, Stripe, Laconic, and Bitlets.

6.3 Energy Efficiency and Resource Efficiency

Since the processing methods of each accelerator are different, it's essential to explore whether the benefits outperform the incurred overhead. We compared the energy efficiency and resource efficiency of Bit-balance with Eyeriss [9], Cambricon-X [13], Stripe [16], Laconic [21], and Bitlet [23], achieving $2.7\times \sim 11.3\times$, $1.3\times \sim 2.8\times$, $3\times \sim 5.6\times$, $2.7\times \sim 5.4\times$, and $1.8\times \sim 2.7\times$ energy efficiency improvement, respectively, as shown in Fig.11, and achieving $3.9\times \sim 21\times$, $1.6\times \sim 3.9\times$, $1.7\times \sim 3\times$, $3.2\times \sim 6.3\times$, and $2.1\times \sim 3.8\times$ resource efficiency improvement, respectively, as shown in Fig.12. The energy efficiency is calculated as R_{np}/R_p, where R_{np} and R_p represent the ratio of Bit-balance and other baselines in normalized performance and power, respectively. For resource efficiency, the ratio of power, R_p, is replaced with the ratio of area, R_a, in the expression of energy efficiency. Since the PE array size of Stripe has been scaled for computing normalized performance, the expression, R_{np}/R_a should multiply the ratio of peak performance, R_{pp}, representing the throughput per area.

Compared with fixed-point architectures, the MAC units

consume more area and power than add-shift units in Bit-balance. For Eyeriss, the PE consists of a MAC unit and some memories, where the MAC unit only accounts for 9% of total area and power, indicating that Eyeriss consumes much more resources and energy for one MAC operation than that in Bit-balance. Noting that we scale the frequency of Eyeriss to 1GHz, the power should be scaled up by $5\times$ while the area remains unchanged. In the best case, we achieve $8.6/(857/(236 \times 5)) = 13.4\times$ energy efficiency and $8.6/(4.99/12.25) = 21\times$ resource efficiency improvement in 8-bit precision at VGG-16. For Cambricon-X, it introduces the indexing module for sparse weights processing, accounting for 31% of the total area and 34% of the total power. However, the processing of sparse weights in Bit-balance are similar with dense bit-serial computing, inducing little overhead. Thus, we achieve $1.3\times\sim2.8\times$ energy efficiency and $1.6\times\sim3.9\times$ resource efficiency improvement.
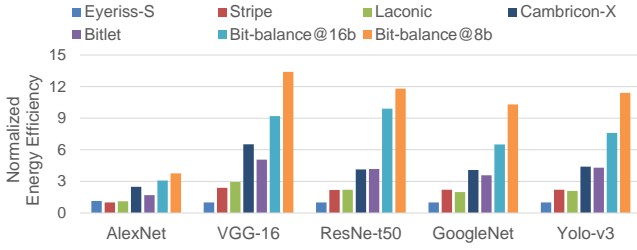
logic units are incurred for processing the intermediate results, resulting in $4.1/2.91 = 1.4\times$ computing core area overhead compared with Bit-balance. Therefore, we achieve $2.7\times\sim5.4\times$ energy efficiency and $3.2\times\sim6.3\times$ resource efficiency improvement. For Bitlet, its computing process consists of three steps: preprocessing, dynamic exponent matching and bit distillation. Each step is implemented in the corresponding modules, introducing 45% area and 69% power overhead. However, the preprocessing of weights in Bit-balance is executed offline, inducing no logic overhead. And the process of skipping zero bits is mastered by the top controller module, without inducing sparse processing logic into PEs. Thus, in the best case, we achieve $1.9/(857/1199) = 2.7\times$ energy efficiency and $1.9/(2.91/5.80) = 3.8\times$ resource efficiency improvement in 8-bit precision at ResNet-50.

Fig. 11: Energy efficiency comparison with Eyeriss, Cambricon-X, Stripe, Laconic, and Bitlets.

