Design of an MNIST Application Accelerator Processor Based on the RISC-V Architecture

4th national RISC-V student contest 2023-2024,France

IMT Atlantique,France

Alexandre Zhang
SEH
IMT Atlantique
Plouzane,29280
alexandre.zhang@imt-atlantique.net

Yongxin Sun
SEH
IMT Atlantique
Plouzane,29280
yongxin.sun@imt-atlantique.net

Jingze Liu
ILSD
IMT Atlantique
Plouzane,29280
jingze.liu@imt-atlantique.net

***Abstract*—With the rapid advancement of Artificial Intelligence (AI) technology, traditional CPU platforms have not fully capitalized on the potential of AI, particularly in the application of neural networks. To address this challenge, we adopted the highly regarded RISC-V architecture for experimental design. Based on this architecture, we developed a new Convolutional Neural Network (CNN) processor designed to enhance the speed and flexibility of CPUs in processing AI tasks. Specifically, we optimized the structure of the CV32A6 RISC-V soft core to accelerate the critical AI application of MNIST digit recognition. Additionally, we deeply optimized the convolution operations within CNNs and implemented a series of custom instructions, including vector load/store, addition, and convolution operations, to improve the efficiency of convolution processing. Through simulation experiments, we verified the efficiency of the processor in executing both general and custom instructions. The final simulation test results not only confirmed the correctness of our design but also demonstrated its significant performance advantages.**

Keywords—RISC-V Processor, CNN, Custom Instruction, and MNIST

I. INTRODUCTION

RISC-V is an open-source Instruction Set Architecture (ISA) designed based on the Reduced Instruction Set Computing (RISC) principles. Launched in 2010 by a research team from the University of California, Berkeley, RISC-V aims to provide a scalable, efficient, and cost-effective ISA. The openness of RISC-V allows anyone to freely use, modify, and implement this architecture without paying royalties or facing complex licensing restrictions. This characteristic makes RISC-V highly suitable for academic research, commercial innovation, and the development of customized hardware.

MNIST is a widely used database for handwritten digit recognition, often serving as a benchmark for evaluating machine learning models. It contains 70,000 labeled images of handwritten digits, each 28x28 pixels in size, divided into 60,000 training images and 10,000 testing images. Due to its moderate size and well-defined problem statement, the MNIST database is an ideal experimental and teaching tool.

Utilizing the RISC-V architecture to develop hardware accelerators specifically for machine learning tasks like those presented by MNIST allows researchers and developers to leverage the flexibility and scalability of RISC-V. They can design and implement optimized hardware solutions to accelerate the inference processes of neural networks, particularly in tasks such as image recognition. For instance, customized instruction sets can be implemented on RISC-V specifically to optimize the convolution operations of Convolutional Neural Networks (CNNs), thus enhancing the efficiency and performance when processing the MNIST dataset.

RISC-V provides an open and flexible platform that allows developers to customize hardware to specifically optimize the execution of machine learning models. With MNIST serving as a standard testing platform, these hardware optimizations can be practically assessed. In this way, RISC-V can help accelerate the research and commercialization processes of AI applications.

II. BACKGROUND

A. RISC-V ISA

The RISC-V Instruction Set Architecture (ISA) is renowned for its highly modular design. The core ISA comprises only essential instructions, while complex functionalities like floating-point operations, atomic operations, and compression instructions can be achieved through optional extensions. This design endows RISC-V with tremendous flexibility, making it suitable for a wide range of applications, from simple embedded systems to complex server processors.

RISC-V supports addressing capabilities from 32-bit, 64-bit, to 128-bit to meet diverse processing needs and future technological developments. This multi-addressing capability enables RISC-V to be widely used in everything from low-power embedded devices to high-performance servers handling large volumes of data.

The basic RISC-V ISA includes four core instruction formats (R, I, S, U) and two special instruction formats (SB, UJ). All instructions are fixed at 32 bits and must be aligned on four-byte boundaries in memory. The instruction formats are as follows:

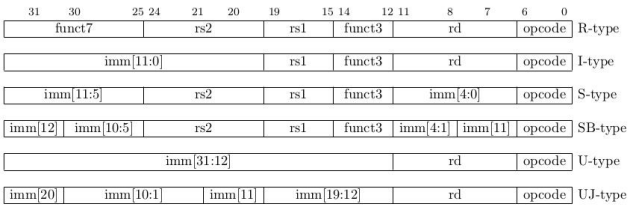

Fig 1. RISC-V Basic Instruction Format

R-type (Register Type): Used for arithmetic and logical operations between registers.

I-type (Immediate Type): Supports operations combining immediate values with register values.

S-type (Store Type): Used for storing data from registers to memory addresses.

B-type (Branch Type): Implements conditional program jumps.

U-type (Upper Immediate Type): Used for loading large immediate values into registers.

UJ-type(Jump Type): Used for unconditional program jumps, commonly used in function calls.

