Cambricon-C: Efficient 4-bit Matrix Unit via Primitivization

Yi Chen
USTC SKLP, ICT, CAS
chen_yi@mail.ustc.edu.cn

Yongwei Zhao
SKLP, ICT, CAS
zhaoyongwei@ict.ac.cn

Yifan Hao
SKLP, ICT, CAS
haoyifan@ict.ac.cn

Yuanbo Wen
SKLP, ICT, CAS
wenyuanbo@ict.ac.cn

Yuntao Dai
USTC
daiyuntao@mail.ustc.edu.cn

Xiaqing Li
SKLP, ICT, CAS
lixiaqing@ict.ac.cn

Yang Liu
SKLP, ICT, CAS UCAS
liuyang22z1@ict.ac.cn

Rui Zhang
SKLP, ICT, CAS
zhangrui@ict.ac.cn

Mo Zou
SKLP, ICT, CAS
zoumo@ict.ac.cn

Xinkai Song
SKLP, ICT, CAS
songxinkai@ict.ac.cn

Xing Hu
SKLP, ICT, CAS SHIC
huxing@ict.ac.cn

Zidong Du
SKLP, ICT, CAS SHIC
duzidong@ict.ac.cn

Huaping Chen
USTC
hpchen@ustc.edu.cn

Qi Guo*
SKLP, ICT, CAS
guoqi@ict.ac.cn

Tianshi Chen
Cambricon Technologies
tchen@cambricon.com

***Abstract*—Deep learning trends to use low precision numeral formats to cope with the ever-growing model sizes. For example, the large language model LLaMA2 has been widely deployed in 4-bit precision. With larger models and fewer unique values caused by low precision, an increasing proportion of arithmetic in matrix multiplication is repeating. Although discussed in prior works, such value redundancy has not been fully exploited, and the cost to leverage the value redundancy often offsets any advantages. In this paper, we propose to *primitivize* the matrix multiplication, that is decomposing it down to the 1-ary successor function (a.k.a. counting) to merge repeating arithmetic. We revisited various techniques to propose Cambricon-C SA, a 4-bit primitive matrix multiplication unit that doubles the energy efficiency over conventional systolic arrays. Experimental results show that Cambricon-C SA can achieve 1.95× energy efficiency improvement compared with MAC-based systolic array.**

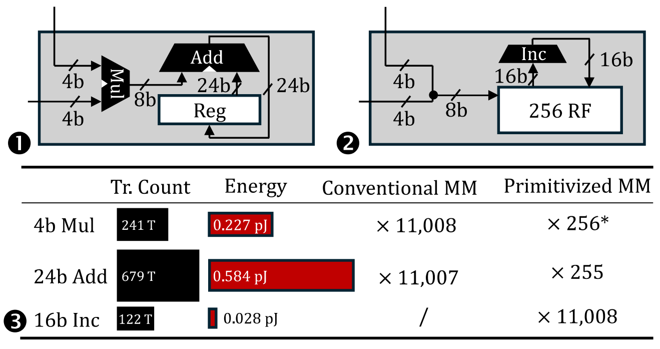
***Index Terms*—AI accelerator, deep learning, large language model, model quantization.**

I. Introduction

In recent years, increasingly large deep learning models have shown superior performance on various tasks [3], [32], [38]. These large models are first quantized before widely deployment to save the ever-growing computing costs. For example, the large language model LLaMA-2 family includes models with parameters from 7 billion to 70 billion, that have been quantized to 4-bit with almost no performance degradation [12]. As 4-bit quantization has been adopted as the go-to solution for many popular models [9], [10], [18], [37], [38], efficient 4-bit matrix multiplication is becoming the core computation in accelerators these days.

Increasingly large models and low-bitwidth quantized data together bring new opportunities to optimize matrix multiplications. It is based on the observation that in these models, most arithmetic operations involved in matrix multiplication are duplicated. In LLaMA2-7B (4-bit), single output values in specific matrix multiplication operators are summed over

*Corresponding Author



	Tr. Count	Energy	Conventional MM	Primitivized MM
4b Mul	241 T	0.227 pJ	× 11,008	× 256*
24b Add	679 T	0.584 pJ	× 11,007	× 255
❸ 16b Inc	122 T	0.028 pJ	/	× 11,008

(∗ required in the last step multiplying $\boldsymbol{m}^{\mathrm{T}}$, 8b×16b to be precise.)

Fig. 1. Concept of Primitivized Matrix Multiplication (PMM). ❶ MAC involves 4b multiplier and 24b adder, while ❷ PMM reduce the arithmetic to 16b increment. ❸ lists the transistor count, energy consumption, and the number of arithmetic operations, where 11008 is the matrix dimension in the FFN layer in LLaMA2-7B.

11,008 4-bit-by-4-bit products, while 4-bit data formats can only represent $2^4 = 16$ unique values. As a fast corollary, no more than $16 \times 16 = 256$ out of $\sim 10^4$ multiplications are unique, and all others are duplicated. This reveals a significant source of optimization.

To leverage the duplicated arithmetic operations, we propose *primitivized matrix multiplication (PMM)*, a novel algorithm for low-bitwidth matrix multiplication. There are two key differences between PMM and the conventional algorithm.

The first key difference is avoiding duplicated multiplications by pre-computing a Look-up Table. Note that the output values in matrix multiplication are obtained by computing inner-products. For computing the output value $y = \sum_{i=0}^{N-1} w_i x_i$, $N \gg 256$, productions for all possible values in w and x can be pre-computed and written as a vector $\boldsymbol{m}$. For 4-bit integer represented in two's complement, $\boldsymbol{m}$ should be a 256-dimension vector

$$\boldsymbol{m} = [\begin{array}{lllll} 0\times 0, & 0\times 1, & 0\times 2, & \dots, & 0\times -1, \\ 1\times 0, & 1\times 1, & 1\times 2, & \dots, & 1\times -1, \\ 2\times 0, & 2\times 1, & 2\times 2, & \dots, & 2\times -1, \\ \vdots & \vdots & \vdots & & \vdots \\ -1\times 0, & -1\times 1, & -1\times 2, & \dots, & -1\times -1 \end{array}], \tag{1}$$

storing all the products between {-8,-7,...,-1,0,1,...7} and {-8,-7,...,-1,0,1,..7}. Then, the computation of y can be conducted by looking up pre-computed products from $\boldsymbol{m}$ instead of performing multiplication for each term, i.e., $y = \sum_{i=0}^{N-1} \boldsymbol{p}_i \boldsymbol{m}^{\mathrm{T}}$, where $\boldsymbol{p}_i$ is the one-hot vector which encoding (x_i, w_i), i.e.

$$\boldsymbol{p}_i = [p_0, p_1, \dots, p_{255}], \quad p_j = \begin{cases} 1, & \text{if } j = x_i \circ w_i \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The second key difference is to reduce the strength of additions. Since vector multiplication is distributive over addition, we compute $\sum \boldsymbol{p}_i$ first, and only multiply $\sum \boldsymbol{p}_i$ by $\boldsymbol{m}^{\mathrm{T}}$ once in the last step, i.e., $y = \left(\sum_{i=0}^{N-1} \boldsymbol{p}_i\right) \boldsymbol{m}^{\mathrm{T}}$. The major part of computation $\sum \boldsymbol{p}_i$ is summing one-hot vectors. Therefore, *additions* of binary numbers are reduced to *increments* to one of 256 numbers. Increment, also known as *1-ary successor function*, is the primitive arithmetic operation defining Peano axioms, and is even lighter to implement than addition. For example, adding two binary numbers requires full adders, whereas half adders are sufficient for incrementing binary numbers. As shown in Figure 1, it could be implemented by 256 counters with very sparse circuit activities (alternatively, very low switching power).

Unfortunately, readers may quickly find out that simple hardware implementation of PMM is not practical. A straitforward PMM implementation requires 256 counters per PE, which cost $\sim 45\times$ area of a MAC unit seen in conventional design. The excessive area causes increased leakage, canceling the superiority on the computational efficiency brought by PMM.

To bring PMM a practical implementation, we revisited several key techniques, and came up with Cambricon-C, an efficient 4-bit matrix unit. Cambricon-C reduced the number of counters from 256 to 29 per PE by using *quarter square multiplication*, a technique dated back to 1817 in literature [14]. Cambricon-C also explored and selected the most energy-efficient counter circuits. Designed with these key techniques, Cambricon-C nearly doubled the energy efficiency compared to the conventional 4-bit MAC-based systolic array.

