Loosely Coupled Architecture in Flexible RISC-V CPU and Configurable CNN Accelerator

Yaofeng Hou, Kaijian Zeng

***Abstract*— This paper describes a methodology for designing a deep learning accelerator system, incorporating RISC-V and CNN capabilities within a loosely coupled architecture(LCA), had been presented to enhance inference performance on edge devices, achieve lower power consumption, and expedite response times. First, a microarchitecture had been designed for cooperative operation between the main processor and the deep learning accelerator, and efficient neural network inference had been enabled through a customized instruction set. Second, flexible configuration and scalability strategies had been adopted, allowing the accelerator to accommodate various neural network models and application requirements. Lastly, widely-used convolutional neural network models such as ResNet-50, YOLOv4-Tiny, and BiSeNet v1 had been selected and rapidly deployed on the system. Experiments had been conducted on the XC7K410T board, demonstrating the synergy advantages between the accelerator and the RISC-V processor. Specifically, the system achieved processing speeds up to 871.1 GOP/s and computational efficiencies up to 96.79 GOP/s/W.**

***Index Terms*— accelerator, CNN, LCA, RISC-V**

I. INTRODUCTION

Under the surge of artificial intelligence, machine learning within edge computing encounters dual challenges: the rapid evolution of emerging applications underscores the necessity for generalizability, while local decision-making imposes increasingly stringent demands on performance and power consumption. Edge devices necessitate real-time responsiveness and intelligent decision-making capabilities, with interaction modes tending to diversify. Convolutional Neural Network (CNN), as potent machine learning techniques, have significantly improved the accuracy of tasks such as image classification [1], object detection [2], and semantic segmentation [3]. However, this enhanced performance comes at the expense of increased computational complexity and resource consumption.

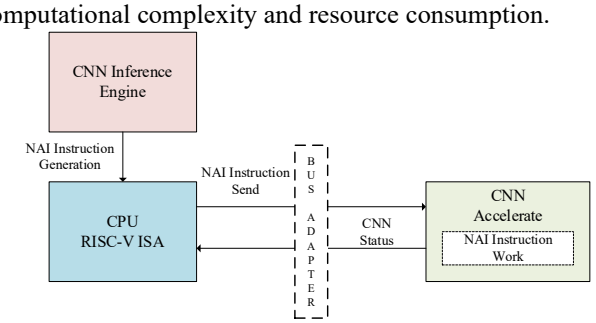

Fig. 1.LCA diagram

Hardware acceleration technologies help edge devices respond in real-time by offloading computational tasks to dedicated neural network accelerators. The RISC-V Instruction Set Architecture (ISA) [4] is a popular choice for deploying neural networks at the edge because it is open and scalable. RISC-V processors and neural network accelerators can use either Loosely Coupled Architecture (LCA) or Tightly Coupled Architecture (TCA) [5]. As shown in Fig.1, LCA is a heterogeneous architecture. The CPU utilizes the RISC-V instruction set and sends NAI instructions, generated by the inference engine, to the accelerator for neural network inference acceleration. Based on the results returned by the accelerator, the CPU decides on the next steps. LCA is flexible and scalable. It allows the configuration of the processor and accelerator to be adapted to the application scenario. It also makes it easier to integrate new components and reduces the complexity of system integration. This is a significant advantage for edge computing applications that require rapid iteration and continuous optimisation.

The contributions of this paper are as follows:

1) A modular Neural Network Acceleration Instruction (NAI) set had been introduced, upon which a convolutional neural network accelerator had been designed.

2) A loosely coupled architecture (LCA), featuring an RV32IM RISC-V processor alongside an NAI-powered neural network accelerator, had been proposed to investigate a neural network heterogeneous computing system with reduced power consumption and enhanced inference speed.

3) Experimental validation had been carried out on the XC7K410T board, showcasing the synergistic benefits of the accelerator operating in conjunction with the RISC-V processor under the LCA.

