doi:10.3772/j.issn.1006-6748.2024.01.006

Cambricon-QR: a sparse and bitwise reproducible quantized training accelerator①

LI Nan(李 楠)[*] [**] [***], ZHAO Yongwei[**], ZHI Tian[**], LIU Chang[**] [***], DU Zidong[**], HU Xing[**], LI Wei[**], ZHANG Xishan②[**] [***], LI Ling[****], SUN Guangzhong[*]

([*] School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, P. R. China)

([**] State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100086, P. R. China)

([***] Cambricon Tech. Ltd, Beijing 100191, P. R. China)

([****] Institute of Software, Chinese Academy of Sciences, Beijing 100086, P. R. China)

Abstract

Quantized training has been proven to be a prominent method to achieve deep neural network training under limited computational resources. It uses low bit-width arithmetics with a proper scaling factor to achieve negligible accuracy loss. Cambricon-Q is the ASIC design proposed to efficiently support quantized training, and achieves significant performance improvement. However, there are still two caveats in the design. First, Cambricon-Q with different hardware specifications may lead to different numerical errors, resulting in non-reproducible behaviors which may become a major concern in critical applications. Second, Cambricon-Q cannot leverage data sparsity, where considerable cycles could still be squeezed out. To address the caveats, the acceleration core of Cambricon-Q is redesigned to support fine-grained irregular data processing. The new design not only enables acceleration on sparse data, but also enables performing local dynamic quantization by contiguous value ranges (which is hardware independent), instead of contiguous addresses (which is dependent on hardware factors). Experimental results show that the accuracy loss of the method still keeps negligible, and the accelerator achieves 1.61 × performance improvement over Cambricon-Q, with about 10% energy increase.

Key words: quantized training, sparse accelerator, Cambricon-QR

0 Introduction

Convolutional neural network (CNN) has become the wide-spread technology in computer vision (CV). However, it is difficult to deploy them efficiently on resources-constrained devices, e.g., cellphones, wearables, etc., especially for training. Quantization is a promising technique to reduce the computation cost by using low bit-width data representations and arithmetics to approximate the full-precision floating point data. However, quantized training is more challenging than quantized inference due to the much higher vulnerability of numerical errors. Recently, state-of-the-art quantized training algorithms succeeded in training CNNs in 8-bit or even less, while still maintaining negligible accuracy loss, e.g., precision-adaptive quantization[1-2], specially-designed data format[3-4], gradient clipping[5-6], etc. However, due to the dynamic quantization approach adopted by these algorithms, they depend on specialized architectural support to achieve actual speedup.

Cambricon-Q[7] is a state-of-the-art accelerator for quantized training. It adopts the dynamic quantization approach from tensor-wise to per local data blocks, thus enables on-the-fly dynamic quantization with specialized units to eliminate excessive data accesses. It also features a near-data-processing (NDP) engine to allow weights updated in-place, reducing data movements furthermore. As a result, Cambricon-Q achieves 4.2 ×/1.6 × training speedups compared with GPU/TPU, which illustrates a viable option to enable train-

① Supported by the National Key Research and Devecopment Program of China (No. 2022YFB4501601), the National Natural Science Foundation of China (No. 62102398, U20A20227, 62222214, 62002338, U22A2028, U19B2019), the Chinese Academy of Sciences Project for Young Scientists in Basic Research (YSBR-029) and Youth Innovation Promotion Association Chinese Academy of Sciences.

② To whom correspondence should be addressed. E-mail: zhangxishan@ict.ac.cn.

Received on Mar. 2, 2023

ing on mobile and edge devices.

However, there are still two caveats in the design of Cambricon-Q. Firstly, the numerical errors are not reproducible. Local dynamic quantization slices the data into local blocks, and only performs dynamic quantization per block. Therefore, the quantization error depends on the slicing. For example, the data in NHWC format will be sliced differently from that in NCHW format, leading to inconsistent numerical behaviors. Other depending factors include hardware specifications (e. g., block sizes, buffer sizes, PE array sizes), programming (e. g., tiling options and task scheduling), compiling and optimizations (e. g., instruction scheduling, memory allocations), etc. Although Cambricon-Q is resilient enough to the slightly varying errors, the non-reproducible numerical behavior may still become a major concern in critical applications such as security, healthcare, autonomous driving. Secondly, Cambricon-Q cannot leverage data sparsity. By utilizing sparsity, considerable cycles could be further squeezed out, which is essential to the application in resource-constrained scenarios.

To address these problems, the acceleration core of Cambricon-Q is redesigned to support fine-grained irregular data processing. Enabled by the fine-grained irregularity support, a new quantization technique is proposed, whose numerals are bit-wise reproducible regardless of hardware factors. Instead of slicing by addresses, the quantization technique divides the data into groups of contiguous value ranges, which is algorithmic deterministic. The data from the same group is placed discontiguously in memory, and processed with the fine-grained irregularity support. Simultaneously, the support for fine-grained irregularity also enables the utilization of sparsity when training sparse CNN models.