We list our contributions as follows:

1) We propose the *primitivized matrix multiplication (PMM)* algorithm, which reduces stronger arithmetic operations (multiplications and additions) to primitive operations (increments) for low-bitwidth matrix multiplication.
2) We revisited *quarter square multiplication*, which significantly reduced the number of counters required in the design of a hardware implementing PMM.
3) We explored the design of energy-efficient counters, and compared various designs based on synthesized circuits.
4) We propose Cambricon-C SA, an efficient 4-bit matrix unit via PMM. The results show that Cambricon-C SA can achieve up to 1.95× energy efficiency over the conventional MAC-based systolic array. We also evaluated the complete accelerator Cambricon-C ACC, which achieves 1.13-1.25× overall improvement of energy efficiency when compared with TPU over various DNN workloads.

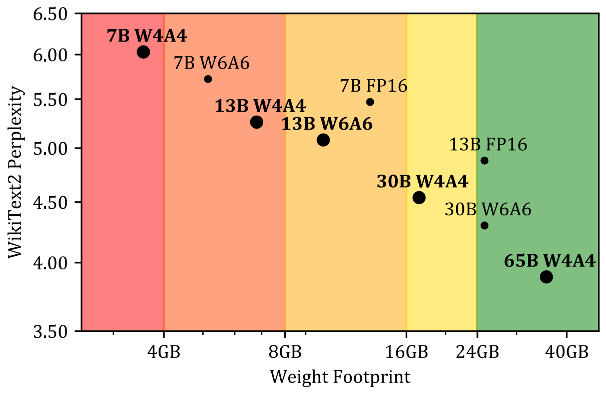


Fig. 2. Language modeling performance (perplexity, lower is better) versus memory footprint on LLaMA family. W4A4/W6A6 denotes all numbers are quantized to 4-bit/6-bit. 4-bit quantized variants of LLaMA2-7B, LLaMA2-13B, LLaMA-30B and LLaMA-65B are most performant under 4 GB, 8 GB, 24 GB and 40 GB limits, respectively [27], [41].

II. Background and Motivation

A. Emerging 4-bit Large Models

Deep learning has become instrumental across various domains, including computer vision, natural language processing (NLP), and speech recognition. A significant milestone in the wide adoption of deep learning was achieved with AlexNet [24], which, with 60 million trainable parameters, outperformed all competitors in the 2012 ImageNet competition. This breakthrough was followed by VGGNet [37], scaling up to 138 million parameters and further demonstrating that larger models could achieve higher performance levels.

The scale of models has been closely linked to their performance capabilities [20]. This trend is exemplified by the evolution from earlier recurrent networks [7] (15M parameters) to the more expansive models like Transformers [39] (65M parameters) and BERT [9] (up to 340M), and to the modern large models like LLaMA2 [38] (up to 70B) and GPT-4 [30] (size not disclosed). With parameter counts moving from millions to hundreds of billions, modern large models have set new benchmarks in natural language processing.

However, their size also imposes significant demands on the computational resources required for inference on various devices. The continuous growth in model scale and the limitations of hardware resources have necessitated the adoption of quantization techniques. As large models with billions of parameters are exceeding the memory capacity of the most powerful consumer-grade hardware (e.g. LLaMA2-13B FP16

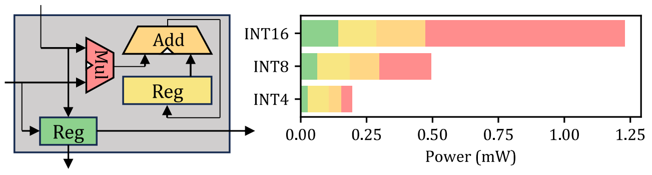


Fig. 3. Power breakdown on a minimalist MAC-based systolic array processing element.

does not fit in a GeForce 4090), their quantized variants as low as 4-bit have never been more widely deployed. Figure 2 plots the language modeling performance (perplexity on WikiText2 dataset, lower is better) and memory footprint to load weights for LLaMA family models. It shows that quantized larger models are often more performant than smaller models with higher precision. For devices with 4 GB, 8 GB, 24 GB or 40 GB memory budgets, 4-bit variants are the best models that fit in the limit. To meet these demands, popular inference servers including GGML [13], vLLM [26], TensorRT-LLM [29] and OpenVINO [31] are supporting 4-bit quantization out-of-the-box. These emerging practices emphasize the efficiency of 4-bit matrix multiplication algorithms and hardware.

B. 4-bit Matrix Units

Systolic array (SA) based on multiply-accumulate units (MAC) has been considered well-understood for its 45 years of history [25], and has been recognized as the paragon micro-architecture specialized for matrix multiplication. To place more MAC units into SA under the same budget, quantization is introduced. Compared to FP32, INT8 MAC-based SA can bring an order-of-magnitude reduction in energy and area [33] and the precision is sufficient for most inference workloads.

However, with lower precision, the efficiency return is diminishing. Figure 3 shows a minimalist implementation of MAC-based SA PE and evaluates its power at different precision. From INT16 to INT8 and from INT8 to INT4, the boolean complexity of the function of the PE is expected to reduce by a factor of 3.3 $\sim$ 3.6, but the power is only reduced by a factor of 2.5. This means that *whenever the quantized bit widths are halved, a 27.5% tax will be charged for MAC-based SA*. When it comes to INT4, the actual computational efficiency is no longer excellent.

The reason is two-fold. First, the quantization effect is showing, and becoming non-negligible at INT4. INT4 can only represent 16 unique values, thus there are no more than 256 unique cases in INT4 multiplication. However, the matrix dimensions processed by SA are often far above 256. This means that most arithmetics are repeating themselves, causing wasted energy.

Second, the power of the accumulator is dominating. Quantization reduces the width of the multiplier, but the width of the accumulator is limited by the max matrix dimensions to support. The accumulator must be guaranteed not to overflow on the largest matrix. Assuming the multiplication of n-order square matrices, the accumulator must have $\log_2 n$ more bits than the product. For INT4, the accumulator requires 24 bits or even higher. Compared with the multiplier (double 4-bit in, 8 bits out), the power consumption of the accumulator (24 bits saved, 8 bits in) is 3.07 times of it. As shown in Figure 3, as the bit width decreases, the power consumption for multiplication decreases rapidly, while the power consumption for addition and registration decreases slowly and becomes dominating. Without improving the efficiency of accumulation, the returns brought by quantization must diminish.

There are several proposals as alternatives to MAC-based SA, in which some have acknowledged the above-mentioned problems. For example, *Stripes* [23] adopted bit-serial computing scheme and removed the need for multipliers, thus avoiding the wasted energy on redundant multiplications. However, it also suffers from increased accumulation. There are following works on the bit serial computing scheme to further exploit potential computational redundancy in matrix multiplication, including Pragmatic [1], Bit-Tactical [8], Loom [36], Laconic [35], etc. There are also works addressing the repetitive arithmetic by lookup, such as UCNN [19], FuseKNA [40], Cambricon-P [17], and CARAT [34]. However, none of these works are targeting 4-bit or lower. On INT4, although the repetition is abundant, INT4 arithmetic is too small such that there is not enough workload to hide overheads brought by complicated architectures. Excluding Stripes which is simple enough, these proposals are hard to achieve net improvements on INT4.

In summary, we need to develop a new technique for 4-bit matrix multiplication. It should save the repetitive arithmetic and reduce the strength of accumulation. The algorithm must be simple and neat, such that not incur too much architectural overhead to implement. In the next section, we propose the algorithm of *Primitivized Matrix Multiplication (PMM).*

III. Algorithm

A. Primitivized Matrix Multiplication

We have formally introduced *Primitivized Matrix Multiplication (PMM)* in Section I. Here we demonstrate the computational process of PMM with concrete examples.

Figure 4 shows a naive implementation of previously defined PMM, and gives an example matrix multiplication to perform. The shown PMM matrix unit is organized as an output-stationary SA, in which each PE undertakes an inner-product and produces a complete output value without interruption (Figure 4 ❶). Inside each PE, there is a counter array including 256 counters to compute $\sum \boldsymbol{p}_i$ by increments (Equation 2).

We take the top-left PE as an example. In this example, this PE receives row-vector $[1, 2, -2, 2]$ and column-vector $[-1, 1, 0, 1]^{\mathrm{T}}$. The inner-product to produce should be 3. As encoded in two's complements, -2 is represented as E (14) and -1 is represented as F (15). The elements in both input vectors are systolically sent to this PE.

