Applied Intelligence
https://doi.org/10.1007/s10489-022-04251-3

An efficient lightweight CNN acceleration architecture for edge computing based-on FPGA

Ruidong Wu[1] · **Bing Liu**[1] · **Ping Fu**[1] · **Haolin Chen**[1]

Accepted: 7 October 2022

Abstract
With the system performance, volume and power restriction requirements in edge computing, single chip based on Field Programmable Gate Array (FPGA), with the characteristics of parallel execution, flexible configuration and power efficiency, is more desirable for realizing Convolutional Neural Network (CNN) acceleration. However, implementing a lightweight CNN with limited on-chip resources while maintaining high computing efficiency and utilization is still a challenging task. To achieve efficient acceleration with single chip, we implement Network-on-Chip (NoC) based on Processing Element (PE) that consists of multiple node arrays. Moreover, the computing and memory efficiencies of PE are optimized with a sharing function and hybrid memory. To maximize resource utilization, a theoretical model is constructed to explore the parallel parameters and running cycles of each PE. In the experimental results of LeNet and MobileNet, resource utilization values of 83.61% and 95.28% are achieved, where the throughput values are 53.3 Giga Operations Per Second (GOPS) and 41.9 GOPS, respectively. Power measurements show that the power efficiency is optimized to 77.25 GOPS/W and 85.51 GOPS/W on our platform, which is sufficient to realize efficient inference for edge computing.

Keywords FPGA · CNN · Acceleration architecture · Efficient inference

1 Introduction

Convolutional Neural Network (CNNs) have been widely used in edge computing scenarios, for example, in nano robot [24], intelligent sensors [11] and wearable devices [3]. In particular, in Field Programmable Gate Array (FPGA) based CNN acceleration, due to system performance, volume and power restriction requirements in these scenarios, it is more desirable to realize CNN acceleration within single chip. However, the internal resources of FPGAs are usually limited, and implementing lightweight CNN acceleration while maintaining efficiency and utilization is a challenging task. Therefore, the focus of related research has changed from pure throughput [29] to comprehensive criteria, such as normalized throughput [18] and runtime multiplication efficiency (RME) [8, 28]. These studies indicate that the improvement of throughput and power energy ratio is determined by a combination of computing efficiency and resource utilization. Therefore, for lightweight CNN acceleration, efficient inference not only reduces hardware platform requirement, with limited computing and memory resource use, but also improves the potential throughput and power efficiency, which makes efficient inference desirable for extreme and harsh scenarios.

In the existing research, CNN acceleration mainly involves two aspects: the algorithm and platform. To improve algorithm, a lightweight CNN compresses the model size by hundreds of times and alleviates the balance between accuracy and speed. There are two types of strategies: (1) compressing the original network with pruning [14] or trained quantization [10] and (2) constructing a network

✉ Bing Liu
liubing66@hit.edu.cn

Ruidong Wu
17B901027@stu.hit.edu.cn

Ping Fu
fuping@hit.edu.cn

Haolin Chen
21S005089@stu.hit.edu.cn

[1] School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150001, China

Published online: 18 October 2022

with other sparse convolution types, such as MobileNet [25], ShuffleNet [17] and Xception [4]. For the implementation of CNNs, commonly used embedded platforms include Advanced RISC Machine (ARM) [27], Graphics Processing Unit (GPU) [15], Application Specific Integrated Circuit (ASIC) [22] and FPGA. Of these platforms, FPGA has been a popular choice, exhibiting the characteristics of online reconfigurability, low power consumption, and high performance [2, 6, 7, 18]. New CNN has been continuously introduced in recent years, and FPGA can be directly usef for novel design and still maintain high compatibility.

Various works on FPGA-based CNN acceleration have been performed in recent years. In the early research on CNN acceleration, the roofline model was proposed as the basic theory to solve the balance problem between throughput and bandwidth [30]. In this research, Computing To Communication (CTC) is introduced to express operational intensity, and it mathematical illustrate the possible ways to improve throughput. To improve throughput, Toeplitz matrix [13] can be used to change CNN and Fully Connected (FC) layer into matrix multiplication. It maps these operations to general matrix multiplication and enables the matrix accelerator to be applied to network acceleration. Tiling technology [19, 20] can be used to divide the original structure into multiple small sets, reducing the memory requirement. With the focus on parallel strategies, standard CNN acceleration, such as AlexNet [1] and VGG-16 [16], has been proposed to increase throughput by the additional consumption of computing units. However, these studies support the acceleration of only limited layer types, and they cannot be directly applied to lightweight CNN with multiple convolution types. In terms of the other type of CNN acceleration, computational transformation can be used to change CNN into a fast algorithm, such as Gauss complex multiplication [31], Strassen's matrix multiplication [5], Winograd [12], Fast Fourier [21], and Fast Finite Impulse Response [26]. These fast algorithms improve throughput by reducing multiplication through an increase in addition operations. However, the sparse property of computational transformation is heavily dependent on the trained parameters, and it also destroys the original timing structure and increases timing constraint requirements. This leads to uncertain CNN acceleration performance, which is not suitable for time-sensitive scenarios of lightweight CNN in edge computing.

In terms of the hardware structure of some prior works, external memory is mounted to cache temporary data. This structure increases the Printed Circuit Board (PCB) size and the power consumption of other components, and does not affect volume and power sensitive scenarios [24]. In research on throughput, although some works have improved the peak performance of hardware platform with more logical resources, their effective utilization is only in the range of 40% to 80% [2, 7, 16, 26]. The mining of potential computing performance requires further research and optimization.

Based on the above analysis, we propose an embedded lightweight CNN acceleration architecture within a single chip for efficient inference, which yields computing and memory resource efficiency, as well as improving the power energy ratio. The main contributions of this research are as follows:

(1) Multiple node arrays with a full-pipelined operator structure are designed as the basic element, and are optimized in terms of resource efficiency through a sharing activation function and hybrid memory.
(2) To maximize utilization, a theoretical model of computing and memory resource is constructed to guide the exploration of parallel parameters and running cycles, and this model also provides a reference for throughput and computing efficiency.
(3) Network-on-Chip (NoC) architecture based on multiple node arrays is constructed for network implementation, and the evaluation of normalized throughput is used to assess the computing efficiency.
(4) The implementation of pruned LeNet and MobileNet is carried out in experiments without external memory, and the normalized throughput and power efficiency are compared with those of related works to verify our design.

2 Full-pipelined operator structure

The basis of CNN acceleration in this study is the full-pipelined operator structure design. With this design, we aim to maintain high computing efficiency for power-sensitive scenarios. This section introduces the component of efficient full-pipelined operator, and then the combination of multiple operators with node arrays.

2.1 Operator design

The full-pipelined structure is the key point of operator design. This design realizes the basic operation of CNN in high performance mode. A diagram of typical CNN is shown in Fig. 1, where R_{in}, C_{in}, and M are the row, column and input channels of feature map, respectively. Moreover, R_{out}, C_{out}, and N are the row, column and channels of output feature map, respectively. In the pixel processing stage of output feature map, a temporary area of input feature map with same size of k is multiplied and accumulated with N kernels. Finally, the accumulated result is transmitted with bias through the activation function.

A commonly used parallelism, as shown in Fig. 1, is T_M. When a large scale dimension is faced, input data

is split into multiple groups. This provides the possibility of adaption to variable dimensional data. Therefore, the full-pipelined operator structure with the components of multiplier and addition tree is realized in macro-design. The input of operator structure is of a flexible size in micro-design and can be arbitrarily configured through basic parallelism T_M. The accumulator automatically completes the addition with bias or previous result according to the structure definition. Moreover, the structure can also optimized with the implementation of a nonlinear activation function, which improves resource utilization and mines potential efficiency.