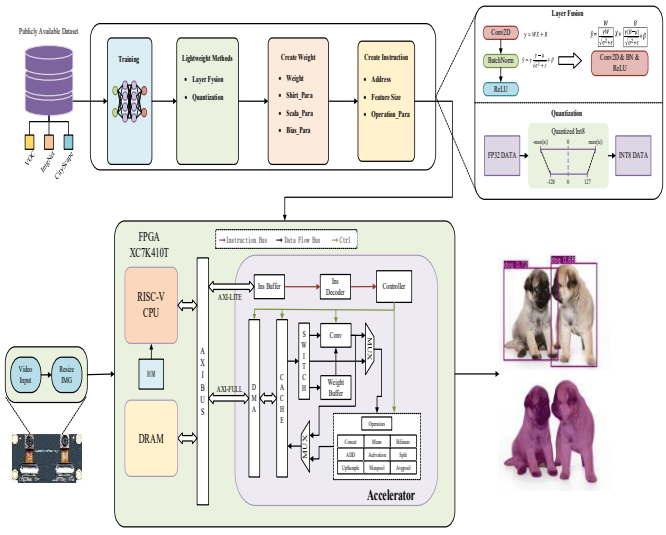

Fig. 2. System Structure Diagram

Manuscript received March 09, 2025

Yaofeng Hou, School of computer science and technology, Tiangong University, Tianjin, China.

Kaijian Zeng, School of computer science and technology, Tiangong University, Tianjin, China.

II. Related Work

Deployment of neural network models has advanced FPGA-based deep learning accelerators. Zhang et al. [6] used pipelined designs on the ZYNQ-7020 to deploy the YOLOv4-Tiny neural network via HLS, speeding up inference times. Guo et al [7] developed a DeepLabv3+ResNet18 accelerator on a Xilinx Virtex-7 XC7VX690T board with outstanding performance and accuracy. These works show that FPGA is great for accelerating CNN inference on edge devices. However, most research has focused on performance enhancements, ignoring generality and scalability. Heterogeneous computing architectures, which integrate processors with RISC-V instruction sets and accelerators featuring custom instruction sets, have demonstrated impressive performance in accelerating inference tasks and model reconfigurations. Mani V.R.S. [8] employed ZYNQ boards with ARM hardware cores to meet the application requirements for processor-controlled systems and data routing. However, the proprietary nature of ARM cores limited the system's flexibility. RISC-V has gained attention for its openness and modular instruction set that can be adapted to various application scenarios. By deploying RISC-V as a soft core on FPGA boards, new opportunities for general-purpose deep learning accelerators have emerged. Wu et al. [9] integrated the accelerator as a co-processor onto the RISC-V CPU core, achieving high acceleration ratios for convolutional computations through extended instruction sets. However, tightly coupled architectures are highly dependent on specific models, which can lead to significant decreases in accelerator efficiency when handling different models or algorithms. To accommodate new functional requirements, extensive reconfigurations of both the CPU and accelerator are often necessary. Alejandra Sanchez-Flores [5] offered comprehensive insights into LCA and TCA, supported by empirical data showcasing favorable power consumption and throughput for LCA centered on RISC-V processors.

III. SYSTEM STRUCTURE

A. System Structure Description

The independence of the LCA facilitates a more flexible and efficient implementation on both the software and hardware sides. This decoupling of software and hardware components also enhances their ability to collaborate effectively. In this architecture, the RISC-V processor serves as the main control unit, managing low-complexity computations and coordinating communication between system components. Meanwhile, the CNN accelerator focuses on efficiently executing deep neural network computations. To improve inference speed, parallel processing and pipeline optimization techniques are utilized. Given the high computational demands of CNN, the accelerator operates at a higher frequency to expedite inference, while the RISC-V processor operates at a lower frequency to manage the control flow. This dual-frequency approach enhances the overall performance and maintainability of the system. The system architecture of the LCA, featuring RISC-V CPUs and deep learning accelerators, is illustrated in Fig.2. The entire inference process of the neural network is controlled by a combination of RISC-V instructions for process control and NAI for computational tasks.