Experiments are conducted on various CNNs. The quantization technique achieves negligible accuracy loss. The redesigned variant of Cambricon-Q, namely Cambricon-QR, outperforms the original design by 1.61 × on average with about 10% extra energy consumption.

1 Background and problem statement

1.1 Quantization

Quantization refers to techniques for performing computations and storing tensors at lower bit-widths than floating point precision, in order to save storage space, speed up computations and save device power consumption. The common formula is $x_q = round\left(\frac{x - offset}{scale}\right)$, where x is the full-precision data, x_q is the quantized data, and *offset* and *scale* are quantization parameters. The quantization can be performed at different stages, corresponding to different quantization types.

Quantization aware training (QAT)[8] was proposed by Google, which models quantization errors in both forward and backward passes by introducing fake-quantization modules. The inputs and weights are quantized and then dequantized to make the loss aware of the quantization errors, thus reducing the loss of inference accuracy on quantized models. The entire computation is performed in floating point.

Post training quantization (PTQ) is performed during inference of neural networks. Much work has been done in this area, including data-free quantization (DFQ)[9], analytical clipping for integer quantization (ACIQ)[10], piece-wise linear quantization (PWLQ)[11], etc.

Quantized training. Different from QAT which aims to improve inference accuracy, quantized training uses quantized data in forward and backward passes to reduce training cost. The main challenge is that training accuracy is much more sensitive to data precision than inference accuracy. Especially, the gradients, during training may lead the optimization of the model to a wrong direction due to quantization errors. DoReFa-Net[12] quantizes gradients of CNNs to different bit-widths and suffers from obvious accuracy loss at low bit-width. Ref. [13] proposed a full-integer training scheme and used a layer-wise scaling factor to replace batch normalization (BN). Ref. [14] quantized BN layer based on Ref. [13] but still suffered from accuracy loss. Ref. [15] used 8-bit floating point numbers for training and proposed chunk-base accumulation to avoid data swamping. Ref. [5] also applied int8 training and tried to minimize the gradient quantization error layer-wise by clipping the gradients periodically with accuracy loss less than 2%. Ref. [6] still cliped the gradients but quantized them channel-wise, achieving negligible accuracy loss. Ref. [3] used block-floating-point (BFP) and proposed a variant of BFP to make the quantization parameters unchanged after transposing the tensor. Ref. [1] adopted a layer-wise precision-adaptive quantization technique using int8 and int16. Ref. [4] proposed a piece-wise fixed point (PWF) format to reduce the quantization error for gradients close to zero.

Quantization can be divided into static quantization and dynamic quantization according to how to obtain the quantization parameters. The former requires

calibration with a representative dataset to determine optimal quantization parameters offline. The latter computes the statistics of data on-the-fly to get quantization parameters.

Due to the fact that the distributions of gradients change significantly during training, most of current quantized training algorithms use dynamic quantization to get the quantization range, which leads to great memory and compute overhead, and extra data movements. To make these algorithms work efficiently, special hardware is designed. Cambricon-Q is a first hardware architecture for quantized training with negligible accuracy loss. It uses local dynamic quantization by slicing the data into fixed-size blocks and implements the corresponding hardware unit. Cambricon-Q also equips the DRAM with a near-data-processing engine to avoid data transferring when updating weights. This work is based on Cambricon-Q.

1.2 Sparsity

The utilization of sparsity can reduce the computation cost of neural networks effectively. Many neural networks take ReLU, i.e., $f(x) = \max\{0, x\}$ as their activation function, which will generate many zeros, increasing the sparsity of neurons significantly. If some pruning techniques are taken, some degree of weight sparsity can also be obtained.

Many accelerators have been designed for sparse neural networks. Cambricon-X[16] filtered unnecessary neurons by an indexing module, and compressed sparse weight by step indexing, but it makes no use of neuron sparsity. Ref. [17] addressed both neuron and weight sparsity, but it only supported full-connection (FC) layer. Cambricon-S[18] proposed a pruning technique to reduce the irregularity of sparse neural networks and selected neurons needed by a neural selector module (NSM) along with weights. Ref. [19] explored sparse and irregular general matrix-matrix multiplications (GEMM) and introduced a flexible and scalable architecture which offers high utilization of its processing elements (PEs). All these work has not been well integrated with quantized training. Ref. [2] supported mixed-precision training of low bit-widths while addressing sparsity, but it only supported float16 and float8.

1.3 Problem statement

Lack of reproducibility. Despite the efficiency and negligible accuracy loss, there are still caveats in Cambricon-Q. For the same tensor, the quantization result of Cambricon-Q may be different if the tensor is fed to it with different layouts (i.e., NHWC and NCHW), due to the local dynamic quantization of fixed-size blocks, which will further affect the training results. It means that even given the same hyper-parameters, datasets and random seeds, the results may be different, which is unreasonable. One may try to solve this problem by adding a flag to the quantization unit of Cambricon-Q to indicate the data layout for channel-wise quantization, or tensor ID for tensor-wise quantization, which is still not practical. One tensor is usually sliced to fit the factors of the hardware when fed to it, thus making the results relevant with the hardware. So this work proposes a quantization technique which is irrelevant with both factors, i.e., splitting data by contiguous value ranges instead of addresses.