Based on this mechanism, the general full-pipelined operator structure is constructed in Fig. 2. The input is T_M from feature map A and weight W, and the output is the accumulated result Z with post-processing of activation function, which can be expressed by (1). Depending on whether computing resources are needed, activation functions can be divided into two categories: (1)Logical operations, such as maximum, Rectified Linear Unit (ReLU), shift and truncation. These activation functions can be directly constructed with programmable logic and do not consume Digital Signal Processing (DSP) units; (2) Arithmetical operations, such as Leaky Relu, Batch Normalization (BN) [9], linear mapping, Sigmoid and Tanh. These functions require additional DSP units to release arithmetic computations, which may cause a significant drop in computing efficiency.

$$Z = f\left(\sum_{i=1}^{T_M} A\left(i\right) \times W\left(i\right) + b\right) \tag{1}$$

$$Z_O = f\left(\sum_{j=0}^{\left\lceil \frac{M}{T_M} \right\rceil - 1} \sum_{k_0=1}^{K} \sum_{k_1=1}^{K} \sum_{i=1}^{\min(T_M, M - jT_M)} A\left(k_0, k_1, i + jT_M\right) \times W\left(O, k_0, k_1, i + jT_M\right) + b_O\right) \tag{2}$$

Without loss of generality, the processing of CNN in Fig. 1 is carried out with the grouping expansion on the basis of (1). This can be expressed with (2), where the input feature map is $A \in \mathbb{R}^{k \times k \times M}$, the weight is $W \in \mathbb{R}^{N \times k \times k \times M}$ and the output feature map is $Z \in \mathbb{R}^{N}$. In (2), the inner loop includes the parallelism T_M of (1), and the outer loop includes the kernel size K and input channel T_M. Due to the requirements of different kernel sizes and output channels, the accumulator is configurable, and the activation function can be optimized with the minimum interval.

2.2 Node array design

The full-pipelined operator with node arrays is the foundation of data reuse. In the processing of CNN, the detail processing is shown in (3). In conjunction with Fig. 1, the intuitive parallelisms in (3) include R_{out}, C_{out}, M, N and K. Note that the following problems are faced with the parallelism of K. (1) The reading address is not continuous, and an additional encoding module is required to automatically generate the read address. (2) Since the kernel sizes of different layers may be inconsistent, the original operator cannot be directly applied to these layers and thus is not useful for the following acceleration design. Therefore, an additional coding module is needed to accommodate the address change, which makes the logic design and timing constraint more complex.

$$OP = R_{out} \times C_{out} \times M \times N \times k \times k \times 2 \tag{3}$$

On the basis of the full-pipelined operator, other parallel parameters are released in (4) in addition to T_M, including the output channel parallelism T_N, column parallelism T_C and row parallelism T_R. Then, the number of cycles T of CNN processing is shown in (4), where the range of parallelism is also defined.

$$\begin{cases} T = \left\lceil \frac{R_{out}}{T_R} \right\rceil \times \left\lceil \frac{C_{out}}{T_C} \right\rceil \times \left\lceil \frac{M}{T_M} \right\rceil \times \left\lceil \frac{N}{T_N} \right\rceil \times K \times K, \\ \left(T_R, T_C, T_M, T_N\right) \in \mathbb{N}+, \\ T_R \in [1, R_{out}], \\ T_C \in [1, C_{out}], \\ T_M \in [1, M], \\ T_N \in [1, N], \end{cases} \tag{4}$$

The expression of (4) is shown in Fig. 3 with the combination of multiple node arrays, where the basic unit is the operator structure in Fig. 2. This operator completes the stacking of node arrays in three-dimensional space. Moreover, the projection of node arrays on two-dimensional space is shown on the right of Fig. 3, where the horizontal and vertical axes indicate the reuse of weight T_C and feature map T_N, respectively. Since the node of Fig. 3 satisfies the full-pipelined mechanism in data processing, it lays the foundation for computing efficiency in subsequent design.

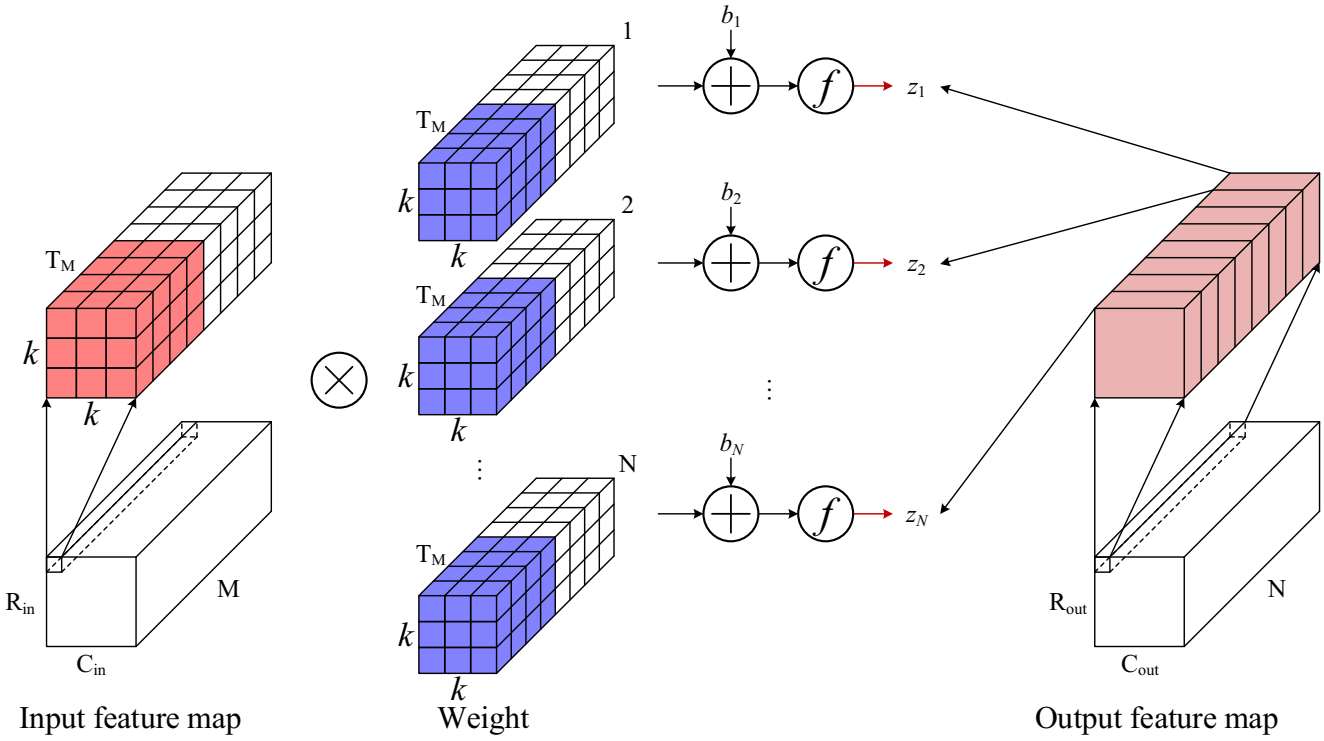


Fig. 1 The processing diagram of convolution neural network

3 Resource estimation model and optimization design

To maximize resource utilization, a resource estimation model of computing and memory resources should be constructed to guide the exploration of parallel parameters. This model not only includes the coarse-grained model of the full-pipelined operator structure, but also includes fine-grained optimization of computing and memory resources. This section expresses the DSP-based computing efficiency and BRAM-based memory efficiency from the introduction of resource estimation model.

3.1 Computing resource model and efficiency optimization

The computing resource model is based on the feature of DSP utilization, where Giga Operations Per Second (GOPS) is used for the benchmark of throughput. The utilization of DSP commonly indicates the degree of mining the potential throughput. Constructing a theoretical model is necessary to evaluate the number of DSP units, and also provides theoretical guidance for the solution to explore parallel parameters.

The single node in Fig. 2 can be divided into a Multiply Accumulate Tree (MAT) and an Activation Function (AF). Factors that affect the number of DSPs in a MAT include the data type and input dimension. The number α of DSPs for a simple multiplication with different data types in the classical design of Xilinx FPGA is shown in (5).

$$\alpha = \begin{cases} 5 \times T_M, \text{type} = float32 \\ 3 \times T_M, \text{type} = int32 \\ T_M, \text{type} = int16 \\ 2^{\lceil \log_2 T_M \rceil}, \text{type} = int8 \end{cases} \tag{5}$$

According to Fig. 3, the number N_{MAC} of DSPs for the MAT in three-dimensional space is:

$$N_{MAC} = T_R \times T_C \times T_N \times \alpha \tag{6}$$

Similarly, if the AF uses T_A DSPs, the corresponding number N_{ACT} of DSPs in three-dimensional space is:

$$N_{ACT} = T_R \times T_C \times T_N \times T_A, \{T_A \in \mathbb{N}\} \tag{7}$$

Then, the total number N_{Total} of DSPs is expressed in (8).

$$N_{Total} = T_R \times T_C \times T_N \times (\alpha + T_A) \tag{8}$$

In (8), although the construction of AF requires additional DSPs, there is no obvious improvement in throughput. If α is greater than T_A, the impact of AF can be ignored. However, when the number of channels in input feature map is small, a significant decrease in computing efficiency occurs, and an optimized design is required to reduce the negative impact of this phenomenon.