The RISC-V processor retrieves instructions and parameters from ROM, transfers weight data to DRAM via the AXI bus, and sends convolutional layer operation commands to the accelerator through the AXI-Lite bus. These commands are stored in the accelerator's instruction cache. Upon receiving a start or completion signal from the previous layer, the instruction cache releases the current layer's instructions to the decoder. The decoder then configures the compute kernel, memory accesses, and cache control accordingly.Under the guidance of the controller, the DMA extracts necessary parameters and image data from the DRAM and sends them to the computational unit of the accelerator to perform the convolution operation. Once completed, the output data can be directed to subsequent processing stages, such as pooling and activation, minimizing unnecessary data write-backs to the DRAM. This approach dramatically reduces data transfer latency and energy consumption, improving overall inference efficiency.To minimize resource utilization, we adopted a lightweight algorithm design approach focused on reducing computational load, parameter count, and actual runtime. Specifically, we employed weight quantization and operator fusion. Weight quantization reduced the model's parameter count while maintaining accuracy, thereby enhancing computational efficiency. Batch normalization (BN) fusion addressed the vanishing gradient problem by enabling a single computation to complete the Conv+BN+ReLU layers in hardware. This also reduced power consumption on DRAM, positively impacting the system's energy efficiency ratio.

B. CNN Inference Engine

Different neural network models typically have distinct compositions, as illustrated in Table 1. For example, ResNet-50, YOLOv4-Tiny, and BiSeNet v1 exhibit substantial differences in operator composition. Even when operators are shared, their computational parameters can vary. To accommodate the support of various neural network models, a CNN inference engine was proposed. This engine generates data and instructions corresponding to the microarchitecture based on the computational graph.Under the LCA, all computational tasks are offloaded to the accelerator, leaving the CPU solely for control purposes.

Table 1 Network model information

	ResNet-50	YOLOv4-Tiny	BiSeNet v1
CONV	√	√	√
MaxPooling	√	√	√
AvgPooling	√	-	√
Concat	√	√	√
Upsample	-	√	√
LeakyRelu	-	√	√
Sigmoid	-	√	√
Mul	-	-	√
ADD	√	√	√
Split	-	-	√
Input Size	224x224	416x416	640x640
Conv layer	49	21	34
Operations	8.9	6.8	127.7
Kernel Size	3 × 3, 7 × 7	1 × 1, 3 × 3	1 × 1, 3 × 3
Accuracy(FP32)	Top-1 75.08%	Iou≥ 0.7 56.92%	mIoU 68.24%
Accuracy(INT8)	Top-1 74.56%	Iou≥ 0.7 56.58%	mIoU 67.82%

As shown in Fig.3(a), the inference engine has two phases: translation and optimization. The translation phase extracts

necessary information from the model, performs operator fusion on closely related layers in the CNN to reduce computation, and quantizes the kernel weights with negligible loss of accuracy. The generated quantized data is then organized into an intermediate representation (IR). The processed weights are reordered according to the results of the network slicing and optimization phases.The optimization phase parses the translated IR to maximize throughput by slicing the network. Finally, the optimized demapping generates Neural Network Acceleration Instruction (NAI).

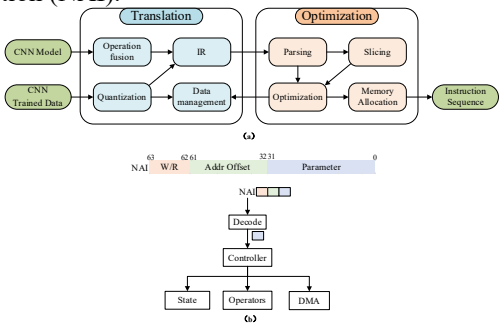

Fig.3. Engine and instruction flow

C. NAI Instruction

NAI was defined in this study as a specialized 64-bit instruction set architecture. The [63,32] bit range of this architecture was designated as the Opcode, which determined the instruction type through recognition. As depicted in Fig.3(b), the Opcode consisted of read/write operations and address information, with the Decode module tasked with resolving the corresponding read/write addresses based on the Opcode. Subsequently, the module transferred the [31,0] bit parameters of NAI to the Controller. NAI was subdivided into three primary categories based on its function and purpose: Parameter Type (P-Type), Control Type (C-Type), and Load/Store Type (L-Type) instructions. The number of bits occupied by each instruction type and the details of their information were detailed in Fig.4.