Irregularity and small convolutions. Dealing with sparsity is a necessity for the proposed quantization technique. As previously mentioned, the quantization technique slices data into groups by different value ranges. It means the numbers of a dot-product operation fed to the accelerator can be divided into several parts of irregular sizes. If the current design of Cambricon-Q is used continually, whose PE can only output one partial sum one time, many cycles will be needed to finish a dot-product operation. This problem can be solved by regarding these parts of data as several groups of fine-grained irregular data.

Besides, in some light-weight neural networks, e.g., MobileNet[20] and Xception[21], some layers adopt depth-wise convolutions which apply a single convolution filter for each input channel, to reduce computations and the number of parameters. Such depth-wise convolutions are usually rather small, e.g., 3×3, which are not friendly to hardware. The utilization of a 32-input adder tree which is fed with this convolution is only 28.125%. So this work redesigns the acceleration core of Cambricon-Q to address irregularity and small convolutions.

2 Algorithmic-deterministic quantization technique

In this section, an element-wise quantization (EWQ) technique is introduced, whose results are algorithmic-deterministic, and independent of the data layout and the hardware, to guarantee the reproducibility of quantization results.

Existing quantization techniques usually need to get the statistics of a set of data, e.g., the maximum and minimum value, to get *scale* and *offset*. It is costing, and the quantization results depend on how the data is grouped. Though the data can be grouped by layer or channel, it is not reasonable for hardware to

implement these quantization techniques because the data fed to hardware is usually sliced again according to hardware specifications. Instead, EWQ groups data by the data range it belongs to, and the quantization result of each element only depends on the element itself.

Grouping. One simple way of grouping is to group float numbers by their exponents, e. g., grouping float16 numbers into 32 groups. However, it is not flexible and takes no consideration of different data distributions. If the data in a range is very dense, the range needs to be split into more groups by using not only the exponents but some bits of the mantissas when grouping. If the data in a range is rather sparse, some groups can be merged into one by using just some high bits of the exponents. Thus, this work proposes quantization prefix codes to measure the quantization granularity. Quantization prefix code is composed of the exponent and some high bits of the mantissa, or only some high bits of the exponent. For example, if 011000101 is a quantization prefix code, the range it represents is from 0110000101000000 to 0110000101111111 in binary form. If the prefix of a float-point number is the same with the quantization prefix code, the number belongs to the range represented by the quantization prefix code. It can also be noticed that the longer the quantization prefix code is, the narrower range it represents, and the more precise the quantization is. Specially, a single group needs to be spared for number zeros for the addressing of sparsity.

Scale and *offset*. Given the quantization prefix code, *scale* and *offset* can be computed. Let a float16 number be x, and its exponent and mantissa be E and M, $M = f_9 f_8 \cdots f_0 = \sum_{i=0}^{9} 2^{i-10} f_i$, then $x = 2^{E-15}(1+M)$. If the length of quantization prefix code is l and the bit-width of the quantized integer x_q is w, *scale* and *offset* can be obtained as follows (only normalized numbers are considered here, the case for denormalized numbers is more complex):

(1) $l \geqslant 6$. The quantization prefix code contains not only the 5-bit exponent but the high $l-5$ bits of the mantissa. At most $w-1$ bits need to be extracted from the following bits, so:

$$x_q = f_{14-l} f_{13-l} \cdots f_{16-w-l} = \sum_{i=0}^{w-2} 2^i f_{16-w-l+i} \quad (1)$$

$$scale = 2^{-(E-15)} \times 2^{l-5+w-1} = 2^{9+l+w-E} \quad (2)$$

$$x \cdot scale - x_q = (2^{l+w-6} + 2^{l+w-7} f_9 + \cdots + 2^{w-1} f_{15-l}) + (2^{-1} f_{15-w-l} + \cdots + 2^{l+w-16} f_0) \quad (3)$$

From Eq. (3), it can be noticed that:

$$bias = 2^{l+w-6} + 2^{l+w-7} f_9 + \cdots + 2^{w-1} f_{15-l} \quad (4)$$

$$err = 2^{-1} f_{15-w-l} + \cdots + 2^{l+w-16} f_0 \quad (5)$$

If $l + w \geqslant 16$, there will be no quantization errors for float16 numbers with this quantization prefix code.

(2) $l < 6$. The quantization prefix code contains only the high l bits of the exponent. The high $w-2$ bits of the mantissa should be extracted, so:

$$x_q = 1 f_9 f_8 \cdots f_{12-w} = 2^{w-2} + \sum_{i=0}^{w-3} 2^i f_{12-w+i} \quad (6)$$

$$scale = 2^{w-2-b} \quad (7)$$

$$bias = 0 \quad (8)$$

where, b is the upper bound of the exponent represented by the quantization prefix code, i. e., $b = (E \mid (2^{5-l} - 1)) - 15$.

The quantization formula is $x \cdot scale - bias = x_q$. The round mode is consistent with IEEE 754, i. e., rounding to even. Besides, EWQ avoids the problem of the long-tail-distributed data naturally because it groups data by proximity.