From the expression of (2), the interval cycle of output is $K \times K \times \frac{M}{T_M}$. Obviously, there are inevitable bubbles

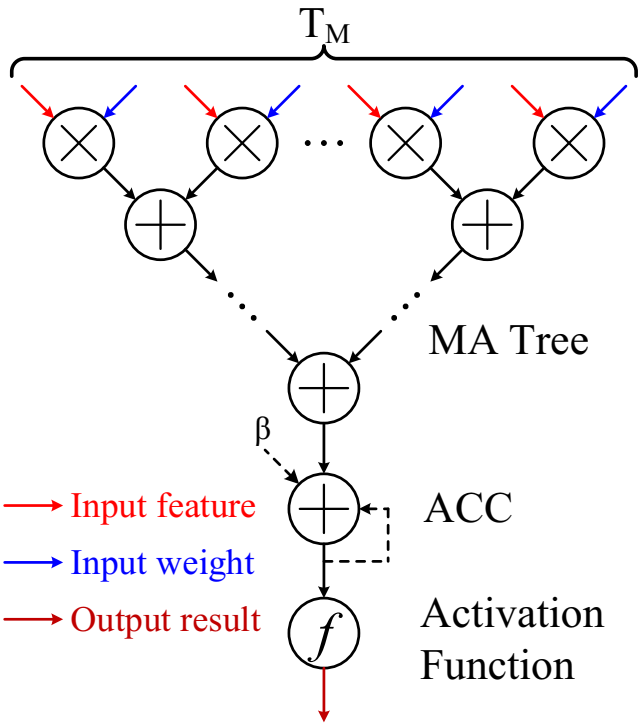


Fig. 2 General full-pipelined operator structure

in the AF, especially when K is large and T_M is small. We proposes sharing activation function to optimize the input of AF. Figure 4 shows a schematic diagram of the sharing activation function, where S_f is the sharing factor. The optimization mechanism is as follows: If there

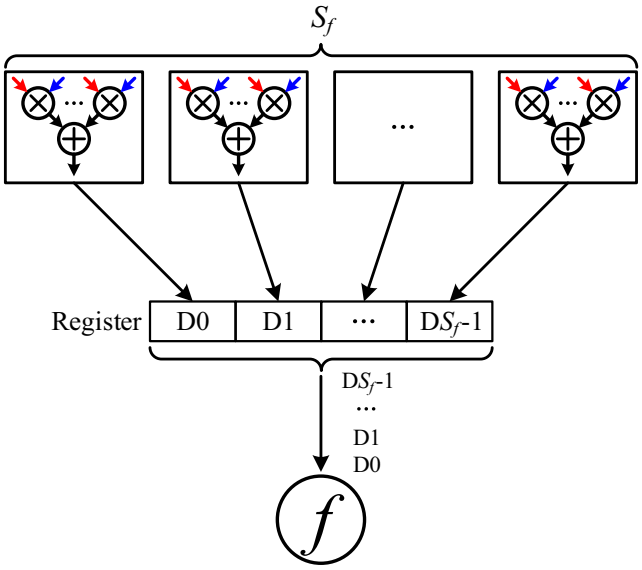


Fig. 4 Sharing activation function

are S_f accumulated results of MAT storing in multiple registers, the contents of register are $D1 \sim D_{S_f}$. Multiple registers enter an independent activation function through the parallel-to-serial converter for sharing.

Therefore, the new number N'_{ACT} of DSPs with the sharing activation function is:

$$N'_{ACT} = \left\lceil \frac{T_R \times T_C \times T_N}{S_f} \right\rceil \times T_A, S_f \in \mathbb{N}+ \tag{9}$$

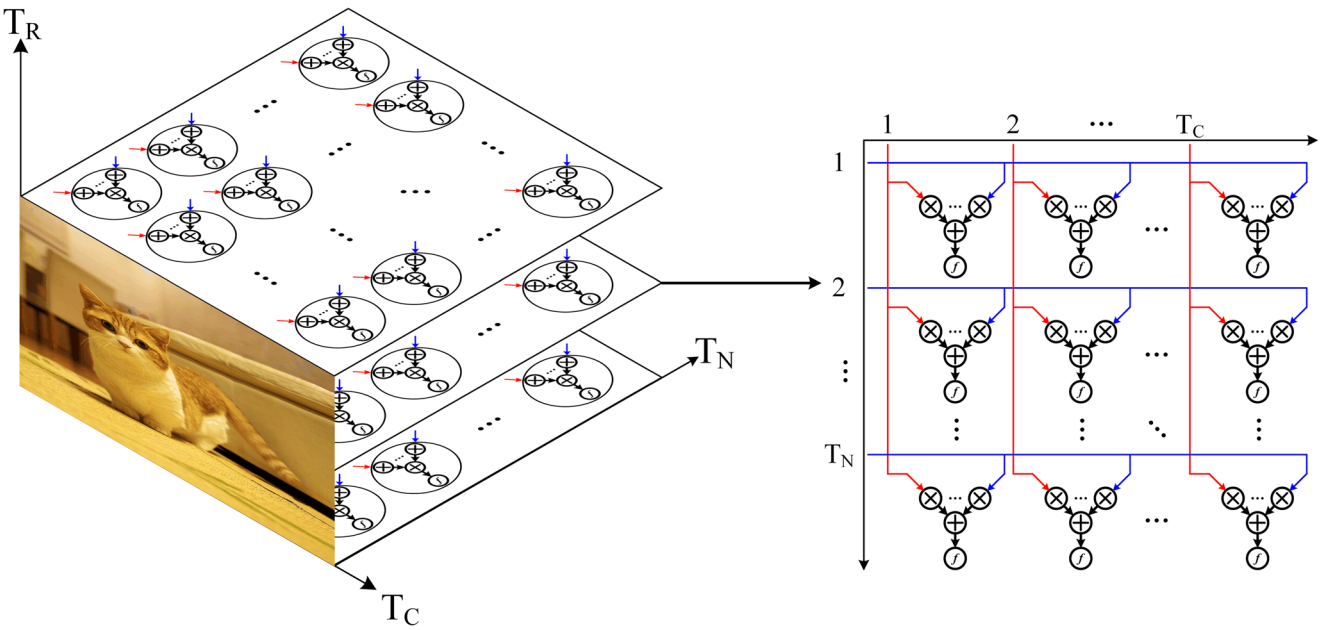


Fig. 3 The schematic diagram of full-pipelined operator with multiple node arrays

Similarly, due to the limitation of node arrays and the interval cycle of single node, the range of sharing factor is:

$$1 \le S_f \le \min\left(T_R \times T_C \times T_N, \left\lceil \frac{M}{T_M} \right\rceil \times K \times K\right) \quad (10)$$

Assuming that the impact of rounding up in (10) is not considered, the optimization coefficient P of sharing activation function is:

$$P = \frac{N_{DSP} + N'_A}{N_{DSP} + N_A} \approx \frac{S_f\alpha + T_A}{S_f\alpha + S_f T_A} \quad (11)$$

Through the combination of (10) to (11), the maximum and minimum values of P are:

$$\begin{cases} \max(P) = \lim\limits_{S_f \to 1} P = 1 \\ \min(P) = \lim\limits_{S_f \to +\infty} P = \frac{\alpha}{\alpha + T_A} \end{cases} \quad (12)$$

Then, the range of P is:

$$P \in \left(\frac{\alpha}{\alpha + T_A}, 1\right] \quad (13)$$

Equation (13) shows that the effect of optimization depends on the data type, T_M and activation function. Extremely, when T_M is the smallest value and T_A is largest value, P exhibits the best optimization effect. In depthwise convolution, T_M is fixed to 1 in 16bit, and activation function requires additional down-scaling operation [10]. The typical value of optimization T_A is 4. The coefficient P_{DW} of depthwise convolution is expressed in (14). This will be verified in the experimental results.

$$P_{DW} = P_{(\alpha=1, T_A=4)} \in \left(\frac{1}{5}, 1\right] \quad (14)$$

In summary, the computing efficiency optimization is based on the theoretical evaluation of operator structure, and sharing activation function reduces the consumption of computing resources.

3.2 Memory resource model and efficiency optimization

In the operation of full-pipelined operator structure, parallel reading and writing is a key factor from local memory. The point of memory resource design is how to guarantee parallel access and increase the effective utilization.

A commonly used parallel memory strategy for feature map is shown in Fig. 5, where A_{bit} is the bit width of the feature map. From the different parallelisms of operator, the minimal bit width of the memory cell is $T_R \times T_C \times T_M \times A_{bit}$ in (4), and the depth is shown in Fig. 5.