At the first cycle, this PE receives 1 and F (Figure 4 ❷). The PE selects and activates the counter corresponding to $(1, F)$ and increment it, to record an occurrence of the term $1 \times F$. The value in this counter is increased from 0 to 1. In

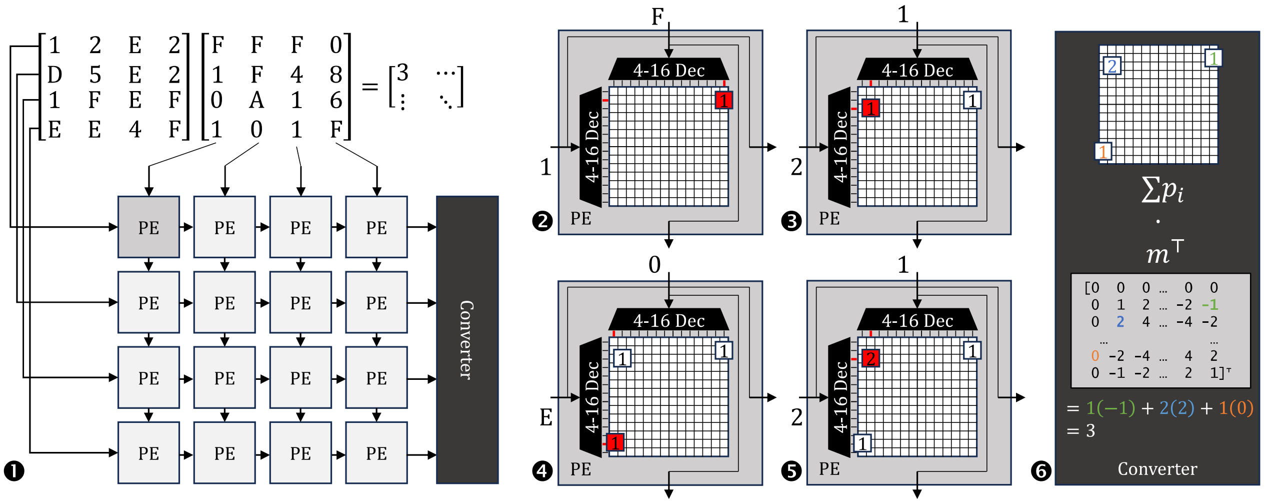


Fig. 4. A 256-counter naive PMM implementation based on ❶ an output-stationary systolic array. Each PE adopts a 16 × 16 counter array to compute $\sum \boldsymbol{p}_i$ (Equation 2) by increments. ❷-❺ Showing the top-left PE for example. Each pair of input and weight select the corresponding counter to increase. ❻ Converter read $\sum \boldsymbol{p}_i$ out from PE, multiply by $\boldsymbol{m}^{\mathrm{T}}$ (Equation 1) to get final result.

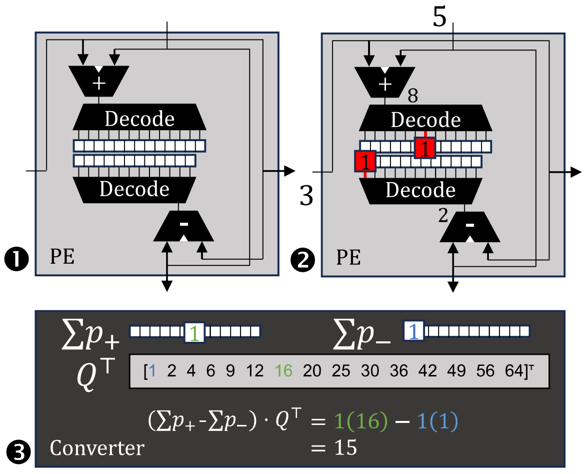
the following three cycles, three more terms arrived, and the values in the corresponding counters are increased accordingly (Figure 4 ❸❹❺). Although each PE include many counters, there is only one counter activated per cycle. The power consumption of a counter is significantly lower than a MAC unit. As a result, the switching power is still much lower than the conventional MAC-based PE.

After the systolic processing of the input matrices, the PEs get $\sum \boldsymbol{p}_i$. There is a last step to multiply by $\boldsymbol{m}^{\mathrm{T}}$ (Equation 1) to get the final result. This step is an inner-product between fixed length vectors ($\sum \boldsymbol{p}_i$ and $\boldsymbol{m}^{\mathrm{T}}$, 256 elements), and is completed in a dedicated Converter (Figure 4 ❻). Because the length is fixed and often much shorter than the dimension of original matrices under computation, the amortized cost of converting is negligible.

However, including too many counters will bring excessive area and leakage power costs. The naive PMM-based PE implementation in Figure 4 ❷ costs $\sim 45\times$ area of a conventional MAC-based PE, and almost half of the power is burnt by leakage. Therefore, we must find a technique to significantly reduce the number of counters used in the design, to achieve net improvement on energy efficiency.

B. Quarter Square Multiplication

Quarter square multiplication (QSM) is an old technique to compute products easier with the help of lookup tables. Literature records that QSM has been applied in 1817 when people rely on lookup tables and slide rules to perform arithmetic [14]. In 1970, this scheme was introduced to implement binary multipliers when transistors were expensive [6]. However, with the rapid development of semiconductor and computer technology, QSM is no longer in use today. In this paper, we revisit QSM to reduce the counters in PMM implements.



5
+
8
Decode
Decode
3
2
PE
Σp+
Σp−
Q^T
[1 2 4 6 9 12 16 20 25 30 36 42 49 56 64]^T
(Σp+-Σp−) · Q^T = 1(16) − 1(1)
= 15
Converter

Fig. 5. A 29-counter PMM implementation via QSM. ❶ Each PE include a 4-bit adder, a 4-bit subtractor, 15 up-counters and 14 down-counters. ❷ Example computing 3×5, select and increment up-counter $|3+5|$ and down-counter $|3-5|$. Note that counters are numbered starting from 2. ❸ Converter subtract values from up-counters by down-counters, and multiply by $\boldsymbol{Q}^{\mathrm{T}}$ (Equation 6) to get final result.

It is intuitively obvious that

$$\frac{(x+y)^2}{4} - \frac{(x-y)^2}{4} = xy. \tag{3}$$

When x and y are integers, xy must be integer as well. Additionally, $x+y$ and $x-y$ are either both even or both odd, such that the divisions in Equation 3 always generate canceling remainders. In other words, we have

$$(x+y)^2 \equiv (x-y)^2 \mod 4. \tag{4}$$

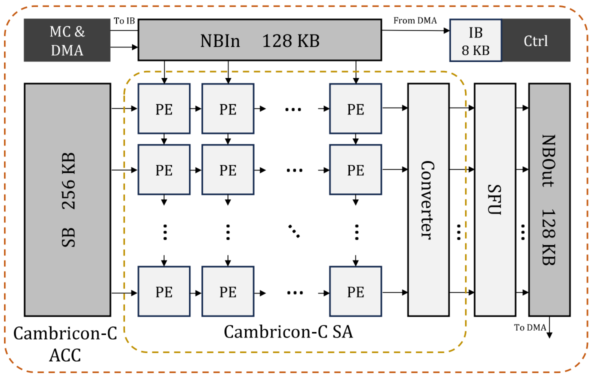


Fig. 6. Overall architecture of Cambricon-C ACC

As corollary,

$$\left\lfloor \frac{(x+y)^2}{4} \right\rfloor - \left\lfloor \frac{(x-y)^2}{4} \right\rfloor = xy, \qquad x, y \in \mathbb{Z}. \tag{5}$$

With a pre-computed lookup table of quarter squares $Q(n) = \lfloor n^2/4 \rfloor$, the product xy can be computed with an addition, a subtraction, and lookup $Q(n)$ twice.

The lookup table $Q(n)$ is much shorter than the naive $\boldsymbol{m}$ given in Equation 1. For the 4-bit integer represented in two's complement, $x+y$ lies in $[-16, 14]$ and $x-y$ lies in $[-15, 15]$. Due to the symmetry of $Q(n)$ about zero, $Q(n)$ for negative n equals $Q(-n)$. Moreover, since $Q(0) = Q(1) = 0$, the range can be further trimmed down to $[2, 16]$, 15 terms in total. Specifically, represented into a vector again, they are

$$\boldsymbol{Q} = [1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64]\,. \tag{6}$$

As shown in Figure 5, with the help of QSM, the required counters for implementing PMM reduced to 29 per PE, including 15 up-counters for $x+y$, and 14 down-counters for $x-y$. If provided, the 29 up/down counters can be merged into 15 bidirectional counters. This is due to the fact that $|x+y| = |x-y|$ implies $xy = 0$, such that the activated counters will not conflict on effectual data.