3 Cambricon-QR architecture

In this section, the detailed architecture of Cambricon-QR is presented. It follows the training framework of Cambricon-Q, so this section mainly discusses about the design of EWQ in Section 2 and the leverage of sparsity of neural networks.

3.1 Overview

Fig. 1 shows the overall architecture of Cambricon-QR which is modified based on Cambricon-Q. While the NDP Engine is used for training, the acceleration core is mainly discussed here, which is mainly composed of three parts: 5 on-chip buffers, a PE array and a scalar functional unit (SFU). The 5 buffers include NB in (for fixed input neurons), NB out (for float output neurons), SB (for fixed weights), NIB (for group index of input neurons) and SIB (for group index of weights). The PE array performs vector/matrix computing, and the SFU performs scalar operations.

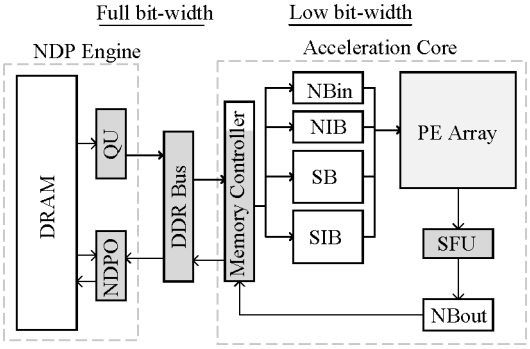

Fig. 1 Architecture overview of Cambricon-QR

The acceleration core gets quantized data and coupled group indexes from NDP Engine through the memory controller, and stores them in the on-chip buffers. The PE array gets data from buffers, computes and dequantizes the result, which is sent to SFU. Then, SFU performs scalar operations, of which the result is stored in NBout and then sent back to DRAM.

3.2 Quantization support

The quantization unit (QU) supports different quantization granularity by configuring quantization prefix codes and quantization bit-width which are stored in registers.

From the quantization formula in Section 2, it can be noticed that it is very easy for the hardware to implement EWQ, as shown in Fig. 2 (not including rounding and true form to 2's complement). The quantization group which the number belongs to can also be obtained by performing XOR operation on the number and each quantization prefix code.

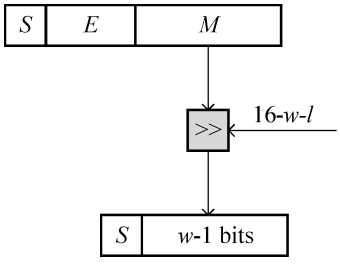

Fig. 2 Float16 quantization with prefix code len ⩾ 6

3.3 Sparsity and irregularity support

The sparsity of data is leveraged by the PE array which consists of several PE lines. As shown in Fig. 3, each PE line contains several PEs, an accumulation unit (ACCU) and a dequantization unit (DQU). Each PE includes a data selecting unit (DSU), a data dispatching unit (DDU), several multipliers and an adder tree.

For a dot-product operation, the compiler will divide it into several parts according to the capacity of one PE line. Each part is a task and dispatched to a PE line according to the utilization of them. Because one PE line may have more than one task and EWQ splits one task into several parts, the hardware gives each number a task ID to distinguish.

DSU takes both neurons and weights with their group indexes and task IDs as inputs, and gives the needed pairs of neuron-weight and the scales of their products, as shown in Fig. 4. The pair of neuron and weight can be deserted if either of them is zero (i.e., the group index is zero.), so an AND operation is applied to neurons and weights to get a bitmap for filter. The scales of the products are just summation of the scales of neurons and weights. Additionally, to avoid the transmission of offsets, the neurons and weights here are actually the summation of quantized results and offsets, i.e., $x_q + bias$. The task IDs are used in the adder tree and ACCU to distinguish different tasks because the PE can be dispatched more than one task.

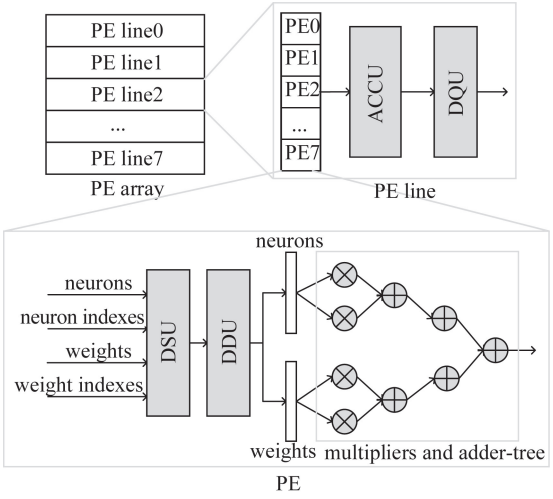

Fig. 3 PE array

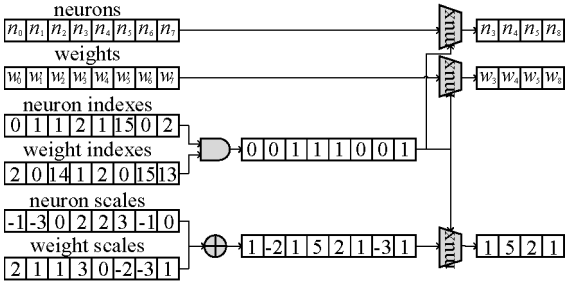

Fig. 4 Data selecting unit

DDU takes the filtered neurons, weights, scales and task IDs as inputs, and outputs the grouped data, which means neurons/weights with the same scales of the products and task IDs are placed contiguously for the convenience of the following computing unit. As shown in Fig. 5, the scales are compared with every possible value to get corresponding bitmaps, and then the neurons and weights are filtered by these bitmaps and concatenated as a whole. It can also be regarded as reordering the data by the scales and task IDs.