Similarly, the parallel access of the weight is shown in Fig. 6, where W_bit is the corresponding bit width. The minimal bit width of this memory cell is $T_M \times T_N \times A_{bit}$.

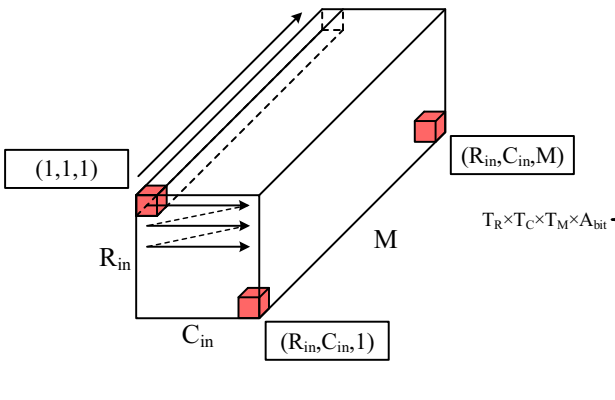
For the memory storage of both feature map and weight, BRAM integrated in an FPGA is usually used as the data buffer. This setup can be flexibly configured with different bit widths to accommodate different data types. Generally, if the bit width is $X_{Bit}(X_{bit} \in \mathbb{N}+)$, the maximum depth of BRAM is:

$$Depth = \begin{cases} 16384, X_{bit} = 1 \\ 8192, X_{bit} = 2 \\ 4096, X_{bit} \in (2, 4] \\ 2048, X_{bit} \in (4, 9] \\ 1024, X_{bit} \in (9, 18] \\ 512, X_{bit} \in (18, +\infty] \end{cases} \quad (15)$$

Based on the maximum depth of BRAM in different bit widths, the numbers of BRAM units consumed by feature map and weight are expressed in (16) and (17), respectively.

$$\hat{B}_A = \left\lceil \frac{T_R \times T_C \times T_M \times A_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{R_{in} \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil \quad (16)$$

$$\hat{B}_W = \left\lceil \frac{T_N \times T_M \times W_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{N \times k \times k \times M}{T_N \times T_M \times Depth} \right\rceil \times 0.5 \quad (17)$$



Addr=0	Addr=1	Addr=⋯	Addr=$\left\lceil \frac{R_{in}\times C_{in}\times M}{T_R\times T_C\times T_M} \right\rceil$-1
(1,1,1)	(T_R+1,T_C+1,T_M+1)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in}-T_C,M-T_M+1)
(1,1,⋯)	(T_R+1,T_C+1,⋯)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in}-T_C,⋯)
(1,1,T_M)	(T_R+1,T_C+1,2T_M)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in}-T_C,M)
(1,2,1)	(T_R+1,T_C+2,,T_M+1)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in}-T_C+1,M-T_M+1)
(1,2,⋯)	(T_R+1,T_C+2,⋯)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in}-T_C+1,⋯)
(1,T_C,T_M)	(T_R+1,2T_C,2T_M)	(⋯,⋯,⋯)	(R_{in}-T_R,C_{in},M)
(2,1,1)	(T_R+2,T_C+1,T_M+1)	(⋯,⋯,⋯)	(R_{in}-T_R+1,C_{in}-T_C+1,M-T_M+1)
(2,⋯,⋯)	(T_R+2,⋯,⋯)	(⋯,⋯,⋯	(R_{in}-T_R+1,⋯,⋯)
(T_R,T_C,T_M)	(2T_R,2T_C,2T_M)	(⋯,⋯,⋯)	(R_{in},C_{in},M)

Fig. 5 Schematic diagram of feature map parallel memory

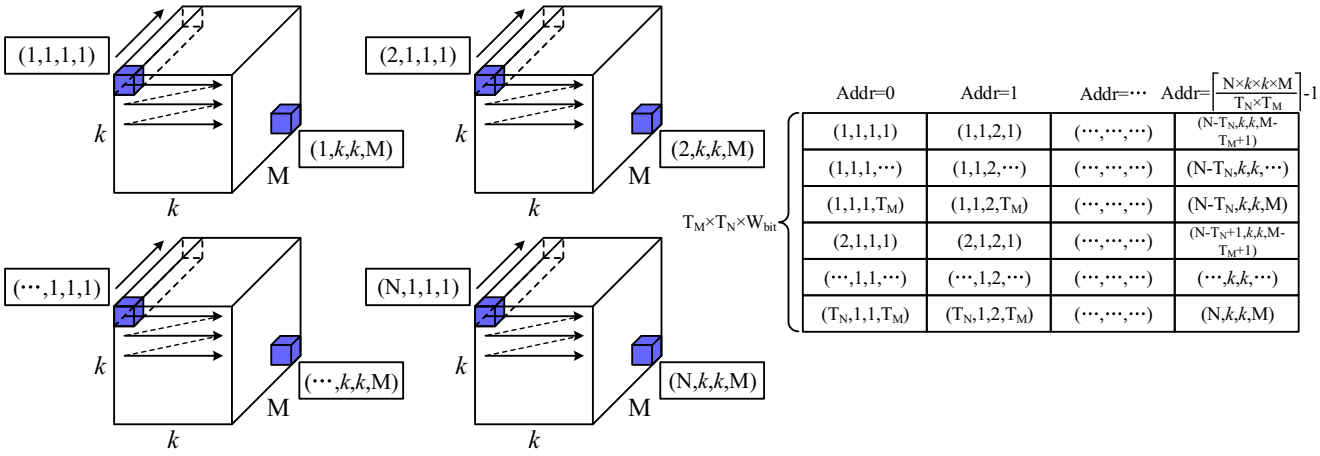


Fig. 6 Schematic diagram of weight parallel memory

When on-chip memory resources match the scale of network, (16) and (17) can be used for the storage of feature map and weight. However, as the scale increases, the entire feature map cannot be buffered in each layer. Moreover, in terms of weight, the memory depth in the shallow layer is much smaller than the maximum depth of (17), which leads to an inefficient utilization of BRAM. Therefore, we combines the hybrid memory to realize optimization of memory efficiency.

In the description of (16), frame buffer is used to save the entire feature map. In the processing of CNN, only part of feature map is essential to complete convolution processing with multiple kernels. Therefore, it is acceptable to buffer only part of feature map. As shown in Fig. 7, row buffer is proposed for the optimization of feature map. Theoretically, buffer needs to store only the same rows of kernel size(K), which is shown in the red part of Fig. 7. Considering the independent reading and writing of BRAM, we use the additional rows of stride S to ensure this mechanism, which is shown in the yellow part of Fig. 7.

Therefore, the expression of BRAM used for feature map in the row buffer model is updated to (18). This equation shows that the row buffer has a higher efficiency than frame buffer. This is the key to reduce the demand of BRAM.

$$\hat{B}_{A-Row} = \left\lceil \frac{T_R \times T_C \times T_M \times A_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{(S+k) \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil \tag{18}$$

Considering the balance between memory and logic resources, we further proposes a hybrid memory based on the depth threshold θ to buffer the feature map, as shown in (19), where the value of θ depends on the network structure and hardware resources. When the depth is less than the threshold, the benefit of row buffer cannot balance the additional logic resources required, so it stays in original mode. In contrary, if row buffer satisfies memory requirement, it is the perfect selection to save memory resource.

$$\hat{B}'_A = \begin{cases} \left\lceil \frac{T_R \times T_C \times T_M \times A_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{R_{in} \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil, & \left\lceil \frac{R_{in} \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil \le \theta \\ \left\lceil \frac{T_R \times T_C \times T_M \times A_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{(S+k) \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil, & \left\lceil \frac{R_{in} \times C_{in} \times M}{T_R \times T_C \times T_M \times Depth} \right\rceil > \theta \end{cases} \tag{19}$$

$$\hat{B}'_W = \begin{cases} 0, & \frac{N \times k \times k \times M}{T_N \times T_M} < \frac{Depth}{4} \\ \left\lceil \frac{T_N \times T_M \times W_{bit}}{X_{bit}} \right\rceil \times \left\lceil \frac{N \times k \times k \times M}{T_N \times T_M \times Depth} \right\rceil \times 0.5, & \frac{N \times k \times k \times M}{T_N \times T_M} \ge \frac{Depth}{4} \end{cases} \tag{20}$$