As there is still some overhead introduced with QSM, most noticeably of which is a pair of the 4-bit adder and subtractor, the best design needs to be explored. In the next section, we list various design candidates (with and without QSM) and compare them based on synthesized circuits.

IV. Architecture

In this section, we propose several candidate architectures for Cambricon-C, and compare them based on synthesized circuits. We first describe the overview. As counters play a significant role in PMM implementations, we discuss several ways to implement counters. At last, we explore and discuss the design candidates for the PEs and Converters.

A. Overview

Figure 6 illustrates the overall architecture of our proposal. First, we build a SA based on PMM, referred to as Cambricon-C SA. Same as previously described in Figure 4 ❶, Cambricon-C SA assumes the output-stationary systolic data flow. Different from conventional SA, Cambricon-C SA requires an additional converting step to get the final result. Therefore, there is a converter at the end of the readout data path. The design of the converter is dependent on the design of PE, which will be discussed later in this section.

Furthermore, we build a complete accelerator architecture for evaluation, referred to as Cambricon-C ACC. Cambricon-C ACC adopts a classic 3-buffer neural network accelerator framework from DianNao [5]. It has 128 KB input neuron buffer (NBIn), 128 KB output neuron buffer (NBOut), and 256 KB synapse buffer (SB). Buffers and main memory are connected with a direct memory access module (DMA). The main compute unit in Cambricon-C ACC is a Cambricon-C SA of 32×32 PEs. We will discuss the PE designs later in this section. Besides the Cambricon-C SA, there is also a vector special function unit (SFU) to perform activation, normalization, and quantization for the output data. The controller (Ctrl) orchestrates all modules in Cambricon-C ACC, and is accompanied with a 8 KB instruction buffer (IB).

B. Counter

Counters play a significant role in Cambricon-C as its power and latency decide the efficiency and performance of Cambricon-C SA. Hence it is essential to understand the design of counters before diving into the design of PEs. In this work, we have explored three types of counters, including *SRAM-based counters*, *ripple counters*, and *bidirectional counters*.

SRAM-based counters

The most important characteristic of PMM is its sparse activity. PMM requires numerous counters to store the values in $\sum \boldsymbol{p}_i$ (Equation 2), but there is only one (two if QSM enabled) of these counters acted per cycle. Intuitively, we consider efficient storage over incrementing. That is, using SRAM to store values, and using half adders to compute the increments.

Figure 7 ❶ shows the circuit design of SRAM-based counters. The SRAM is dual-port, one port for read and one port for write. Values are saved row-wise in the SRAM cell array. On incrementing, the read port wordline decoder in the SRAM peripheral translates the address to enable one of the read port wordlines. The corresponding row of cells is activated, and the value in the cells is read out through read port bitlines by sense amplifiers. Increment to the value is performed by ripple-carry half adders connected to the read port of SRAM. After the increment, the value is buffered in the register for a cycle, and starts to write back to the SRAM in the next cycle. The write port wordline decoder follows the read port, and activates the same row by enabling the write port wordline. The new value is written into the SRAM cells by write drivers. As the SRAM is dual-port, in the writing cycle, the counter can start reading the next value, achieving one incrementing operation per cycle. SRAM adopts a read-after-write strategy on port address conflicts.

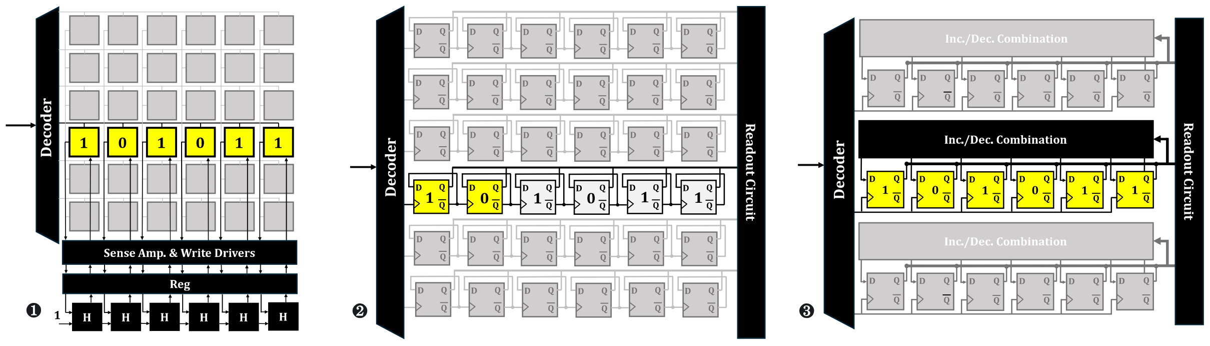


Fig. 7. The circuit design of different counters. ❶ SRAM counters; ❷ Ripple counters; ❸ Bidirectional counters;

The SRAM-based counter has the following pros and cons:

- **Pros:**
  1) Efficient storage for idle values. Using SRAM, values are stored in cells (latches) instead of flip-flops, such that both the area and the leakage current are under good control. Even though the naive PMM requires numerous counters, the static power of the SRAM-based counters is still lower than the MAC unit.
  2) Mainly implemented in highly optimized standard macro.
- **Cons:**
  1) Inefficient incrementing. Incrementing is performed outside of SRAM, so that the data path involved in incrementing is the longest among all discussed designs of counters. The register between SRAM and adders is also consuming a considerable portion of energy.
  2) Hard to trim unnecessary counters. For example in naive PMM implementation without QSM (Figure 4), 31 out of 256 counters are accumulating zeros. However, it is hard to customize SRAM to remove these ineffectual counters, which means to modify the wordline decoder and cell array structures.

Ripple counters

Due to various limitations of SRAM, we consider efficient incrementing as well.

Ripple counters as shown in Figure 7 ❷ are the simplest practice of unidirectional counters. Ripple counters are built with a series of 16 D flip-flops (Dff), each storing a bit of current value. The inverse data output $\overline{\mathrm{Q}}$ of each Dff is fed back to the data input D, so that on the positive edge of the clock signal, the stored bit is flipped. $\overline{\mathrm{Q}}$ is also connected to the clock of the next Dff, sending a positive clock edge to the next Dff on value transit from 1 to 0, effectively implementing *carry propagation*. On incrementing, feed a positive clock edge to the least-significant Dff in the ripple counter, the least-significant bit is flipped, and it ripples to the next Dff automatically until the carry propagation is stopped. Therefore, without full adders or half adders, ripple counters implemented incrementing purely based on Dff.

Ripple counter has the following pros and cons:

- **Pros:**
  1) Efficient incrementing. Unless the value is carrying to the high bits, the higher-significant flip-flops are not activated. In the sense of amortized costs, only two flip-flops are activated per cycle, resulting in dynamic power savings.
  2) High speed. Flip-flops in ripple counters are self-timed so that other parts in the circuit do not need to wait for the end of the ripple before moving to the next clock cycle. The latency is significantly shorter than other designs based on half adders.
  3) Easy to customize. As the decoder is fully customized, it can decode the address with various transformations, including trimming the ineffectual addresses, taking the absolute values (as required in QSM), etc.
- **Cons:**
  1) Larger area and leakage. Compared to SRAM-based counters, ripple counters store values in flip-flops, which are significantly larger than SRAM cells and suffer higher leakage currents.
  2) Unstable while incrementing. Ripple counters cannot be read until the carry propagation is complete, which lasts for an unknown time span. However, in PMM implementations the counters are write-only before read out for converting to the final result, thus this issue is not important.

Bidirectional counters

As QSM involves both up and down counting, we also explore bidirectional counters. With bidirectional counters, the implementation with QSM can combine the up and down counters, reducing the number of counters from 29 to 15. Intuitively, designs with QSM and bidirectional counters are potentially saving area and static power.

Figure 7 ❸ shows the circuit of bidirectional counters. They are basically registers accompanied with a combinational arithmetic unit, which is capable of incrementing and decrementing.

