Design and Implementation of Convolutional Neural Network Accelerator Based on RISCV

Yangyang He*
1 School of Computer Science, Beijing University of Posts and Telecommunications, BEIJING, 100876, China
*Corresponding author’s e-mail: hyy@bupt.edu.cn

Abstract. Internet of Things devices are faced with ever-increasing amounts of data, and they can no longer only do data collection as they did in the past, and hand over computing tasks to servers on the cloud. The growing computing requirements in the field of Internet of Things and the diversity of its scenarios put forward configurable and flexible customization requirements for customized processors. This paper conducts in-depth analysis and research on configurable customized processors, and analyzes related work in the field of RISC-V chips and deep convolutional neural networks in the field of device-side optimization at home and abroad. Based on this, this paper proposes an instruction and hardware design of a customizable deep convolutional neural network accelerator attached to Rocket-Chip Generator based on the RISC-V modular instruction set.

1. Introduction
As electronic components are gradually approaching their physical limits, the development speed of processors has slowed in recent years. The stagnation of processor development and the limitation of network bandwidth promote the transition from cloud computing to edge computing. The excess performance of IoT devices is used for front-end data processing and analysis, and then transmitted back to the cloud for processing to relieve the pressure on the cloud. Microprocessors based on RISC-V can be constructed relatively quickly and give full play to the characteristics of RISC's simplified instruction set. It can be customized to meet the diverse needs of IoT devices.

At present, the research of neural network accelerators is mainly divided into three directions: 1. For the CPU/GPU platform, due to its strong computing power and sufficient memory bandwidth, it is mainly to optimize the software of the neural network code itself and use compression, Pruning, quantization and other methods; 2. ASIC: specifically design specific hardware circuits for specific tasks, and implement software algorithms through hardware to accelerate neural networks; 3. FPGA: use FPGA programmability to design hardware structures To adapt to the algorithm, FPGA has a higher energy efficiency ratio than CPU/GPU; although the energy efficiency ratio is not as good as ASIC, FPGA provides a smaller cost and more flexible design.

Many companies have open sourced neural network end-to-side computing frameworks developed for mobile and embedded devices, including Tencent’s NCNN, Alibaba’s MNN, Xiaomi’s MACE and OPEN AI LAB’s Tengine.[1,2,3,4]. These frameworks mainly use Winograd transformation to optimize convolution operations, and use the ARM-based NEON instruction set for parallel operations to optimize data structure, memory management, and scheduling among multiple cores. Research on acceleration of convolutional neural networks is mainly focused on two aspects: improving parallelism; optimizing memory utilization.
2. Accelerated analysis of convolutional neural networks

Studies have shown that in convolutional neural networks, the longest time-consuming operation is the convolution operation, which accounts for almost 90% of the time of the neural network.[5] The most important thing to study the acceleration of convolutional neural network is to accelerate the convolution operation. At present, the main methods for accelerating the convolution operation are: 1. Change the order of loop nesting, once load calculation is completed, improve the data reuse rate, and reduce the number of memory accesses; 2. Unroll the loop, use the underlying data to be independent of each other, and make the calculations parallel.

2.1. Addition tree

The addition tree is a typical hardware acceleration method for loop unrolling. As shown in Figure 1, expand the 6-fold loop in the convolution operation. The convolution kernel is expanded, and 8 convolution operations of 4 convolution kernels are performed at the same time (each convolution kernel corresponds to the input feature map channel 2). The convolution kernel is expanded because the input feature maps for the 8 convolution operations are the same, and there is no dependency between them, so the double loop is reduced by increasing the calculation unit, and the theoretical speedup is 8.

Due to the introduction of parallel computing for the addition tree, it increases the demand for hardware resources. An addition tree requires 8 multipliers and 5 adders. The disadvantage of additive trees is that due to parallel computation the data is not fully utilized and the additive tree needs to be reloaded after each run, which increases the bandwidth requirement. Assuming that the convolution kernel is K×K, the input feature map size is M, and the convolution step stride is 1, the amount of data that needs to be read for a convolution calculation of a feature map in the original loop is (M-K+1)²×K², much larger than the amount of data to load a picture M². After using the addition tree, the amount of data that needs to be read to complete one layer of convolution as shown in Figure 1 is (M-K+1)²×K², while the original loop requires (M-K+1)²×K²×2, which is only half the amount of data, and there is still a lot of room for optimization compared with K².

![Figure 1. Addition tree.](image)

2.2. GEMM optimization

The general form of General Matrix Multiplication (GEMM) is C=A×B. The GEMM optimization method is to divide the output into 4×4 sub-blocks to improve the reuse of data. At the same time, it optimizes by using a large number of registers, reducing memory access, eliminating pointer calculations, reorganizing the memory to make the address continuous, etc., and obtained 7 Times performance improvement.[6]
Figure 2. GEMM Optimization.