The goal of weight data optimization is to achieve the efficient usage of BRAM. Generally, the weight of shallow layer has the characteristics of high parallelism and low depth, at far less than the maximum depth of BRAM in (17). This results in insufficient usage of BRAM, as shown in Fig. 8. In terms of the memory type, Look-Up-Table (LUT) RAM is generated with logic resources rather than Block RAM. A hybrid memory for weight data in a shallow layer can effectively solve the problem of insufficient usage. As

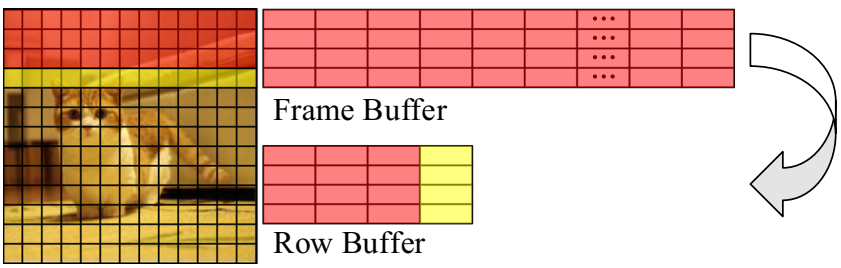


Fig. 7 Schematic diagram of row buffer

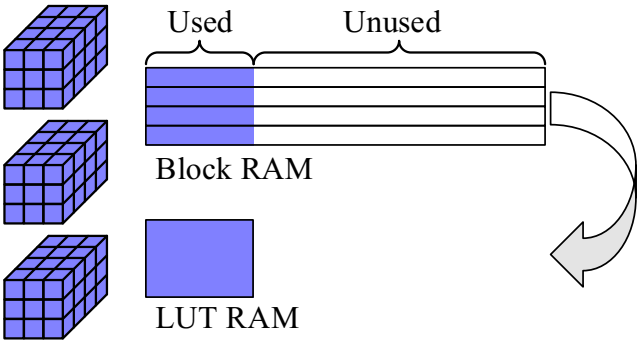


Fig. 8 Schematic diagram of hybrid memory type

shown in (20), when the depth of weight data is less than quarter of the maximum depth, BRAM is replaced by LUT RAM, where the consumption of BRAM is reduced to zero. In other cases, the memory type remains unchanged.

Therefore, the memory efficiency optimization is realized to reduce the requirement of BRAM resources through the use of row buffer for feature map and hybrid memory type for weight data. This method implements the parallel reading of feature map and weight data with minimal memory cost, and also reduces unnecessary memory resource.

4 System architecture and performance estimation method

In this section, we describe the integration of the work of above sections with hardware to form the NoC architecture. Based on this architecture, we estimate the performance model through the analysis of theoretical value, and normalized throughput is used as the benchmark of computing efficiency.

4.1 System architecture overview

Figure 9 shows a schematic diagram of system architecture. It is composed of a heterogeneous processor based on Multiple Processor System on Chip (MPSOC), which mainly includes Processor System (PS) and Programmable Logic (PL). The core part of PS has four Cortex-A53 processors. They are connected to the internal central converter through the cache-coherent interconnect, and then reconnect to the peripheral controllers, such as Secure Digital Input and Output (SDIO), Universal Asynchronous Receiver/Transmitter (UART) and CNN acceleration in PL, where operator and memory organization are combined into Processing Element (PE). Multiple PEs form the main framework of CNN acceleration in the NoC architecture.

In the acceleration of CNN, parameters (such as row, column, input channel, output channel, kernel size and

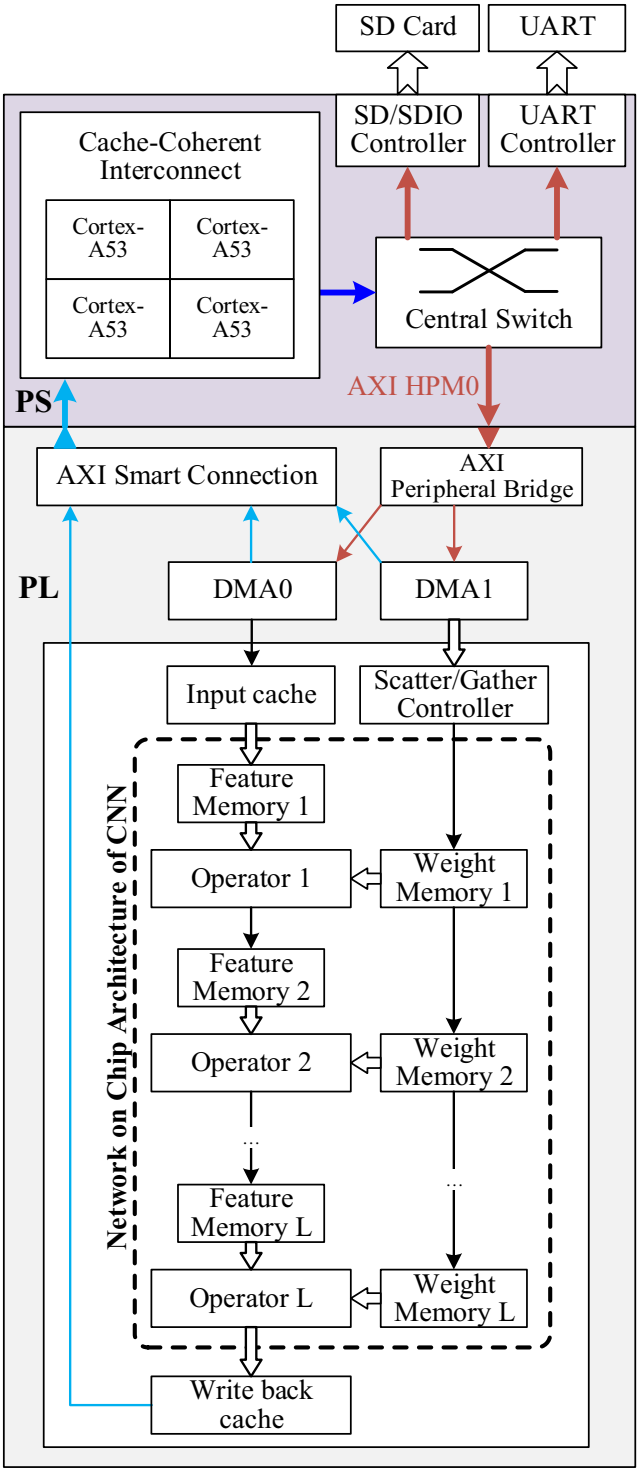


Fig. 9 Schematic diagram of system architecture

stride) are directly solidified into the PE without additional control signals except for clock and reset. Especially for weight data, a dynamic loading is available through Direct Memory Access (DMA). Therefore, in the NoC architecture of Fig. 9, CNN acceleration is divided into a series of PEs. Each PE runs independently in full-pipelined model,

which enables NoC architecture to integrate high computing efficiency inherited from these PEs.

4.2 Performance estimation method

To estimate system performance, we need to statistically analysis the throughput in NoC architecture, which is directly related to the number of DSPs. In Fig. 9, the total number of DSPs in multiple PEs is:

$$N_{DSP} = \sum_{i=1}^{L} T_{Ri} \times T_{Ci} \times T_{Ni} \times \alpha_i \tag{21}$$

In (21), if the clock frequency of DSP is f_{DSP}, then we can obtain the theoretical throughput in (21).

$$TP_{DSP} = 2 \times f_{DSP} \times \sum_{i=1}^{L} T_{Ri} \times T_{Ci} \times T_{Ni} \times T_{Mi} \tag{22}$$

Similarly, the running cycle $\hat{T}_i$ of each PE can be expressed by (23).

$$\hat{T}_i = \left\lceil \frac{R_{outi}}{T_{Ri}} \right\rceil \times \left\lceil \frac{C_{outi}}{T_{Ci}} \right\rceil \times \left\lceil \frac{M_i}{T_{Mi}} \right\rceil \times \left\lceil \frac{N_i}{T_{Ni}} \right\rceil \times k_i \times k_i, i \in [1, L] \tag{23}$$

However, due to the unbalanced payload of multiple PEs, the actual throughput TP' is determined by the maximum cycle, as shown in (24). Ideally, if the cycle of T_i is equal to others, (24) can be seen as a special form of (24).

$$TP' = \frac{2 \times f_{DSP} \times \sum_{i=1}^{L} R_{outi} \times C_{outi} \times M_i \times N_i \times k_i \times k_i}{\max\left(\hat{T}_i\right)} \tag{24}$$