The adder tree gets data from multipliers and accumulates them as the final fixed result of this PE. However, the adder tree usually gives one result, which means the operands fed to it must have the same scale and task ID. When performing small convolutions (e.g., depth-wise convolution in MobileNet[20], small parts of irregular sizes split by EWQ), the PE may

have operands of many different scales or task IDs. Thus, the data must be splited to parts, leading to low utilization of this PE. One simple improvement is to allow each adder node in the adder tree to give the result. But this way still has its limitation, as the operands must be placed aligned according to the number of them. For example, there is a 16-input adder tree, and 4 dot product operations of size 3. The operands are allocated to the multipliers whose indexes are (0,1,2), (4,5,6), (8,9,10), (12,13,14) respectively. In this case, the utilization of this PE is still only 75%.

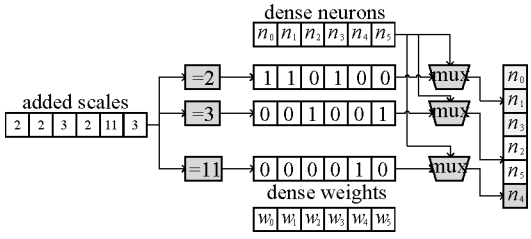

Fig. 5 Data dispatching unit

To maximize the utilization of the PE, the forward adder network (FAN) proposed by SIGMA[19] is adopted. It settles the drawback of low utilization caused by irregular size of dot-product operations by connecting low-level adders to high-level adders. As shown in Fig. 6, each node in FAN is not only an adder which performs addition operation, but also a switch which bypasses the two inputs. The routing algorithm will decide whether a node is an adder or a switch, and which result is to be selected by high-level nodes for addition.

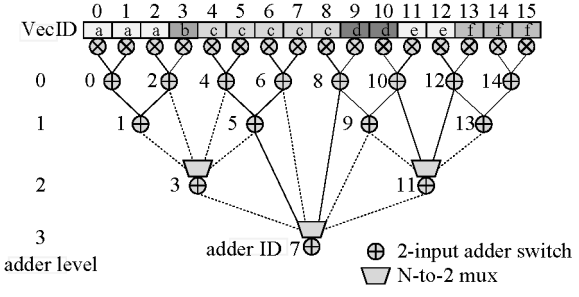

Fig. 6 Forward adder network[19]

ACCU is used to aggregate the results of all PEs of this PE line, and accumulates them by task IDs. The outputs of each PE may contain partial sums of the same task but with different scales, so they are first requantized and then added, to make them belong to different tasks. The parts of the same task may also be dispatched to different PEs, so previous results of all PEs are requantized and added again, to get the final sum of each task. The results are then sent to DQU.

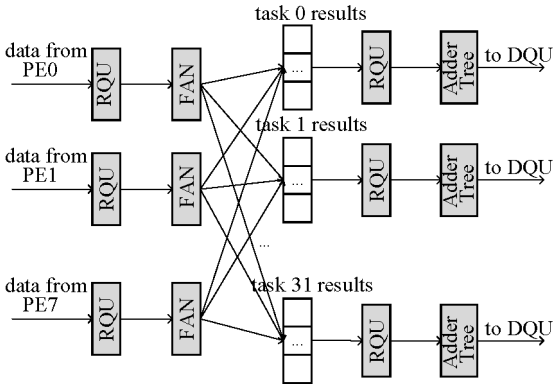

Fig. 7 Accumulation unit

4 Experimental methodology

In this section, the experimental setup is listed, including the benchmarks and the configurations of hardware platforms for comparison.

4.1 Benchmarks

The benchmarks include several CNNs listed in Table 1. The datasets are all TinyImageNet which is a subset of ImageNet.

Table 1 Benchmarks

Model	Batch size	Learning rate
AlexNet[22]	16	0.08
ResNet-18[23]	16	0.08
SqueezeNet[24]	16	0.60
MobileNet-V2[25]	16	0.50

4.2 Hardware configurations

The training performance and hardware cost of Cambricon-QR with TPU and Cambricon-Q are compared.

The performance of TPU is obtained by a simulator based on SCALE-sim[26]. It is modified to make TPU have the same configurations with Cambricon-Q and Cambricon-QR, i.e., 2 Tops @ Int4, 256 kB NBin, 512 kB SB and 256 kB NBout, and 17.06 GB/s memory bandwidth. The quantization technique is consistent with Cambricon-Q, i.e., the dynamic statistic quantization.