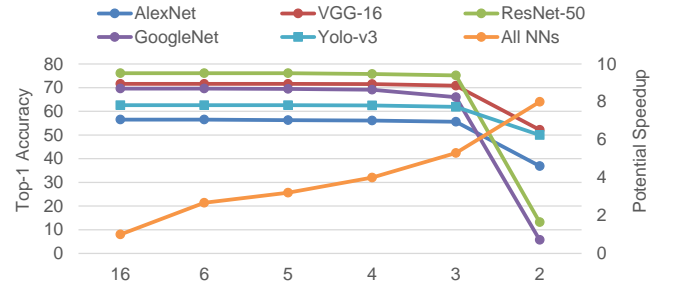Fig. 13: Potential Speedup and accuracy loss with different N_{nzb_max} in 16-bit precision.

Fig. 12: Resource efficiency comparison with Eyeriss, Cambricon-X, Stripe, Laconic, and Bitlets.

Fig. 14: Potential speedup and accuracy loss with different N_{nzb_max} in 8-bit precision.

Compared with bit-serial architectures, the overhead of sparse computing in Bit-balance is smaller. For Stripe, it merged 16 shift-add units to one module, simplifying the intermediate steps, while the shift-add operations are performed individually in each PE of Bit-balance for sparse processing. Besides, since the PE array size scales by square while the memory scales linearly, the larger PE array size is, the less proportion memory accounts for. Thus, it consumes $(64225/1024)/(122.5/4.99) = 2.5\times$ less resource than Bit-balance for one add-shift operation. Combining the overhead with performance, we achieve $3\times\sim5.6\times$ energy efficiency and $1.7\times\sim3\times$ resource efficiency improvement. For Laconic, its PE unit decomposed the MAC into 16 pairs of IFM-bit and weight-bit operations. Though it can exploit the sparsity of both IFM and weight bit, many

6.4 Sensitivity to Sparsity

With higher sparsity of weight bits comes higher performance, under the sacrifice of accuracy. To explore the influence of weight-bit sparsity on performance and accuracy, we trained several groups of weights with different maximum NNZBs, as shown in Fig.13 and Fig.14. The top-1 accuracy of float 32-bit precision in AlexNet, VGG-16, ResNet-50, GoogleNet, and Yolo-v3 are 56.5%, 71.6%, 76.1%, 69.8%, and 62.6%, respectively referring to Pytorch [27]. For 16-bit precision, the accuracy loss of each NN without bit-sparsity quantization is only 0~0.2%. Then, we quantized the maximum NNZB to 2~6, achieving $2.67\times\sim8\times$ speedup. In the stage of $N_{nzb\_max} = 6\sim4$, the accuracy of all NNs remains stable, with only 0.1%~1.0% loss. At the point of

$N_{nzb\_max} = 3$, the accuracy of GoogleNet drops about 3.7%. When $N_{nzb\_max} = 2$, the accuracy of all NNs precipitously drops. Thus, we chose $N_{nzb\_max} = 3$ or 4 for 16-bit precision in our work.

The phenomenon in 8-bit precision is similar with the 16-bit precision. The 8-bit precision accuracy loss of each NN without bit-sparsity quantization is 0.2%~2.1%. In the stage of $N_{nzb\_max} = 7$~5, the accuracy holds steady. At the point of $N_{nzb\_max} = 4$, the accuracy of AlexNet, ResNet-50, and GoogleNet drops about 1.9%~2.8%. When $N_{nzb\_max} = 3$, the accuracy of all NNs precipitously degrades. Thus, we chose $N_{nzb\_max} = 4$ or 5 for 8-bit precision computing.