Generally, the Normalized ThroughPut (NTP) is used to evaluate the efficiency of DSP [8, 18, 28], specifically indicating the average operations per DSP slice in each clock cycle, as shown in (25).

$$\xi = \frac{TP}{N_{DSP} \times f_{DSP}} \tag{25}$$

Then, the typical values of normalized throughput with different data types are inferred according to (5) and (25).

$$\hat{\xi} = \begin{cases} 0.2, \text{type} = float32 \\ 0.67, \text{type} = int32 \\ 2, \text{type} = int16 \\ 4, \text{type} = int8 \end{cases} \tag{26}$$

From the analysis of typical value, the computing efficiency is the ratio of the actual value to typical value, which is shown in (27) through substituting (21) and (24) into (25).

$$\hat{\eta} = \frac{2 \times \sum_{i=1}^{L} R_{outi} \times C_{outi} \times M_i \times N_i \times k_i \times k_i}{\max\left(\hat{T}_i\right) \times \hat{\xi} \times \sum_{i=1}^{L} T_{Ri} \times T_{Ci} \times T_{Ni} \times \alpha_i}, i \in [1, L] \tag{27}$$

$$\eta = \frac{2 \times \sum_{i=1}^{L} R_{outi} \times C_{outi} \times M_i \times N_i \times k_i \times k_i}{T' \times \hat{\xi} \times \sum_{i=1}^{L} T_{Ri} \times T_{Ci} \times T_{Ni} \times \alpha_i}, \left\{\forall i \in [1, L], \exists \hat{T}_i \le T'\right\} \tag{28}$$

In (27), the numerator term is a constant value, which is determined by network structure, and the denominator term contains the variable parallel parameters and maximum cycle. If parallel parameters are solved with the exploration of the network and hardware platform, the computing efficiency is related to the maximum cycle. Since additional cycles are requires for data loading and synchronization in the application, if we use T' as the representation of actual cycles, the actual computing efficiency is updated to (28). This means that system performance depends only on the clock frequency and total number of DSPs.

In the performance estimation, the actual computing efficiency is related to the parallel parameters and maximum cycles after implementation. In next section, we will use experimental results to verify (27) and (28).

5 Analysis and comparison of experimental results

The hardware platform in this paper is XAZU3EG based on a heterogeneous processor, that contains 216 BRAM and 360 DSP units. The experimental results are obtained with LeNet based on pruning and MobileNetV1 based on depthwise convolution.

5.1 Resource exploration and implementation

As shown in Algorithm 1, the gradient descent algorithm is used to perform resource exploration. It gives the optimal values for the parallel parameters and running cycles of each PE through mathematical analysis. In the initialization, the network model and available resources are given as a constant condition, and the parallel parameters are initialized to 1. In the analytical solution stage, the index j of the maximum value of PEs is selected as the working space. Then, the resources are allocated in an attempt to

Data: Network Model (R_{outi}, C_{outi}, M_i, N_i, k_i)
Available resource (BRAM=216; DSP=360)
Result: Parallel parameters; Running cycle;
Computing efficiency;

Initialize parallel parameters;
for $i < L$ **do**
  $T_{Ri} = 1$; $T_{Ci} = 1$; $T_{Mi} = 1$; $T_{Ni} = 1$;
end
Analytic solution;
while *User defined boundary* **do**
  Select j satisfies $\hat{T}_j = max(\hat{T}_i), i \in [1, L]$;
  Try $\hat{T}_j = \frac{1}{2}\hat{T}_j$;
  Update N_{DSP}, $\hat{B}'_A$, $\hat{B}'_W$;
  if $N_{DSP} \leq 360$ *and* $\hat{B}'_A + \hat{B}'_W \leq 216$ **then**
    Update T_{Rj}, T_{Cj}, T_{Mj}, T_{Nj};
    Update $\hat{T}_j$;
  else
    Break;
  end
end
Return exploration;
Update TP', $\hat{\eta}$;
return T_{Rj}, T_{Cj}, T_{Mj}, T_{Nj}, $\hat{T}_j$, $\hat{\eta}$;

Algorithm 1 The solution of resource exploration.

reduce the running cycle T_j by half. If the updated resources are in the range of user-defined boundary, the current iteration satisfies analytic solution; otherwise, the iteration is ended. Finally, the analytical results of running cycle and computing efficiency are returned for the following implementation.

The structure of LeNet based on pruning is shown in Table 1. The input is an 8-bit grayscale image(28*28), and 16-bit dynamic fixed-point quantization with a power of 2 is used to reduce the weight and bias data volume. The data width conversion with power of 2 quantization is simplified to logical operations of shifting and truncation, which is advantageous for hardware design. Finally, the classification is generated after FC layer. The exploration results of resources and running cycles are shown in Table 1. In shallow layers, due to the limited parallelism of input and output channels, the column parallelism is used to improve throughput. As the number of layers increases, the column parallelism is reduced to 1, while the channel parallelism increases for layer balance.

Moreover, the structure of MobileNet composed of depthwise convolution is explored in Table 2, where the input is an RGB image(128*128*3), $Conv$ is used for standard convolution, and $Conv\_dw$ is used for depthwise convolution. In this network, Quantization Aware Training (QAT) is used to further reduce computational cost and memory space, where the weight is asymmetrically quantized to 8-bit width with 16-bit offset, and the bias is quantized to 32 bits. Therefore, in each layer, an arithmetic operation is adapted to the bit width conversion, which requires additional DSP units [10]. Compared to LeNet, the number of PEs gradually increases for MobileNet. Therefore, we merge the depthwise and pointwise convolution into a composite PE to reduce the data path, and simplifying the number of PEs is applicable. Since the range of parallel parameters is widely effective for MobileNet, achieving efficient inference with optimization strategy is essential. Finally, the optimized results of DSP' and $BRAM'$ are listed in Table 2.

The resource utilization after the exploration and optimization of both networks is shown in Table 3. Obviously, the utilization of DSP is maintained at a higher level than other resources, while the utilization of BRAM depends on the scale of network structure. Combining Tables 1 and 2 shows that the proposed optimization can reduce the utilization of BRAM and DSP by 43.46% and 27.79%, respectively. Especially in the construction of PEs in shallow layers, the optimization makes both networks more feasible with same hardware resource constraints and reduces power consumption.

Table 1 Parameter table of pruning LeNet

Type / Stride	PE	Input Size	Output Size	Filter Shape	T_R	T_C	T_M	T_N	BRAM	DSP	Cycle
Conv/s1	1	28*28*1	26*26*4	3*3*1*4	1	26	1	4	3.5	104	234
Max Pool/s2		26*26*4	13*13*4	2*2	–	–	–	–	–	–	–
Conv/s1	2	13*13*4	11*11*8	3*3*4*8	1	4	4	8	4	128	297
Max Pool/s2		11*11*8	6*6*8	2*2	–	–	–	–	–	–	–
Conv/s1	3	6*6*8	4*4*16	3*3*8*16	1	1	8	8	2	64	288
Max Pool/s2		4*4*16	2*2*16	2*2	–	–	–	–	–	–	–
Flatten	4	2*2*16	1*64	–	–	–	–	–	–	–	–
FC/s1		1*64	1*10	64*10	1	1	4	1	2	4	160
Total	–	–	–	–			–	–	11.5	300	297

Table 2 Parameter table of MobileNetV1_0.25_128

Type / Stride	PE	Input Size	Output Size	Filter Shape	T_M	T_N	BRAM	DSP	BRAM'	DSP'	Cycle
Conv/s2	1	128*128*3	64*64*8	3*3*3*8	3	8	36	56	1	28	36864
Conv dw/s1	2	64*64*8	64*64*8	3*3*8	1	8	18	40	1	12	36864
Conv/s1		64*64*8	64*64*16	1*1*8*16	8	1		12		12	65536
Conv dw/s2	3	64*64*16	32*32*16	3*3*16	1	4	33.5	20	3	8	36864
Conv/s1		32*32*16	32*32*32	1*1*16*32	8	1		12		12	65536
Conv dw/s1	4	32*32*32	32*32*32	3*3*32	1	8	19	40	2	12	36864
Conv/s1		32*32*32	32*32*32	1*1*32*32	16	1		20		20	65536
Conv dw/s2	5	32*32*32	16*16*32	3*3*32	1	2	17.5	10	4.5	6	36864
Conv/s1		16*16*32	16*16*64	1*1*32*64	8	1		12		12	65536
Conv dw/s1	6	16*16*64	16*16*64	3*3*64	1	4	10.5	20	4.5	8	36864
Conv/s1		16*16*64	16*16*64	1*1*64*64	16	1		20		20	65536
Conv dw/s2	7	16*16*64	8*8*64	3*3*64	1	1	10.5	5	5.5	5	36864
Conv/s1		8*8*64	8*8*128	1*1*64*128	8	1		12		12	65536
5*Conv dw/s1	8∼12	8*8*128	8*8*128	3*3*128	1	2	42.5	50	42.5	30	36864
5*Conv/s1		8*8*128	8*8*128	1*1*128*128	16	1		100		100	65536
Conv dw/s2	13	8*8*128	4*4*128	3*3*128	1	1	12.5	5	12.5	5	18432
Conv/s1		4*4*128	4*4*256	1*1*128*256	8	1		12		12	65536
Conv dw/s2	14	4*4*256	4*4*256	3*3*256	1	1	19	5	19	5	36864
Conv/s1		4*4*256	4*4*256	1*1*256*256	16	1		20		20	65536
Avg Pool/s1	–	4*4*256	1*1*256	4*4	–	–	–	–	–	–	–
FC/s1	15	1*1*256	1*1001	256*1001	4	1	64	4	64	4	64064
Total	–	–	–	–	–	–	283	475	159.5	343	65536