Cambricon-Q has 64 × 64 4-bit PE array working at 1 GHz, providing a peak performance of 8 Tops @ Int4 or 2 Tops @ Int8. The performance is obtained by the simulator.

A simple implementation of Cambricon-QR is accomplished in Verilog register transfer level (RT)L. It is synthesized, placed and routed under TSMC 45 nm

technology to get the area and power. The performance is still obtained by the simulator. Cambricon-QR has a PE array working at 1 GHz which has 8 PE lines, and each PE line contains 8 PEs, each of which includes 16 8-bit multipliers, thus providing a peak performance of 2 Tops @ Int8, consistent with Cambricon-Q. The units which deal with sparsity are designed to support at most 50% sparsity. Besides, it also has two index buffers to place neuron group indexes and synapse group indexes, i. e. , 128 kB NIB and 256 kB SIB.

5 Experimental results

In this section, the power and area of Cambricon-QR are evaluated. Its performance is also compared against graphics processing unit (GPU), TPU and Cambricon-Q.

5.1 Hardware characteristics

Table 2 lists the hardware characteristics (area and power of each unit, and their proportions in total area and power) of Cambricon-QR. The acceleration core of Cambricon-QR occupies 16.10 mm^2 area and consumes 964.80 mW. Compared with Cambricon-Q, Cambricon-QR has much larger area and a little higher power consumption, which is mainly attributed to the units which deal with sparsity (DSU, DDU, ACCU). Besides, the extra buffers for indexes also contribute a little to area.

Table 2 Hardware characteristics

	Area /mm^2	Proportion /%	Power /mW	Proportion /%
Acceleration Core	16.10	100.00	964.80	100.00
SFU	2.11	13.11	483.88	50.15
NBin	0.72	4.47	4.43	0.46
NIBin	0.54	3.35	3.32	0.34
SB	1.52	9.44	9.65	1.00
SIB	1.08	6.71	6.65	0.69
NBout	0.72	4.47	4.43	0.46
DSU	2.81	17.46	122.82	12.73
DDU	1.72	10.68	83.80	8.69
FAN	2.01	12.50	138.44	14.35
ACCU	2.78	17.27	101.26	10.50
DQU	0.09	0.54	6.12	0.64
NDP engine	1.50	100.00	113.84	100.00
QU	1.43	95.34	97.57	85.71
NDPO	0.07	4.66	16.27	14.29

5.2 Performance

Table 3 lists the training accuracy of EWQ against full-precision training. Various configurations of EWQ are tested, of which n is the number of quantization groups and w is the bit-widths of fixed data. EWQ achieves $\leqslant 1\%$ accuracy loss, and performs even better on AlexNet and ResNet-18. Only for $n = 16$, $w = 8$, EWQ suffers obvious accuracy loss, which may be attributed to fewer parameters. Though MobileNet-V2 is also a light-weight neural network, its activation function is ReLU6 while that of SqueezeNet is ReLU. The former leads to the greater robustness at low bit-width.

Table 3 Training accuracy results(%)

Model	FP32	$n=64$ $w=10$	$n=32$ $w=10$	$n=32$ $w=9$	$n=32$ $w=8$	$n=16$ $w=8$
AlexNet	31.29	31.27	31.55	31.41	31.50	31.32
ResNet-18	45.82	45.89	46.15	45.71	45.68	45.59
SqueezeNet	33.50	34.11	33.80	34.41	34.05	27.06
MobileNet-V2	29.51	28.58	29.16	29.03	28.63	28.46

Fig. 8 (a) shows the performance improvement of Cambricon-QR against TPU and Cambricon-Q. The average speedup of Cambricon-QR on four neural networks is 3.54 × against TPU and 1.61 × against Cambricon-Q. The performance is mainly due to the sparsity. Table 4 lists the neuron and weight sparsity of these four neural networks during forward and backward pass (pruning on weights is not performed, so the weight sparsity is rather low). Though AlexNet is of high sparsity, the speedup is still only 1.98 × because Cambricon-QR is designed to support at most 50% sparsity. MobileNet-V2 has only 15.9% average sparsity but its speedup against Cambricon-Q is 1.44 ×, which is due to the support of small convolutions of Cambricon-QR.

Table 4 Sparsity of neural networks on TinyImageNet

Model	Fwd. neuron	Fwd. weight	Bwd. neuron	Bwd. weight	Average
AlexNet	0.617	1.21e-6	0.903	1.23e-6	0.802
ResNet-18	0.490	1.07e-6	0.454	1.08e-6	0.466
SqueezeNet	0.455	4.51e-7	0.275	4.61e-7	0.336
MobileNet-V2	0.229	1.34e-6	0.124	1.34e-6	0.159

As is done in Cambricon-Q, the training process is broken down into six parts, which include forward pass (FW), backward pass (computing gradients on

neurons (NG), computing gradients on weights (WG)), statistical analysis (S) and quantization (Q). The results are shown in Fig. 8 (b).