Though the performance improvement of 8-bit precision is higher than the 16-bit in Bit-balance owing to the adaptive bitwidth computing, there still exits some NNs of specific domains that are more suitable for 16-bit precision to maintain accuracy. For example, the accuracy of Yolo-v3 with 16-bit precision is 2% higher than that of 8-bit precision on average, which is a significant accuracy improvement for NNs. Thus, adaptive bitwidth is essential for sparse bit-serial computing in NN acceleration.

6.5 Comparison with Basic Bit-serial Architecture

Since Bit-balance processes the weights with encoding format, it induces storage overhead and increases power consumption of DRAM access compared with original weights. To illustrate the benefit of sparse processing in Bit-balance, we compared the energy efficiency of the whole system with basic bit-serial architecture (Bit-balance without sparse processing). The power of DRAM access is estimated with CACTI [28] by the total DRAM access counts and runtime. Based on the dataflow in Section.V, though we consume $1\times$~$1.4\times$ power for $1\times$~$2.4\times$ DRAM access as shown in Fig.15 and Fig.16, the energy efficiency of Bit-balance is still $1.14\times$~$5.3\times$ higher than basic bit-serial architecture owing to $1.6\times$~$5.3\times$ performance improvement as shown in Fig.17.
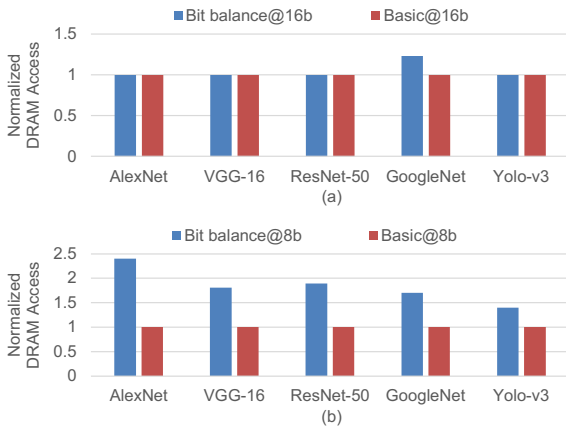

Fig. 15: DRAM access comparison with basic bit-serial architecture. (a)16-bit precision. (b)8-bit precision.

For 16-bit precision, the storage per weight of encoding format with $N_{nzb\_max} = 3$ is 16-bit, which allocates 1 bit for sign, 3 bits for bitmap, and 3×4 bits for weight-bit

position. In the case of $N_{nzb\_max} = 4$, the storage per weight is $1 + 4 + 4 \times 4 = 21$-bit. Since the encoding format induces $1\times$~$1.3\times$ weight storage overhead compared with original 16-bit weight, the DRAM access increases $1\times$~$1.23\times$. Thus, Bit-balance consumes $1\times$~$1.07\times$ power. Owing to $4\times$~$5.3\times$ performance, the energy efficiency of Bit-balance is $3.73\times$~$5.3\times$ higher than basic bit-serial architecture.

For 8-bit precision, the storage per weight with $N_{nzb\_max} = 4$ and $N_{nzb\_max} = 5$ are 17-bit and 21-bit respectively, inducing $2.1\times$~$2.6\times$ weight storage overhead compared with origianl 8-bit weight. Thus, Bit-balance consumes $1.12\times$~$1.41\times$ power for $1.4\times$~$2.4\times$ DRAM access. Owing to $1.6\times$~$2\times$ performance, the energy efficiency of Bit-balance is $1.14\times$~$1.79\times$ higher than basic bit-serial architecture.
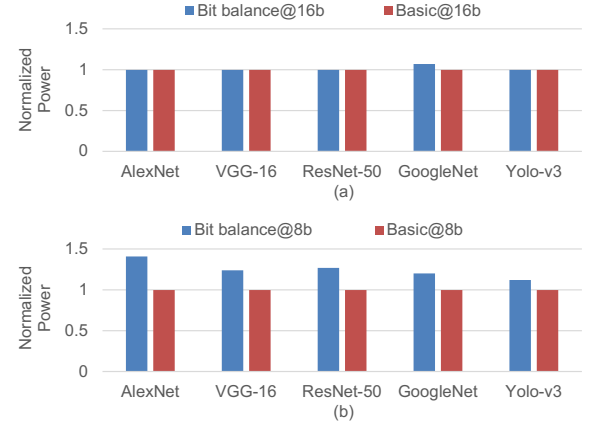