As shown in the left of Figure 2, expand K, the calculation of C becomes the result of calculating 4/K C at a time. The pseudo code is shown in the right of Figure 2. Observing the code, we can find that the innermost loop is on the basis of AB reuse, and it has almost no effect on C. C is optimized to the next level loop, and the number of memory accesses can be reduced to 1/4MNK+1/4MNK+(4×4)×(1/4M×1/4N), namely 1/2M NK+MN times. Compared with the original 4MNK, the number of memory accesses has been optimized by nearly 8 times.

2.3. Systolic array

The fixed-weight systolic array is shown in Figure 3. The weight is fixed in each basic calculation unit, and the input data flow formula passes through each calculation unit, so as to achieve the purpose of reusing the input data. The process of convolution operation through the fixed-weight systolic array is as follows:

- First, read in the weight data to a fixed position in the calculation unit.
- Next, through 4 cycles of data flow, the input feature map data is read in to fill up the 16 calculation units.
- Finally, start the calculation, one bit of data flows in each cycle, and the output result is calculated.

The systolic array with fixed weights requires K² multiplication operations and K²-1 addition operations to calculate a convolution kernel of size K. The multiplication operation uses MAC resources and is also completed in one cycle. It can be calculated that the original convolution kernel operation requires 2×K²-1 cycles, and the use of fixed-weight systolic arrays requires 1 cycle, and the calculated speedup ratio is 2×K²-1. For a feature map of size M, using this convolution kernel requires (M-K+1)² cycles to perform the convolution operation with Stride=1. However, because the input feature map needs to be reloaded when changing lines, and there is time consumption for filling data at the beginning of each line. Therefore, the actual convolution of a feature map needs to consume (M-K+1)×(M-K+1)×1+(M-K+1)×1×M cycles, that is (M-K+1)×2M cycles, are greatly reduced compared to the (M-K+1)³×K² cycles of direct calculation.
3. Accelerator design

The convolutional neural network accelerator designed in this paper uses GEMM optimized weight fixed systolic array method, which combines GEMM optimization with weight fixed systolic array. The overall architecture of the deep neural network accelerator is shown in Figure 4. The purple part in the figure is the accelerator, which is connected to RocketCore through ROCC. The accelerator accesses L2Cache and Dram through the DMA module to read and write back feature map data. The accelerator is composed of three parts, the control module, the calculation module and the cache module.

![Figure 4. accelerator architecture.](image)

The matrix calculation module is shown in Figure 5. The most basic arithmetic unit PE is responsible for completing a simple multiplication and addition operation, in which a weight buffer register is set to fix the weight value. The basic PE arithmetic units together constitute the basic arithmetic unit Tile in GEMM, which is then connected to form the entire systolic array. A Tile here corresponds to a basic operation block in the previous GEMM optimization. Registers are set between the tiles to organize the tile pipeline for pulsation calculation.

![Figure 5. Systolic array.](image)

4. Conclusion

First, synthesize the Rocket-Chip convolutional neural network accelerator code, check the power consumption and hardware resource consumption as shown in Figure 6.
Figure 6 Systolic array.

Then connect the Genesys2 development board through JTAG, access the DDR through the FPGA internal module, use vivado to call the relevant interface to access the DDR, and write the test program, input feature map, and yolov3-tiny weight into the DDR. Finally, the convolutional neural network accelerator operates, and the operation is completed to retrieve the result for analysis. Analysis of the results shows that the detection time of each picture is about 0.024s, about 42fps. In this environment, 64-bit BOOM dual-core and the convolutional neural network accelerator designed in this paper are used, so the hardware resource consumption is more than the paper [7].

Acknowledgments
First of all, I would like to thank my teacher. Without his guidance and help, there would be no article. Then I would like to thank my classmates for their help during the formation of this article. Finally, I would like to thank my relatives for their encouragement and support during this period.

References
[1] Oaid. (2020) Tengine. https://github.com/OAID/Tengine.
[2] Tencent. (2020) NCNN. https://github.com/Tencent/ncnn.
[3] Xiaomi. (2020) MACE. https://github.com/XiaoMi/mace.
[4] Alibaba. (2020) MNN. https://github.com/alibaba/MNN.
[5] Chen Y H, Krishna T, Emer J S, et al. (2017) Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1): 127-138.
[6] Prof. Robert van de Geijn. how-to-optimize-gemm. https://github.com/flame/how-to-optimize-gemm/wiki
[7] Wang Zifeng. (2020) Development of Artificial Intelligence Chip Software Stack and Algorithm Research. Hangzhou Dianzi University.