P-Type is the type of core instruction required to configure a specific computational task within the accelerator and is used to convey basic configuration information. This includes key configuration information such as image size, step size, input/output channels, bias, scale, and weight count. By controlling these parameters, flexible support can be provided for various hierarchical structures of the CNN.

C-Type is an tool for coordinating operations within the accelerator and controlling the flow of instructions.The C-Type instructions include functions such as initiating specific operator operations, synchronizing the accelerator state, and reading the accelerator state to determine if a computation is complete.

L-Type instructions specify read and write operations on the data, ensuring that the accelerator 's processing units are able to fetch the input data and store the results of the computation correctly. L-Type allows the accelerator to precisely control the data flow and support complex memory access patterns, which are particularly important for efficient weight and feature map storage, optimized memory access modes, and reduced memory latency.

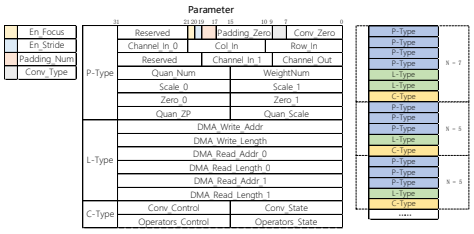

Fig. 4. Instruction unit detail

NAI incorporated an event-based triggering mechanism. As illustrated in Fig.4, under this mechanism, the execution of P-Type and L-Type instructions did not occur immediately but remained in a standby state until triggered by a C-Type instruction to commence their designated operations. This design permitted multiple P-Type and L-Type instructions to be grouped together and simultaneously updated by a single C-Type instruction, forming a comprehensive parameter configuration table. This design avoids the complexity of conditional judgment and execution for each instruction, improving the efficiency of the instruction pipeline and the overall execution speed.

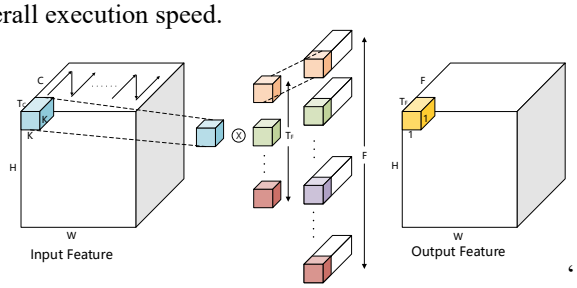

Fig.5. Data access pattern

D. Accelerate Structure

The inference-computation intensive nature of CNN requires efficient parallel computing capabilities. This design employs a multi-level parallelization strategy. This involves loop unfolding of kernel row and column iterations, input feature map channel iterations and output feature map channel iterations to achieve efficient parallel processing. As depicted in Fig.5, W, H, C, F and K represented the width, height, number of input channels, number of output channels, and size of the convolution kernel. Meanwhile, TC and TF represented the number of input and output channels used in the computation.The size of the input sliding block was K×K×TC,the weight block was K×K×TC×TF, and the output sliding block was K×K×TF.Within each clock cycle, the accelerator read the input sliding block and weight block sequentially to perform convolution operations. Following C/TC clock cycles, an output sliding block was calculated. Subsequently, this output sliding block was written back to off-chip memory or passed on to subsequent operator operations.