In Table 4, computing efficiency is compared through theoretical cycles and actual cycles, where the theoretical cycles refer to the maximum cycles of Tables 1 and 2. The actual cycles are measured from experimental result after network implementation. To ensure the accuracy of cycle measurement, with the stream of multiple samples, Integrated Logic Analyzer (ILA) is used to check inference value, and Virtual Input Output (VIO) is used to record the output interval. Normally, the value of actual cycle is usually larger than that of theoretical cycle due to data path transmission and synchronization, which results in the actual efficiency being slightly lower than the theoretical efficiency. Especially in MobileNet, even though the construction of activation function requires additional DSPs, the sharing activation function can suppress this negative impact. The comparison of results shows that the decline in computing efficiency is kept within the range of 2%, which shows that the actual computing efficiency satisfies the evaluation in (27) and (28). It also demonstrates the optimization has the ability to release potential computing capabilities.

5.2 Comparison with software platforms

To illustrate the acceleration and optimization effects, we also deploy both networks on other platforms. The results of Frame Per Second (FPS) and power consumption are included in comparison. In terms of inference engine, we

Table 3 Resource utilization of network

Resource	Available	LeNet		MobileNet	
		Usage	Utilization	Usage	Utilization
LUT	70560	45904	65.06%	40715	66.63%
LUTRAM	28800	3627	12.59%	2648	9.19%
FF	141120	64747	45.88%	48331	34.25%
BRAM	216	16	7.41%	179	82.87%
DSP	360	301	83.61%	343	95.28%

Table 4 Comparison of computing efficiency

	LeNet	MobileNet
Theoretical Cycle	297	65536
Theoretical Efficiency	90.03%	62.21%
Actual Cycle	301	66740
Actual Efficiency	88.84%	61.09%

use American National Standards Institute (ANSI) C language instead of Python to reconstruct the inference framework of both networks to minimize the negative impact of dynamic memory allocation, data path transmission and system scheduling. The comparison results are listed in Table 5.

For software platforms, the frequency is the key factor of performance. A higher frequency can increase FPS through the reduction of clock cycle, but additional power may be needed to offset the increase in frequency. In contrast with the improvement in FPS, although FPGA has a lower frequency, it fully schedules limited computing resources and utilizes the potential parallelism. Moreover, the optimization can further improve the efficiency of resources. FPGA-based platform has a better performance in terms of FPS. In addition, the power of FPGA is lower than that of software platforms. Therefore, power energy ratio is higher than others. With the same power constraints, FPGA-based platform can realize more computing operations than other software platforms.

5.3 Comparison with related works

A comparison of resource utilization with related work is given in Table 6 based on the Modified National Institute of Standards and Technology (MNIST) database, as well as the summary of network architecture. The recent work of [23] uses distribution and block memory types to explore the systematic realization of Multilayer Perceptron (MLP) and CNN. For the acceleration architecture, a single accelerator is reused for multiple layers in [23]. In contrast, NoC architecture based on multiple accelerators is implemented for high throughput in proposed method. Therefore, the utilization of DSP is much higher than that of [23]. For BRAM, [23] is depends on the memory type. Most BRAM units are allocated in block mode, and it is saved in distribution mode, which reflects resource utilization with different memory types. Hybrid memory type combines the advantages of both optimization and extends it to multiple accelerators. This method not only improves the peak throughput, but also balances logical resources and block memory. It is more suitable for embedded CNN acceleration design.

The comparison of efficiency with related works is shown in Table 7. The comparison includes platform, network parameters, resource utilization, throughput and power efficiency. The utilization of DSP in both networks is greater than that of related works, which enables the acceleration to tap the potential computing power. For the computing efficiency η, the implementation of LeNet reaches 88.84%. Moreover, due to the inherent imbalance between the operation of depthwise and pointwise convolution, the efficiency of depthwise is lower than pointwise. This phenomenon is also reflected in [2], but that study does not describe and optimize the data width conversion between layers. Under the premise of data conversion between layers, when implementing depthwise convolution, the computing efficiency of MobileNet in this paper is higher than [2].

We show a comparison of power and power efficiency in Table 7, where the power of FPGA is measured with the implementation of "*power_opt_design*" to maximize power savings. In our design, the power of MobileNet is lower than LeNet due to throughput. Benefiting from the optimized design in terms of memory efficiency, there is no requirement for external memory in the acceleration architecture. This also saves power due to the lack of memory interface with external components. Therefore, compared with that of related works, the power is optimized in our experimental result. According to Table 7, the power values of MobileNet and LeNet are reduced to 0.49W and 0.69W, and the power efficiency values are increased to 85.51GOPS/W and 77.25GOPS/W, respectively. Especially for high-power [6] and middle-power [16], using lightweight CNN is more suitable for on micro and power sensitive scenarios.

Table 5 Comparison with software platform

Platform	Type	Frequency	FPS		Power(W)
			LeNet	MobileNet	
Ryzen 3700x	CPU	3.6GHz	36941	385	65
XAZU3EG (ARM)	ARM	1.2GHz	1179	35	2.09
XAZU3EG (FPGA)	FPGA	100MHz	332225	1498	0.69[a]/0.49[b]

[a]Power of LeNet. [b] Power of MobileNet

Table 6 Comparison of resource utilization with related work

	[23]				Ours
Network Type	MLP (784-512-512-10)		CNN (LeNet-5)		CNN (Modified LeNet-5)
Memory	Block	Distribution	Block	Distribution	Hybrid
LUT	3080	85886	42064	43965	45904
FF	1040	1368	37344	30568	64747
BRAM	175	3.5	165	1	16
DSP	3	3	3	3	301

Experimental results show that the acceleration architecture is not only suitable for different convolution types, but also maintains higher computing and power efficiency. Benefiting from the resource optimization strategy and estimation model, the potential computing capabilities can be fully tapped through the exploration of limited resources. In the future, this study can provide a reference for subsequent acceleration research on large-scale.

6 Conclusion

In this paper, multiple node arrays with a full-pipelined operator structure are proposed to maintain high computing efficiency for micro and power sensitive scenarios, and this design is applicable for different convolution types. DSP-based computing efficiency optimization with sharing activation function is proposed. This suppresses the negative impact of data conversion on computing efficiency. BRAM-based memory efficiency optimization reduces resource requirements through the use of row buffer and hybrid memory. To maximize resource utilization, a theoretical model of computing and memory resources is constructed to explore parallel parameters and running cycles. The board-level verification of NoC architecture without external memory achieves 53.3GOPS with pruned LeNet and 41.9GOPS with MobileNet, where the normalized throughput of both networks reaches 1.78 and 1.22 operations per DSP slice in each clock cycle. In terms of efficiency, the computing efficiency is 88.84% and 61.09%, respectively, and the power efficiency of board verification achieves 85.51GOPS/W and 77.25GOPS/W, which is suitable for micro and power sensitive scenarios. However, due to the extreme constraint of available resources on FPGA platform, it fundamentally influences on the scale of the accelerated network, such as trained parameters and convolutional layers. Especially when encountering out-of-scale networks, future research on acceleration strategies and hardware architectures should be explored for computing efficiency and memory hierarchy.