5.3 Energy

Fig. 8 (c) shows the energy comparison of Cambricon-QR against TPU and Cambricon-Q. Cambricon-QR achieves 1.69 × and 0.90 × energy efficiency against TPU and Cambricon-Q. Fig. 8 (d) shows the energy breakdown of each module, including acceleration core (ACC), on-chip buffer (BUF), memory standby (DDR-SB) and memory dynamic (DDR-DY). Energy efficiency of Cambricon-QR differs from that of Cambricon-Q in two parts. One part is that the acceleration core needs to get quantization indexes from the quantization unit in NDP engine, which causes extra data transferring on DDR bus, and this is the same as the on-chip buffers. The other part is that the acceleration core of Cambricon-QR consumes less power than Cambricon-Q due to the addressing of sparsity,

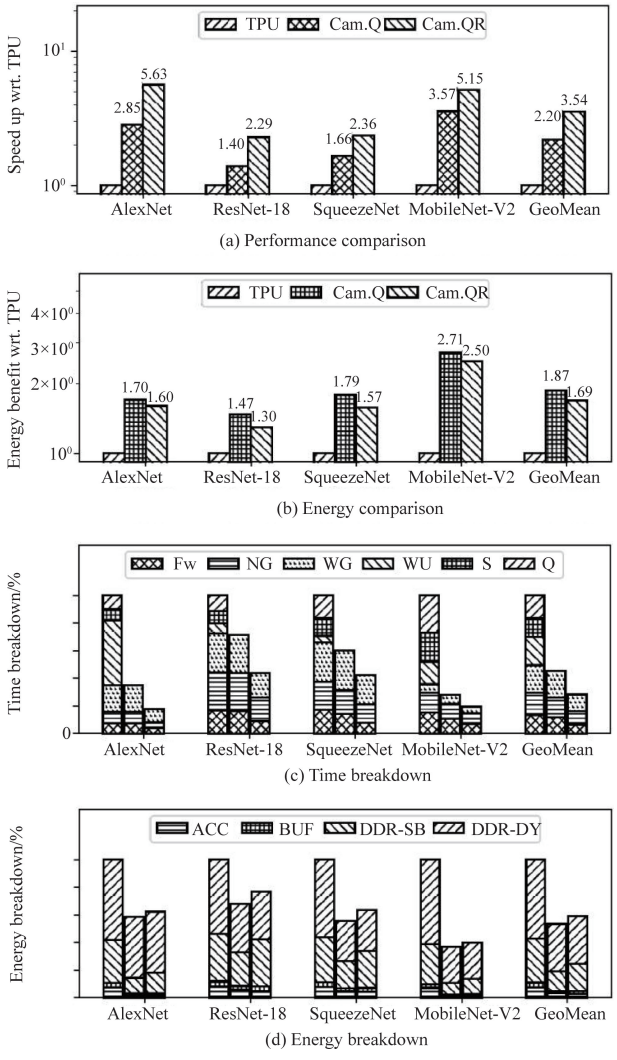

Fig. 8 Performance/energy comparison and breakdown

but this cannot completely compensate for the increased power of DDR bus and on-chip buffers.

5.4 Discussion

As mentioned in Section 2, the proposed quantization technique can utilize different data distributions of neural networks by using quantization prefix codes of different lengths, in which case, the grouping is non-uniform. The training accuracy is tested on ResNet-18 and SqueezeNet whose training accuracy drops rapidly at low bit-width. As shown in Table 5, while uniform grouping (quantization prefix codes are of the same length) causes severe accuracy loss at low bit-width, non-uniform grouping still maintains the accuracy comparable with full-precision training. Notice that when $w = 4$, the accuracy of non-uniform grouping is even higher than FP32 a lot on SqueezeNet, a reasonable explanation is that quantization can be regarded as a regularization method[27] to prevent overfitting.

Table 5 Training accuracy with variable data ranges

	Model	ResNet-18	SqueezeNet
	FP32	45.82	33.50
$w = 6$	uniform	45.28	27.47
	non-uniform	/	34.85
$w = 4$	uniform	37.27	22.31
	non-uniform	43.69	36.85

Considering the area of DSU which now takes 17.46% area of acceleration core, Cambricon-QR only support 50% sparsity currently. If aiming to leverage more sparsity, the number of inputs fed to DSU will be larger and the area of DSU will increase a lot. Optimizing the design and reducing the area to support higher sparsity will be the future work.

6 Conclusion

In this work, an algorithmic-deterministic quantization technique is proposed to guarantee the reproducibility of quantization errors with negligible accuracy loss. The acceleration core of Cambricon-Q is also redesigned as Cambricon-QR to support sparse neural networks. It fits well for small convolutions. Experiment results show that Cambricon-QR outperforms Cambricon-Q by 1.61 × on average with about 10% energy increase.