IV. EXPERIMENTAL RESULT

In order to verify the effectiveness of using LCA, we decided to experimentally validate three CNNs (Resnet-50, YOLOv4-Tiny and BiSeNet v1) on the XC7K410T board shown in Fig.6. Measured by actual RTL code experiments, Table 2 displayed a performance comparison between our design and existing FPGA-based accelerators. For all evaluated networks, LCA outperformed all others in terms of energy efficiency (GOP/s/W). We achieved an energy efficiency of 82.92 to 96.79 GOP/s/W. The works proposed in [6] and [10] utilized 16-bit quantization for computation,

resulting in less efficient DSP utilization and lower energy efficiency compared to ours. The implementation of the YOLOv3-Tiny network on a RISC-V processor, as proposed in [10], resulted in power consumption of 3.87W.However, due to their low operating frequencies, their throughput was approximately one-fifth of ours, resulting in energy efficiency being roughly half of what we achieved. Although the designs proposed in [11] and [12] had slightly higher throughput than ours, their power consumption was three times greater, leading to inferior energy efficiency compared to our design.

V. Conclusion

This paper explores a neural network heterogeneous computing system with lower power consumption and higher inference speeds on a LCA consisting of an RISC-V processor and a NAI neural network accelerator. For the requirement of deploying new neural network models, designs and optimizations were carried out separately on the processor and accelerator through decoupling. Experiments conducted on various neural network models demonstrated that our system, on the strength of its strong generality, exhibited superior performance.

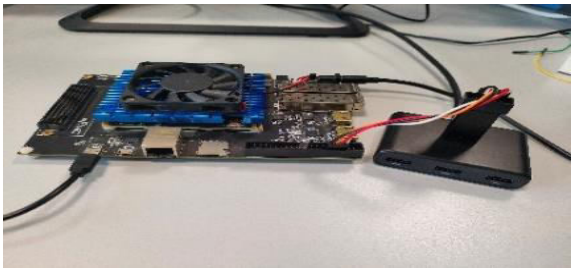

Fig.6. Evaluation board for LCA

VI. References

[1] Bharadiya, J. "Convolutional neural networks for image classification." International Journal of Innovative Science and Research Technology 8.5 (2023): 673-677.

[2] Zhao H, Morgenroth J, Pearse G, et al. A systematic review of individual tree crown detection and delineation with convolutional neural networks (CNN)[J]. Current Forestry Reports, 2023, 9(3): 149-170.

[3] Nasreen, Ghazala, et al. "A comparative study of state-of-the-art skin image segmentation techniques with CNN." Multimedia Tools and Applications 82.7 (2023): 10921-10942.

[4] Cham, Switzerland, RISC-V Specification, vol. 1, Unprivileged Spec V, 2019.

[5] A. Sanchez-Flores, L. Alvarez and B. Alorda-Ladaria, "Accelerators in Embedded Systems for Machine Learning: A RISCV View," 2023 38th Conference on Design of Circuits and Integrated Systems (DCIS), Málaga, Spain, 2023.

[6] Zhang F, Li Y, Ye Z. Apply yolov4-tiny on an fpga-based accelerator of convolutional neural network for object detection[C]//Journal of Physics: Conference Series. IOP Publishing, 2022, 2303(1): 012032.

[7] Guo Z, Liu K, Liu W, et al. An Overlay Accelerator of DeepLab CNN for Spacecraft Image Segmentation on FPGA[J]. Remote Sensing, 2024, 16(5): 894.

[8] Mani V.R.S, Saravanaselvan A, Arumugam N. Performance comparison of CNN, QNN and BNN deep neural networks for real-time object detection using ZYNQ FPGA node[J]. Microelectronics Journal, 2022, 119: 105319.

[9] Wu N, Jiang T, et al. A reconfigurable convolutional neural network-accelerated coprocessor based on RISC-V instruction set[J]. Electronics, 2020, 9(6): 1005.

[10] Pestana D, Miranda P R, Lopes J D, et al. A full featured configurable accelerator for object detection with YOLO[J]. IEEE Access, 2021, 9: 75864-75877.

[11] Liu S, Fan H, Ferianc M, et al. Toward full-stack acceleration of deep convolutional neural networks on FPGAs[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021, 33(8): 3974-3987.

[12] Nguyen D T, Je H, Nguyen T N, et al. ShortcutFusion: From tensorflow to FPGA-based accelerator with a reuse-aware memory allocation for shortcut data[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2022, 69(6): 2477-2489.