Bidirectional counter has the following pros and cons:

- **Pros:**
  1) Combined up/down counters required in QSM, may lead to the smaller design.
  2) Same as ripple counter, it is easy to customize.
- **Cons:**

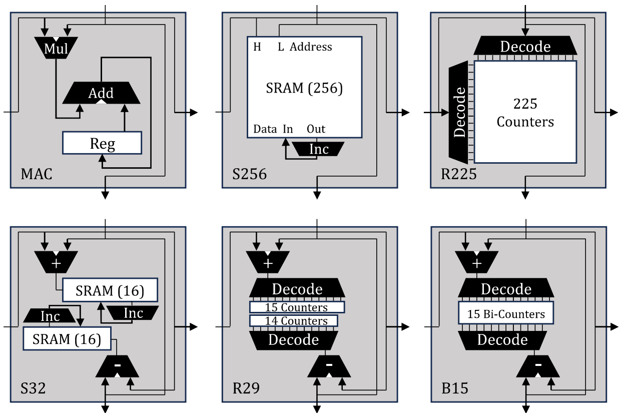


Fig. 8. Six different PE designs referred to as MAC, S256, R225, S32, R29, and B15. Readout circuits and systolic registers are omitted for simplicity.

1) Larger area and leakage. The added combinational circuit is causing increased area and static power.
2) Less efficient than ripple counter. Unlike ripple counters that only activate the Dffs on demand, a bidirectional counter must activate all bits in a value for incrementing or decrementing. This causes increased dynamic power. For example, the value in the bidirectional counter may oscillate around $-1 \sim 0$, resulting in the flip-flops flipping between $000\ldots00$ and $111\ldots11$ back and forth.

With these different choices for counters, we came up with several design candidates for Cambricon-C SA. Each design has a different architecture for PE and Converter. We dive into these architectures to make a comparison in the next subsection.

C. PE Designs

Figure 8 illustrates six designs of PE, including MAC, S256, R225, S32, R29, and B15. The naming of PEs is derived from the initial letter of the counter type and the number of counters within the PE. Table I summarizes the comparison of each PE design.

MAC is the well-studied design for conventional systolic array, without PMM. Because Cambricon-C SA is assuming output-stationary data flow, we removed redundant components from MAC, making it specific to the output-stationary data flow. In MAC, the accumulate register is set to 24 bits to guarantee no overflow for matrices up to 65k dimensions, equivalently to 16-bit counters in PMM implementations. The netlist-level simulation result shows that MAC can run at 357 MHz clock frequency, and the area is 1422 um^2. The energy consumption per operation of each MAC PE is 2.508 pJ.

S256 is the design with PMM and SRAM-based counters but without QSM. S256 uses 256 SRAM-based counters. Although there are 31 out of 256 terms in $\boldsymbol{m}$ zero, leaving these counters in the design is still most beneficial, because the decoder in SRAM is hard to customize. In S256, two 4-bit input values are concatenated into the 8-bit address, which is directly addressing the SRAM. The SRAM can run at 970 MHz, costing 1.365 pJ access energy and 21138 um^2 area. A complete S256 PE costs 21480 um^2 and 3.367 pJ. Because the number of counters is too large, S256 is suboptimal in these design candidates.

R225 is the design with PMM and ripple counters, but without QSM. R225 uses 225 ripple counters, removing the 31 zero terms where either side of inputs is zero. The decoder in R225 is designed in two 4-bit to 15-bit one-hot decoders (excluding zero) to select row-lines and column-lines. The counter is activated by the logical-and between its row line and column line. Based on flip-flops, R225 has a higher area and leakage power compared with S256. The area of R225 is 57113 um^2 and the static power constitutes nearly half of the total power of R225. However, with the lower read/write energy, the total energy of R225 is lower than S256, at 2.525 pJ. Although better than S256, the number of counters is still not tolerable, R225 is also suboptimal in these design candidates.

S32 is the design with PMM, QSM, and SRAM-based counters. QSM decreased the number of counters, but two counters are activated per cycle, thus S32 is using two banks of SRAM-based counters, 16 words each. Once again, due to the non-customizable SRAM structure, the capacities of both banks are rounded up to 16. The 16 terms are counting for $Q(n), n \in [1, 16]$. n is computed by a 4-bit adder/subtractor as explained in Section III-B. If $n = 0$, the SRAM is disabled. If $n = 16$, the address is wrapped back to 0. The area of the SRAM is 11442 um^2 in total, the frequency of the SRAM could reach 1170 MHz, and each data access consumes 0.217 pJ. The total energy consumption of the S32 PE is reduced to 2.439 pJ, with a total area of 13045 um^2. Although significantly reduced the number of counters, QSM increased the incrementing counters per cycle from 1 to 2, so that the inefficient incrementing in SRAM-based counters started to take effect. S32 is suboptimal.

R29 is the design with PMM, QSM, and ripple counters. R29 is using 15 up counters and 14 down counters as the same as explained in Section III-B. Since the decoder is customized, it can translate $x+y$ and $x-y$ to $|x+y|-2$ and $|x-y|-2$ with negligible overhead. As a result, 3 ineffectual counters left in S32 are trimmed. QSM decreases static power by reducing the number of counters, and ripple counters exhibit lower dynamic power compared to other types of counters. Consequently, the R29 PE emerges as the optimal design among all discussed candidates, with an energy consumption of 1.274 pJ and an area of 8914 um^2.

B15 is the design with PMM, QSM, and bidirectional counters. B15 uses bidirectional counters so this design only needs 15 counters. The area and power of B15 are 9821 um^2 and 2.734 pJ, respectively. The energy efficiency is worse than R29.

Table I lists the key characteristics of different PEs. PEs with SRAM-based counters or ripple counters can run at 400 MHz, higher than the baseline MAC. This is because SRAM-based counters break the incrementing into two cycles, and ripple counters have low latency as previously analyzed. Take R29

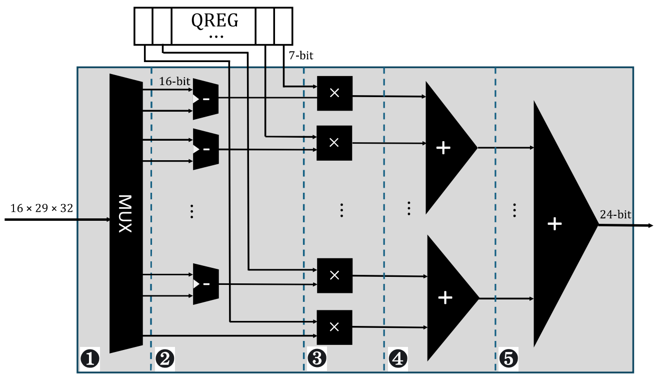


Fig. 9. The circuit design of the converter for R29.

as an example, the critical path only needs to go through a 4-bit adder/subtractor and a decoder, ending at the first flip-flop of the counter. It is quite simpler than in MAC which includes a multiplier and a 24-bit adder.

In summary, we select R29 as the PE design in Cambricon-C SA because of its excellent energy efficiency and speed. Although the area is much larger than the baseline, the significant reduction in power makes Cambricon-C SA valuable under power-sensitive scenarios.

D. Converter

Different PE designs are paired with different converter designs. As we have chosen R29, in this subsection, we focus on the converter design matching R29. As shown in Figure 9. The converting process is divided into four steps:

1) *Counter Readout* (❶): The converter is directly connected to all of the counters in Cambricon-C SA through the multiplexer. In each cycle during the converting process, values stored in all PEs in one column are selected and read out.
2) *Subtraction of Counter Values* (❷): The converter then subtracts the values from the down counters from the corresponding values in the up counters.
3) *Multiplication by Pre-loading Value* (❸): The result from the subtraction is then multiplied by a pre-loaded value stored in QREG, denoted as Q in Equation 6.
4) *Final Output* (❹❺): The multiplication result is summed to yield the final output.

In terms of power consumption, the converter does not operate during the incrementing phase of Cambricon-C SA, which means it primarily incurs static power consumption. The static power consumption is amortized since PEs from different columns share the same converter circuit by time multiplexing. For 2048-dim square matrix multiplication, the conversion power is only 1.4% of the total power consumption. In terms of the area, the converter occupies an area of 36749 um^2, showing the area overhead is not significant.

Converters for different PE designs behave differently. For example, converters for PE designs without QSM require much more cycles and joules due to larger lookup tables and more counters. And the converter for B15 does not require step ❷. Due to these differences, we evaluate all these designs in the experiments in Section VI and show energy breakdowns.

V. Methodology

A. Quantization Network

Table II shows the 8 DNN models we evaluated, spanning a range of architectures including convolutional neural networks (CNNs) and transformers. The quantization methods used for each model are also listed. Those marked with an asterisk* represent quantization-aware training techniques, which can quantize the model to 4-bit without any loss in accuracy.