Reference

[1] ZHANG X, LIU S, ZHANG R, et al. Fixed-point back-propagation training [C]//Proceedings of the IEEE/CVF

Conference on Computer Vision and Pattern Recognition. Seattle: IEEE,2020: 2330-2338.
[2] LEE J, LEE J, HAN D, et al. 7.7 LNPU: a 25.3 TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16[C]//2019 IEEE International Solid-State Circuits Conference. San Francisco: IEEE,2019: 142-144.
[3] NASCIMENTO M G, PRISACARIU V A, FAWCETT R, et al. Hyperblock floating point: generalised quantization scheme for gradient and inference computation[C]//Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. Waikoloa: IEEE, 2023: 6364-6373.
[4] LIU C, ZHANG X, ZHANG R, et al. Rethinking the importance of quantization bias, toward full low-bit training[J]. IEEE Transactions on Image Processing, 2022, 31: 7006-7019.
[5] ZHU F, GONG R, YU F, et al. Towards unified int8 training for convolutional neural network[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Seattle: IEEE,2020: 1969-1979.
[6] ZHAO K, HUANG S, PAN P, et al. Distribution adaptive int8 quantization for training CNNs[C]//Proceedings of the AAAI Conference on Artificial Intelligence. Virtual: AAAI Press, 2021: 3483-3491.
[7] ZHAO Y, LIU C, DU Z, et al. Cambricon-Q: a hybrid architecture for efficient training[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2021: 706-719.
[8] JACOB B, KLIGYS S, CHEN B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Salt Lake City: IEEE, 2018: 2704-2713.
[9] NAGEL M, BAALEN M, BLANKEVOORT T, et al. Data-free quantization through weight equalization and bias correction[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. Seoul: IEEE, 2019: 1325-1334.
[10] BANNER R, NAHSHAN Y, SOUDRY D. Post training 4-bit quantization of convolutional networks for rapid-deployment[C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems. Vancouver: NIPS, 2019:7950-7958.
[11] FANG J, SHAFIEE A, ABDEL-AZIZ H, et al. Post-training piecewise linear quantization for deep neural networks[C]//Computer Vision – ECCV 2020: 16th European Conference. Glasgow: Springer, 2020: 69-86.
[12] ZHOU S, WU Y, NI Z, et al. Dorefa-net: training low bitwidth convolutional neural networks with low bit width gradients[EB/OL]. (2016-06-20)[2023-04-26]. https://arxiv.org/pdf/1606.06160.pdf.
[13] WU S, LI G, CHEN F, et al. Training and inference with integers in deep neural networks[EB/OL]. (2018-02-13)[2023-04-26]. https://arxiv.org/pdf/1802.04680.pdf.
[14] YANG Y, DENG L, WU S, et al. Training high-performance and large-scale deep neural networks with full 8-bit integers[J]. Neural Networks, 2020, 125: 70-82.
[15] WANG N, CHOI J, BRAND D, et al. Training deep neural networks with 8-bit floating point numbers[C]//Proceedings of the 32nd International Conference on Neural Information Processing Systems. Montréal: NIPS, 2018: 7686-7695.
[16] ZHANG S, DU Z, ZHANG L, et al. Cambricon-X: an accelerator for sparse neural networks[C]//2016 49th Annual IEEE/ACM International Symposium on Microarchitecture. Taibei: IEEE, 2016: 1-12.
[17] HAN S, LIU X, MAO H, et al. EIE: efficient inference engine on compressed deep neural network[C]//2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture. Seoul: IEEE, 2016: 243-254.
[18] ZHOU X, DU Z, GUO Q, et al. Cambricon-S: addressing irregularity in sparse neural networks through a cooperative software/hardware approach[C]//2018 51st Annual IEEE/ACM International Symposium on Microarchitecture. Fukuoka: IEEE, 2018: 15-28.
[19] QIN E, SAMAJDAR A, KWON H, et al. SIGMA: a sparse and irregular GEMM accelerator with flexible interconnects for DNN training[C]//2020 IEEE International Symposium on High Performance Computer Architecture. San Diego: IEEE, 2020: 58-70.
[20] HOWARD A G, ZHU M, CHEN B, et al. MobileNets: efficient convolutional neural networks for mobile vision applications[EB/OL]. (2017-04-17)[2023-04-26]. https://arxiv.org/pdf/1704.04861.pdf.
[21] CHOLLET F. Xception: deep learning with depthwise separable convolutions[C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Honolulu: IEEE, 2017: 1251-1258.
[22] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.
[23] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 770-778.
[24] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size[EB/OL]. (2016-02-24)[2023-04-26]. https://arxiv.org/pdf/1602.07360.pdf.
[25] SANDLER M, HOWARD A, ZHU M, et al. MobileNetv2: inverted residuals and linear bottlenecks[C]//Proceedings of the IEEE Conference On Computer Vision And Pattern Recognition. Salt Lake City: IEEE, 2018: 4510-4520.
[26] SAMAJDAR A, ZHU Y, WHATMOUGH P, et al. Scale-SIM: systolic CNN accelerator simulator[EB/OL]. (2018-10-16)[2023-04-26]. https://arxiv.org/PDF/1811.02883.pdf.
[27] HUNG P H, LEE C H, YANG S W, et al. Bridge deep learning to the physical world: an efficient method to quantize network[C]//2015 IEEE Workshop on Signal Processing Systems. Hangzhou: IEEE, 2015: 1-6.

LI Nan, born in 1998. He is studying for M. S. degree in University of Science and Technology of China. He received his B. S. degree from University of Science and Technology of China in 2020. His researches focus on neural networks and accelerators.