Table 7 Comparison of efficiency with related works

	[6]	[6]	[2]	[16]	[16]	Ours	Ours
Platform	GX1150	GX1150	10AS066N	VX690T	VX690T	XAZU3EG	XAZU3EG
Network Type	Custom	Custom dw	MobileNetV2	AlexNet	LeNet	MobileNetV1	LeNet
f_{DSP} (MHz)	150	180	133	100	100	100	100
Precision (bit)	16	16	16	16/8	16/8	16/8	16
BRAM	–	–	1844	1021	277	179	16
Used DSP	760	712	1278	2872	2907	343	301
Total DSP	1518	1518	1687	3600	3600	360	360
Utilization	50.07%	46.90%	75.76%	79.78%	80.8%	95.28%	83.61%
TP (GOPS)	87.50	98.91	170.6	445.6	424.7	41.9	53.3
NTP (OP/DSP/freq)	0.77	0.77	1.00	1.55	1.72	1.22	1.78
η	38.38%	38.58%	50.18%	77.58%	73.05%	61.09%	88.84%
Power (W)	8.69	8.52	–	24.8	25.2	0.49	0.69
Power Efficiency (GOPS/W)	10.07	11.61	–	17.97	16.85	85.51	77.25

Acknowledgements This work was supported in part by the National Natural Science Foundation of China(NSFC) under Grant 62171156.

References

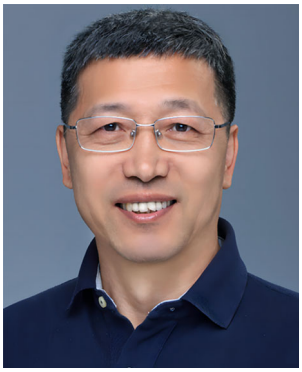
1. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, Santamaría J, Fadhel MA, Al-Amidie M, Farhan L (2021) Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. pp 1–74. Springer
2. Bai L, Zhao Y, Huang X (2018) A cnn accelerator on fpga using depthwise separable convolution. IEEE Trans Circ Syst II: Express Briefs 65(10):1415–1419
3. Bianchi V, Bassoli M, Lombardo G, Fornacciari P, Mordonini M, De Munari I (2019) Iot wearable sensor and deep learning: an integrated approach for personalized human activity recognition in a smart home environment. IEEE Internet Things J 6(5):8553–8562
4. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 1251–1258
5. Cong J, Xiao B (2014) Minimizing computation in convolutional neural networks. In: International conference on artificial neural networks, pp 281–290. Springer
6. Ding W, Huang Z, Huang ZA, Tian LA, Wang HA, Feng SA (2019) Designing efficient accelerator of depthwise separable convolutional neural network on fpga. J Syst Archit 97:278–286
7. Gilan AA, Emad M, Alizadeh B (2019) Fpga-based implementation of a real-time object recognition system using convolutional neural network. IEEE Trans Circ Syst II: Express Briefs 67(4):755–759
8. Huang W, Wu H, Chen Q, Luo C, Huang Y (2021) Fpga-based high-throughput cnn hardware accelerator with high computing resource utilization ratio. IEEE Trans Neural Netw Learn Syst PP(99):1–15
9. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (ICML), pp 448–456. PMLR
10. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 2704–2713
11. Jafari A, Ganesan A, Thalisetty CSK, Sivasubramanian V, Oates T, Mohsenin T (2018) Sensornet: a scalable and low-power deep convolutional neural network for multimodal data classification. IEEE Trans Circ Syst I: Regular Papers 66(1):274–287
12. Kala S, Jose BR, Mathew J, Nalesh S (2019) High-performance cnn accelerator on fpga using unified winograd-gemm architecture. IEEE Trans Very Large Scale Integration (VLSI) Syst 27(12):2816–2828
13. Liao S, Samiee A, Deng C, Bai Y, Yuan B (2019) Compressing deep neural networks using toeplitz matrix: algorithm design and fpga implementation. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 1443–1447. IEEE
14. Lin M, Ji R, Wang Y, Zhang Y, Zhang B, Tian Y, Shao L (2020) Hrank: filter pruning using high-rank feature map. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 1529–1538
15. Liu X, Yang J, Zou C, Chen Q, Yan X, Chen Y, Cai C (2021) Collaborative edge computing with fpga-based cnn accelerators for energy-efficient and time-aware face tracking system, pp 252–266. IEEE
16. Liu Z, Dou Y, Jiang J, Xu J, Li S, Zhou Y, Xu Y (2017) Throughput-optimized fpga accelerator for deep convolutional neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 10(3):1–23
17. Ma N, Zhang X, Zheng HT, Sun J (2018) Shufflenet v2: practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp 116–131
18. Ma Y, Cao Y, Vrudhula S, Seo Js (2018) Automatic compilation of diverse cnns onto high-performance fpga accelerators. IEEE Trans Comput-Aided Des Integrated Circ Syst 39(2):424–437
19. Ma Y, Cao Y, Vrudhula S, Seo Js (2018) Optimizing the convolution operation to accelerate deep neural networks on fpga. IEEE Trans Very Large Scale Integration (VLSI) Syst 26(7):1354–1367
20. Ma Y, Cao Y, Vrudhula S, Seo JS (2019) Performance modeling for cnn inference accelerators on fpga. IEEE Trans Comput-Aided Des Integr Circ Syst 39(4):843–856
21. Mathieu M, Henaff M, LeCun Y (2014) Fast training of convolutional networks through ffts. In: 2nd International Conference on Learning Representations, ICLR 2014
22. Moolchandani D, Kumar A, Sarangi SR (2021) Accelerating cnn inference on asics: a survey. J Syst Archit 113:101887
23. Mukhopadhyay AK, Majumder S, Chakrabarti I (2022) Systematic realization of a fully connected deep and convolutional neural network architecture on a field programmable gate array. Comput Electr Eng 97:107628
24. Palossi D, Conti F, Benini L (2019) An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs. In: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp 604–611. IEEE
25. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 4510–4520
26. Wang J, Lin J, Wang Z (2017) Efficient hardware architectures for deep convolutional neural network. IEEE Trans Circ Syst I: Regular Papers 65(6):1941–1953
27. Wang S, Ananthanarayanan G, Zeng Y, Goel N, Pathania A, Mitra T (2019) High-throughput cnn inference on embedded arm big. little multicore processors. IEEE Trans Comput-Aided Des Integr Circ Syst 39(10):2254–2267
28. Yu Y, Wu C, Zhao T, Wang K, He L (2019) Opu: an fpga-based overlay processor for convolutional neural networks. IEEE Trans Very Large Scale Integr VLSI Syst 28(1):35–47
29. Zeng H, Chen R, Zhang C, Prasanna V (2018) A framework for generating high throughput cnn implementations on fpgas. In: Proceedings of the 2018 ACM/SIGDA international symposium on field-programmable gate arrays, pp 117–126
30. Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J (2015) Optimizing fpga-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays, pp 161–170
31. Zhang Y, Li X (2020) Fast convolutional neural networks with fine-grained ffts. In: Proceedings of the ACM international conference on parallel architectures and compilation techniques, pp 255–265

Ruidong Wu received a B.S from the Southwest University of Science and Technology, Mianyang, China, in 2014. He received a M.Sc. from the Taiyuan University of Technology, Taiyuan, China, in 2017.

He is currently pursuing a Ph.D. degree with the Department of Measurement and Control Engineering at the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China. His current research interests include high performance acceleration for ultimate edge computing.

Bing Liu received a B.S., a M.Sc., and a Ph.D. from the Harbin Institute of Technology (HIT), Harbin, China, in 2005, 2007 and 2012, respectively.

From 2015 to 2016, he was a Visiting Scholar at the University of Wisconsin-Madison, Madison, WI, USA. He is currently an Assistant Professor in the School of Electronics and Information Engineering, HIT. His research interests include automatic test technologies, machine learning for image processing and FPGA-based computing.



Ping Fu received a B.S. degree from the Department of Radio Engineering, University of Science and Technology of China, Hefei, China, in 1989, and a M.Sc. in communication and electronic systems and a Ph.D. in measurement technology and instrumentation from the Harbin Institute of Technology (HIT), Harbin, China, in 1992 and 1999, respectively.

He is currently a Full Professor in the Department of Measurement and Control Engineering, School of Electronics and Information Engineering, HIT, and also the Vice Director of the Automatic Test and Control Institute. His current research interests include automatic test technologies, compressive sensing, machine learning, image processing and FPGA-based computing.

Haolin Chen received a B.S. from the Harbin Institute of Technology, Harbin, China, in 2017.

He is currently pursuing a M.Sc. degree with the Department of Measurement and Control Engineering at the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China. His current research interests include visual object detection, visual object tracking and algorithm design for resource constrained platform.