Cambricon-C SA can effectively work with different quantization methods. For each output or channel, it needs a consistent quantization ratio to be applied. Two key quantization methods are LSQ [11] and KDLSQ [22]. Concretely, LSQ considers the quantization scale factors as trainable parameters and is able to achieve 3-bit precision without accuracy loss on models like VGG16 and ResNet18. KDLSQ uses knowledge distillation [21] to assist in quantizing the network, enabling BERT-base to achieve slight accuracy improvements even when quantized to 4-bit. Cambricon-C SA also supports popular LLM quantization methods like QLLM [27]. These methods often partition the weight matrix into multiple blocks to handle outlier weights. This weight partitioning could potentially interrupt the increment-based computation in Cambricon-C, leading to performance degradation. Fortunately, the partitioning is conducted along the output dimension, so each weight block still contains all the weights required for a single output value. This allows Cambricon-C SA to maintain its advantages with these LLM quantization techniques.

Rank is an important metric that indicates the optimization potential of Cambricon-C for different models. A higher rank

TABLE I
Quantitative comparison of PE designs.

PE Design	Area (um^2)	Energy (pJ)	Frequency (MHz)
MAC	**1422**	2.508	357
S256	21480	3.367	**400**
R225	57113	2.525	**400**
S32	13045	2.439	**400**
R29	8914	**1.274**	**400**
B15	9821	2.734	300

TABLE II
The DNN benchmarks.

Model	# Param.	Quantization	Avg. Rank
VGG-16	138 M	LSQ* [11]	1141
ResNet-18	11.2 M		730
BERT-Base	110 M	KDLSQ* [22]	789
BERT-Large	340 M		1045
OPT-1.3B	1.3 B	QLLM [27]	2235
OPT-2.7B	2.7 B		2664
LLaMA2-7B	6.6 B		4013
LLaMA2-13B	12.7 B		5002

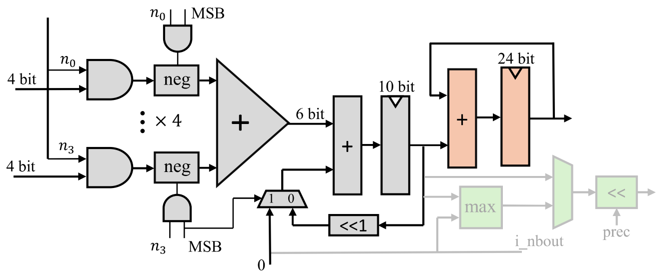


Fig. 10. The circuit design of the modified SIP

allows Cambricon-C to better leverage its advantages. We define rank as the number of MAC (multiply-accumulate) operations required to compute one output value in a matrix multiplication. For example, in the matrix multiplication $\mathbb{R}^{d_1 \times d_2} \times \mathbb{R}^{d_2 \times d_3} = \mathbb{R}^{d_1 \times d_3}$, the rank is d_2. In the case of DNN models, the rank for convolutional layers is determined by the filter size. For a $3 \times 3 \times 512$ filter, each output value requires $3 \times 3 \times 512 = 4608$ MAC operations, so the rank is 4608. For fully connected layers, the rank depends on the embedding dimension. Table II presents the average rank across all layers for each DNN model we evaluated. Generally, CNN models tend to have a higher average rank than transformer models, since the dimensions of KQV matrices in transformers are often smaller, like 64 or 128. These smaller matrix multiplications reduce the average rank. In contrast, for LLMs, the hidden layer dimensions are much larger, often on the order of 2048 or 4096. The feedforward layers also have very high ranks. This provides an opportunity for Cambricon-C to fully leverage its advantages, as its performance scales well with higher-rank matrix multiplications.

B. Hardware Configuration

For Cambricon-C PE we implemented S256, R225, S32, R29, and B15. We choose the R29 to implement the complete Cambricon-C ACC.

The SRAM in S256 and S32 is modeled by OpenRAM [16] and the SRAM buffer is modeld by CACTI7 [2]. Other designs are implemented by the Verilog RTL. We use Synopsys Design Compiler (DC) to synthesize the netlist under FreePDK 45nm technology node [28]. The energy was obtained by the Prime-Time PX with gate-level value change dump (VCD). The cycle-level hardware simulator for 4-bit DNN models was built based on ANT [15].

We evaluated the DNN performance on the traditional MAC-SA (i.e. TPU), Stripes, and Cambricon-C ACC. For the TPU and Cambricon-C ACC, the computation unit is implemented as 32×32 systolic array. The details of the implementation of MAC and Cambricon-C PE are presented in Section IV-C.

For Stripes [23], it takes SIP as the computation unit. We slightly modified the structure of SIP as depicted in Figure 10. Compared to the original design, the modified SIP removes the redundant modules such as the comparator for max pooling, which are denoted in green. Additionally, we replaced the precision shifter with a 24-bit adder and a 24-bit register to meet the precision requirements. As the 24-bit adder in the SIP is not always active, it does not contribute much energy consumption. We also insert a pipeline stage in the adder tree to ensure it can run at 400MHz clock frequency. In the end, each SIP dissipates 2.265 pJ and takes up 1701 um^2. For the complete design of Stripes, we set 1024 SIPs, whose computation ability is the same as the TPU and Cambricon-C SA.

VI. Evaluation

In this section, we first evaluate the power and area overhead of Cambricon-C in Section VI-A. Then, In Section VI-B, we introduce the power consumption of baselines and Cambricon-C. In Section VI-C, we extend the converter that constitutes the Cambricon-C SA for the complete matrix multiplication. Section VI-D presents the performance of Cambricon-C ACC on various DNN models. Finally, we compare Cambricon-C with other computation unit in Section VI-E.

A. Hardware Characteristics

Table III presents the details of hardware characteristics. The full Cambricon-C ACC Core takes up 21.39 mm^2 area, and the peak power reaches 3199.96 mW at the 45 nm technology node. For the baselines, the TPU and modified Stripes occupy 10.62 mm^2 and 10.50 mm^2 respectively, with 2347.66 mW and 2248.33 mW power consumption when normalized to the 400MHz situation.

The value of the peak power of TPU and Stripes are lower than Cambricon-C. However, for Cambricon-C ACC, there are some modules that are not activating most of the time. It will make the average power of Cambricon-C ACC quite lower than the baselines. Though the on-chip area of Cambricon-C ACC is larger than baselines. Considering the off-chip memory, especially in the LLM situation, which needs a large capacity of off-chip memory, the extra area overhead of Cambricon-C is subtle. Besides, as Cambricon-C could run at a higher clock frequency than TPU, it will save more static energy of SRAM and off-chip memory, which will cause considerable energy consumption for the total system.

TABLE III
Hardware Characteristics.

-	Area (mm^2)	(%)	Power (mW)	(%)
Cambricon-C ACC	21.39	100	3199.95	100
Cambricon-C SA	12.23	56.90	1879.36	58.97
R29 Array	9.13	42.47	522.24	16.39
Converter	3.10	14.44	940.86	42.59
SFU	1.06	4.91	193.55	6.07
NBIn	1.95	9.07	469.33	14.73
SB	1.10	5.12	242.34	7.60
NBOut	2.63	12.23	297.04	9.32
Ctrl	0.11	0.51	20.02	0.63
IB	0.36	1.67	10.11	0.32
MC	0.23	1.07	33.20	1.04
PHY	1.83	8.51	41.78	1.31