Fig. 16: Power comparison with basic bit-serial architecture. (a)16-bit precision. (b)8-bit precision.
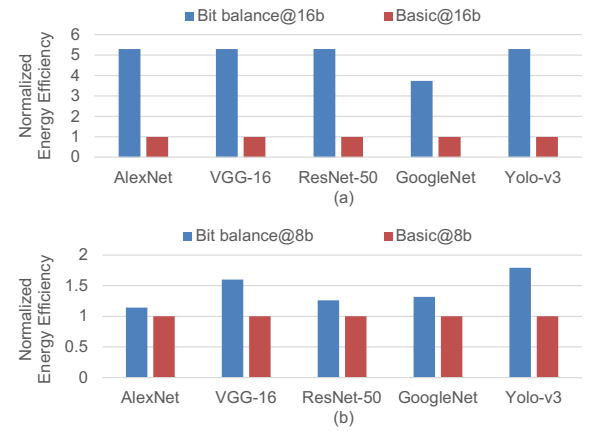

Fig. 17: Energy efficiency comparison with basic bit-serial architecture. (a)16-bit precision. (b)8-bit precision.

7 Conclusion

This paper proposed a sparse bit-serial accelerator, called Bit-balance, for sparse weight bit processing, achieving significant performance improvement with low hardware cost. Meanwhile, we co-designed a bit-sparsity quantization method to maintain the maximum NNZB of weights to no more than a certain value with little accuracy loss, which can

effectively balance the workloads of sparse weight bits in the PE array. Furthermore, to support adaptive bitwidth computing, we merged the 8-bit precision with 16-bit precision operation for higher resource efficiency. Compared with the previous sparse accelerators, Bit-balance can achieve better performance and energy efficiency with smaller area.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: https://doi.org/10.1145/3065386

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[3] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[4] K. He, X. Zhang, S. Ren *et al.*, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[5] J. Redmon, S. Divvala, R. Girshick *et al.*, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[6] T. N. Sainath, O. Vinyals, A. Senior *et al.*, "Convolutional, long short-term memory, fully connected deep neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4580–4584.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 675–678. [Online]. Available: https://doi.org/10.1145/2647868.2654889

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622.

[9] Y.-H. Chen, T. Krishna, J. S. Emer *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[10] N. P. Jouppi, C. Young, N. Patil *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[11] L. Deng, G. Wang, G. Li *et al.*, "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 8, pp. 2228–2246, 2020.

[12] S. Han, X. Liu, H. Mao *et al.*, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243–254.

[13] S. Zhang, Z. Du, L. Zhang *et al.*, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[14] J. Albericio, P. Judd, T. Hetherington *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1–13.

[15] A. Parashar, M. Rhu, A. Mukkara *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 27–40.

[16] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 80–83, 2017.

[17] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[18] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[19] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.

[20] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 382–394.

[21] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 304–317.

[22] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "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*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 749–763. [Online]. Available: https://doi.org/10.1145/3297858.3304041

[23] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 963–976. [Online]. Available: https://doi.org/10.1145/3466752.3480123

[24] Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[25] W. Sun, "Sense: Model hardware co-design for accelerating sparse cnn on systolic array," https://arxiv.org/abs/2202.00389.

[26] "Coco dataset," https://cocodataset.org/#download.

[27] M. Paganini, "Pruning tutorial," https://pytorch.org/tutorials/intermediate/pruning_tutorial.html.

[28] R. Balasubramonian, A. B. Kahng, N. Muralimanohar *et al.*, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, jun 2017. [Online]. Available: https://doi.org/10.1145/3085572