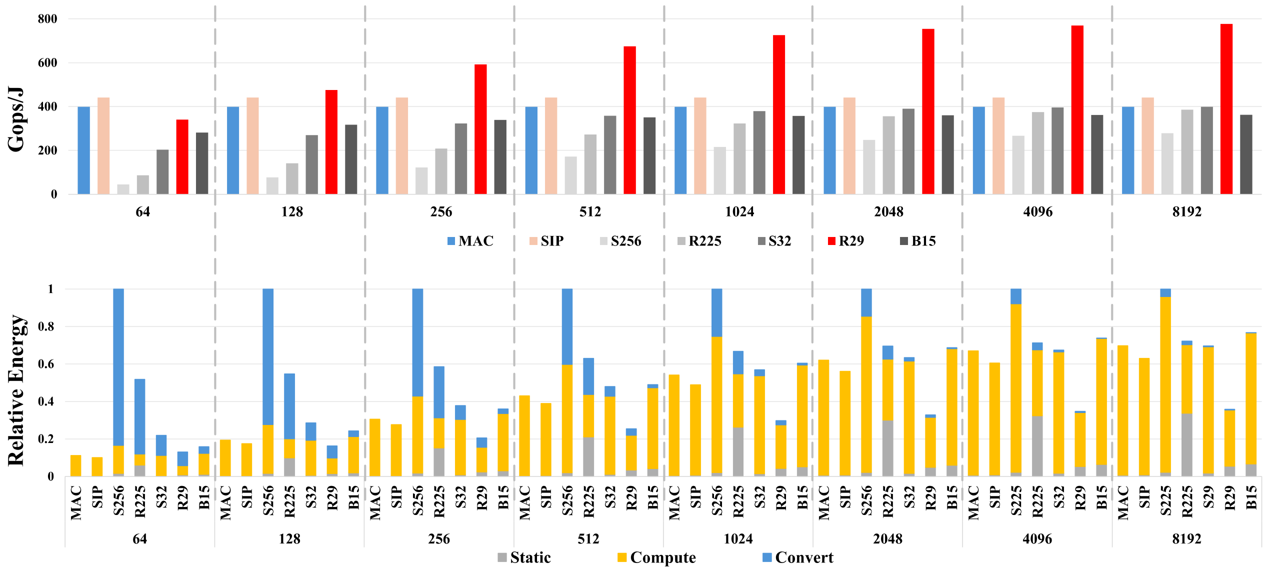


Fig. 11. Energy efficiency (top) and energy breakdown (bottom) under different matrix dimension.

TABLE IV
POWER BREAKDOWN OF PEs, NORMALIZED TO 400MHz.

Power (mW)	Reg & SRAM	Comb.	Static	Total
MAC [33]	0.4254	0.5680	0.0098	1.003
SIP [23]	0.5396	0.3546	0.0122	0.906
S256	1.3221	0.0168	0.0078	1.347
R225	0.4210	0.1054	0.4836	1.010
S32	0.8485	0.1216	0.0054	0.975
R29	0.2224	0.2084	0.0790	0.510
B15	0.4995	0.5056	0.0886	1.094

B. Power Breakdown of the PE

To further explore the performance of Cambricon-C, we break the power consumption of PEs of Cambricon-C and baselines in Table IV.

First, since Cambricon-C primitives the MAC to the counting, the power consumption of combination circuits in most Cambricon-C schemes is lower than that of MAC and SIP.

Besides, the benefits of QSM are evident. For the design of S256 and S32, the gain of power is mainly from reducing the data access power of SRAM, as the size of SRAM decreases from 256 words to 32 words. Although the number of access of SRAM doubles, S32 still gets net benefit. However, as QSM activates two counters per cycle, it introduces an additional set of 16-bit registers and half-adders. It makes the gain of QSM limited on SRAM counters schemes. For R225 and R29, the benefits of QSM mainly lie in reducing the static power. Since ripple counters are based on the cascading of Dffs, they needs more transistors to store counting values compared to the SRAM counter. Reducing the number of counters can significantly reduce the static power.

The R29 is the most energy-efficient design. For both the R225 and R29, the power consumption of the sequential circuit is also lower than that of MAC and SIP, though there are lots of Dffs. This is due to the low switch activity of ripple counters. It significantly reduces the power consumption of the sequential circuit compared to all other schemes. In the end, the energy efficiency of R29 can be improved by $1.97\times$ than MAC at most.

For B15, it doesn't benefit from the bidirectional counters. The power of B15 is higher than S32 and R29. First, the counter has a more complicated circuit for bidirectional counting. It leads to higher combination power consumption and limits the maximum frequency. Besides, the bidirectional counting leads to more bit flip of counters, which brings higher power consumption.

C. Analysis with the Matrix

In this section, we compare the energy performance between MAC-SA, Stripes, and five kinds of PEs of Cambricon-C for matrix multiplication. For Cambricon-C SA, the complete process of matrix multiplication includes counting and conversion. The power consumption of the R29 converter is 42.41 mW, which is much higher than the PEs. However, the converter only runs once at the end of the multiplication operation, so it contributes a little to the total energy. For the MAC-SA and the Stripes, the final result is obtained directly without conversion. Thus, the conversion energy is 0.

Figure 11 illustrates the performance of various designs on matrix multiplications with different dimensions, from 64 to 8192. When the matrix dimension is small, the energy consumption of the converter dominates, resulting in no advantage for Cambricon-C SA over MAC-SA and Stripes. When the matrix dimension reaches 128, the R29-based Cambricon-C SA surpasses MAC and SIP, and becomes the most

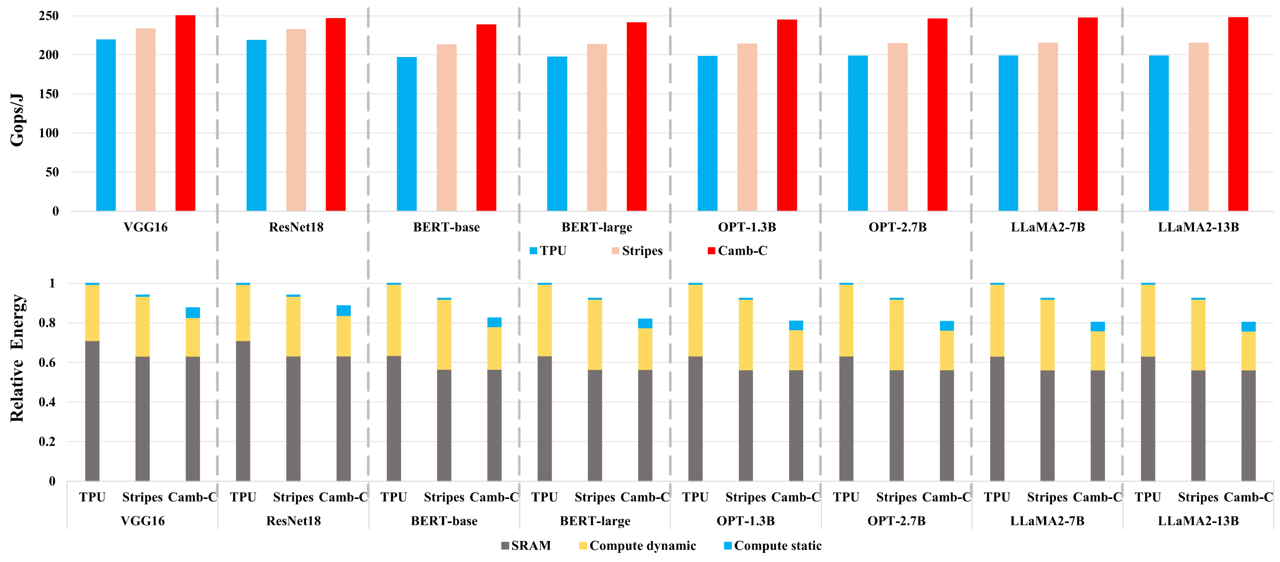


Fig. 12. Energy efficiency of TPU, Stripes, and Cambricon-C on various DNN workloads.

energy-efficient method. As the matrix dimension increases, Cambricon-C SA approximate to the maximum benefit.

To further explore Cambricon-C SA, we break down the total energy into the conversion, compute, and static energy in Figure 11 bottom. First, we compare the different schemes of Cambricon-C SA in general. For the S256 and S32, the converter is a $8b \times 16b$ MAC unit, which means they need multiple cycles to complete the conversion. Though the static power of the S256 and S32 are quite lower than other Cambricon-C PEs, the high conversion energy and computation energy can't bring performance improvement. For R225 and R29, the computation energy is lower because the energy required for data access in D flip-flops is not as high as that for SRAM. When incrementing, it doesn't activate the 16-bit register. Additionally, for R29, it could read out and convert all data in counters at once, which reduces the conversion energy consumption. For the B15, it pursues the least number of counters, which has the lowest conversion energy. However, it leads to a more complicated circuit design for counters, which brings a higher computation energy.

The benefits of QSM are significant. For S32, these benefits come in two parts. On the one hand, as the capacity of SRAM decreases, the energy consumption per data access also decreases. On the other hand, the converter requires only 29 cycles to complete the data conversion, effectively reducing the energy consumption to an acceptable level. Also, in the R29, the QSM brings lower static power, and the converter only needs to process the 29×16-bit counting results, not 225×16. It makes the R29 become the most efficient scheme. Finally, R29 achieves performance gains of approximately $1.95\times$ and $1.76\times$ compared to MAC and SIP, respectively, which is quite close to the theoretical maximum gain.

D. Performance on DNN models

The advantages of Cambricon-C lie in high-dimension matrix multiplication, which is widely present in DNNs. In Table II, we present the average rank of each network. Even the smallest network in the benchmark, ResNet-18, also has an average rank exceeding 512. This indicates that the Cambricon-C has an excellent optimization potential for kinds of DNNs, not limited by the LLM.

To further demonstrate the advantages of Cambricon-C, we implemented a complete on-chip system, Cambricon-C ACC. Both the array size of TPU and Cambricon-C ACC are 32×32. The input data for simulation is sampled from the real neural network. Figure 12 shows the performance of Cambricon-C ACC and TPU in various networks.

The Cambricon-C ACC consistently outperforms TPU and Stripes across various DNN models. In ResNet-18, Cambricon-C ACC shows $1.13\times$ energy improvement on TPU and $1.06\times$ on Stripes. In LLaMA2-13B, the improvement increases to $1.25\times$ and $1.15\times$ respectively. These enhancements are attributed to two factors. First, Cambricon-C ACC could run at a higher clock frequency than TPU, which leads to a lower static power of buffers (i.e. NBIn, NBOut, and SB). Second, the dynamic power of the computation array decreases. Beyond the buffer, the Cambricon-C SA achieves $1.53\times$ and $1.51\times$ improvement compared to the computational units in TPU and Stripes, respectively. The energy gain of Cambricon-C SA is not as high as depicted in Figure 11. There are two main reasons for this. First, MAC and SIP are more sensitive to data distribution compared to R29. Second, the stall cycles in R29 contribute to higher static power consumption than other two schemes.

E. Comparison with Other Methods

We also reproduce the CARAT [34], FuseKNA [40], and Bit-Pragmatic [1], which are designed as alternatives to com-

TABLE V
ENERGY CONSUMPTION OF PEs

Design	Energy (pJ)	Frequency (MHz)
MAC [33]	2.508	357
SIP [23]	2.265	400
PIP [1]	2.549	400
FuseKNA [40]	6.450	400
CARAT [34]	6.638	400
R29	**1.273**	400

mon MAC units. These designs achieve energy efficiency improvements under 8-bit and 16-bit scenarios. However, as discussed in Section II-B, the circuit design for INT4 computation is quite simple and neat, making it challenging to achieve net gains in the INT4 context.

Table V presents the average energy consumption when PEs perform matrix multiplication. The performance of CARAT and FuseKNA drops significantly. CARAT is suitable for the float point data. For INT4, it requires a wider computation array to shed the throughput loss. However, this wider array size results in increased power overhead from the additional registers. For FuseKNA, the LUT-based AGU involves multiple registers, leading to higher energy consumption. PIP is the computation component in Bit-Pragmatic, which is capable of skipping the ineffective bit-value computation to save energy. However, it introduces extra shifters and decoders, which contribute to the additional energy consumption.

Furthermore, all the aforementioned methods are unable to avoid high bit-width accumulation, which is a major contributor to energy consumption in INT4 computation. Consequently, none of these designs achieve energy improvement under the INT4 situation. Cambricon-C addresses this issue by transforming both multiplication and accumulation into a highly energy-efficient counting operation, resulting in net gains.

VII. Related Work

Bit serial computing is a computation approach that unrolls the multiplicand into bit flows so that multiplications are transformed into accumulations. Stripes [23] is capable of processing bit strings of flexible length and enabled early-stop when higher-significant bits to process are all zero. Bit-pragmatic [1] and Bit-tactical [8] further improve by skipping zero bits found inside the flow. Loom [36] focuses on low-precision serial computing in source-limited situations. ShapeShifter [4] explores bit-serial computing in the context of fine-grain group quantization. However, these methods cannot avoid the redundant use of high-cost hardware circuits when computing the same values, which reveals the optimization potential.

There are some studies that exploited the repetitive computation in matrix multiplication. UCNN [19] reduces the memory reads by exploiting the repetition of the weight and activation. FuseKNA [40] captures both ineffectual and repetitive additions in bit-serial computation by exploiting bit repetition and bit sparsity in weights. Cambricon-P [17] eliminates bit-level redundancy observed in inner-products by a bit-indexed inner-product computing scheme. CARAT [34] observed the trends towards batched low-precision data, and transformed multiplications into additions to remove computations on repetitive values. However, none of these works can achieve net improvement on very low-precision data such as INT4, due to their relatively high architectural overheads.

VIII. Conclusion

We propose a novel 4-bit matrix computation unit Cambricon-C SA via primitivization, that is, breaking down MAC operations to 1-ary successor functions, also known as counting. We explored and selected the optimal design of primitivized matrix multiplication unit, which is based on techniques including Quarter Square Multiplication (QSM) and ripple counters. Experimental results show that the proposed Cambricon-C SA achieves up to 1.95× improvement on energy efficiency over conventional MAC-based systolic array. Applied to a complete accelerator, Cambricon-C ACC achieves 1.13-1.25× energy efficiency improvement compared with TPU over various DNN workloads.

Acknowledgment

This work is partially supported by the National Key RD Program of China(under Grant 2023YFB4502200), the NSF of China(under Grants U22A2028, 61925208, 62222214, 62341411, 62102398, 62102399, U20A20227, 62372436, 62302478, 62302482, 62302483, 62302480), Strategic Priority Research Program of the Chinese Academy of Sciences, (Grant No.XDB0660200, XDB0660201, XDB0660202), CAS Project for Young Scientists in Basic Research(YSBR-029), Youth Innovation Promotion Association CAS and Xplore Prize.

References

[1] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture*, 2017, pp. 382–394.

[2] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[3] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh, "Video generation models as world simulators," 2024. [Online]. Available: https://openai.com/research/video-generation-models-as-world-simulators

[4] S.-T. Chen, C. Cornelius, J. Martin, and D. H. Chau, "Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part I 18*. Springer, 2019, pp. 52–68.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[6] T. C. Chen, "A binary multiplication scheme based on squaring," *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 678–680, 1971.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[8] 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*, 2019, pp. 749–763.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[11] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019.

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[13] G. Gerganov, "ggml," 2023. [Online]. Available: https://github.com/ggerganov/ggml

[14] J. W. L. Glaisher, "The method of quarter squares," *Nature*, vol. 41, no. 1045, pp. 9–9, 1889.

[15] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[16] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–6.

[17] Y. Hao, Y. Zhao, C. Liu, Z. Du, S. Cheng, X. Li, X. Hu, Q. Guo, Z. Xu, and T. Chen, "Cambricon-p: A bitflow architecture for arbitrary precision computing," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 57–72.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 674–687.

[20] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," *arXiv preprint arXiv:1712.00409*, 2017.

[21] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[22] J. Jin, C. Liang, T. Wu, L. Zou, and Z. Gan, "Kdlsq-bert: A quantized bert combining knowledge distillation with learned step size quantization," *arXiv preprint arXiv:2101.05938*, 2021.

[23] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

[26] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[27] J. Liu, R. Gong, X. Wei, Z. Dong, J. Cai, and B. Zhuang, "Qllm: Accurate and efficient low-bitwidth quantization for large language models," *arXiv preprint arXiv:2310.08041*, 2023.

[28] N. [n.d.], "Freepdk45," Accessed April 20, 2023. [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk45/

[29] Nvdia, "Tensorrt-llm," 2023, a TensorRT Toolbox for Optimized Large Language Model Inference. [Online]. Available: https://github.com/NVIDIA/TensorRT-LLM

[30] OpenAI, "Gpt-4," 2023. [Online]. Available: https://openai.com/research/gpt-4

[31] OpenVINO, "Openvino," 2024, openVINO™ is an open-source toolkit for optimizing and deploying AI inference. [Online]. Available: https://docs.openvino.ai

[32] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[33] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. Vazir Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. Richard Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. Hyun Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[34] Z. Pan, J. San Miguel, and D. Wu, "Carat: Unlocking value-level parallelism for multiplier-free gemms," in *ASPLOS*, 2024.

[35] 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 *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 304–317.

[36] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[38] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. Singh Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. Michael Smith, R. Subramanian, X. Ellen Tan, B. Tang, R. Taylor, A. Williams, J. Xiang Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[40] J. Yang, Z. Zhang, Z. Liu, J. Zhou, L. Liu, S. Wei, and S. Yin, "Fusekna: Fused kernel convolution based accelerator for deep neural networks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 894–907.

[41] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *arXiv preprint arXiv:2310.19102*, 2023.