In all instruction formats, the positions of the source registers (rs1 and rs2) and the destination register (rd) are unified to simplify the instruction decoding process. The register fields are positioned at the instruction's far left, a design that helps reduce hardware complexity, ensuring system design simplicity and operational efficiency.

B. CV32A6

Developed by the OpenHW Group, the CV32A6 is an open-source processor core based on the RISC-V instruction set that follows the modular design principles of RISC-V, providing a flexible and feature-rich computing platform.

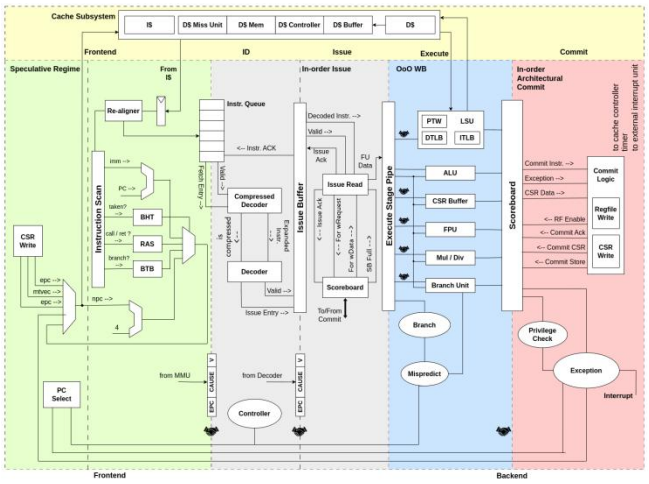

Fig 2. CV32A6 Structure

The core supports the RV32I basic integer instruction set and can be enhanced with additional standard and non-standard extensions such as integer multiplication and division (M extension), atomic operations (A extension), and single and double-precision IEEE 754 floating-point operations (F and D extensions). The CV32A6 has 32-bit addressing capability and streamlined instruction sets optimized for performance and energy efficiency, making it particularly suitable for power-sensitive embedded system applications. Furthermore, the core utilizes modern pipeline technology to enhance instruction processing speed and efficiency, meeting high-performance computing demands. The CV32A6 also includes support for specific hardware acceleration features, such as custom instructions for applications like digital signal processing or machine learning algorithms. In practical applications, CV32A6 is extensively deployed from simple industrial sensors to complex communication devices. Its low power consumption and high performance make it an ideal choice for Internet of Things (IoT) and edge computing devices. The open-source nature allows manufacturers and development teams to freely customize and expand processor functionalities to meet specific product needs, thereby reducing development costs and shortening time to market. Thus, CV32A6, as a highly customizable and technologically advanced processor core, provides immense value in embedded systems and high-performance computing scenarios, especially in applications seeking efficiency and energy optimization.

C. MNIST

The MNIST dataset, standing for "Modified National Institute of Standards and Technology," is a widely used database for handwritten digit recognition. It contains 70,000 28x28 pixel grayscale images, with 60,000 for training and 10,000 for testing, each labeled with the corresponding handwritten digit (0 to 9). Due to its simplicity and clear problem definition, MNIST has become one of the most elementary and popular benchmarks in the fields of machine learning and image processing. It is frequently used to validate the effectiveness of new machine learning algorithms and network architectures, providing an intuitive way to demonstrate how algorithms can learn to process image data for digit recognition. Although relatively simple compared to modern datasets, MNIST still provides an ideal starting point for beginners and offers researchers the opportunity to test and improve algorithms, making it a valuable tool for assessing and showcasing image recognition technology.

III. Related Work

Before you begin to format your paper, first write and

A. Optimization of RISC-V R-Type Instructions

We have utilized R-type instructions, which are register-type operations used for arithmetic and logical computations between registers. The format of R-type instructions includes `opcode` (operation code), `rd` (destination register), `funct3` (3-bit function selector), `rs1` (source register 1), `rs2` (source register 2), and `funct7` (7-bit function selector). This organization of instruction format enables the execution of various fundamental operations, including addition, subtraction, bit shifting, and logical tasks, as shown in Figure 3.

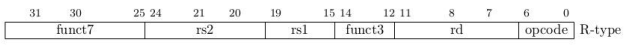

Fig 3. RISC-V R-Type Instruction

R-type instructions streamline computational processes by directly manipulating data between registers, significantly enhancing data processing speed. This characteristic is particularly beneficial for applications requiring high-speed

data handling, such as complex mathematical computations and large-scale data operations. The use of R-type instructions also helps reduce the latency associated with memory access, thereby optimizing the overall execution speed of programs.

Furthermore, we have leveraged the CUSTOM 0 in the RISC-V extension instruction set, a reserved operation code for developers to create custom instructions. We set both `funct3` and `funct7` to 0, with an opcode of 0001011. By this means, we introduced a specific custom instruction designed to accelerate certain computational tasks. This capability for customization allows us to optimize at the hardware level for specific application requirements, thereby further enhancing processing performance and system efficiency, as illustrated in Figure 4. This demonstrates the powerful extensibility of the RISC-V architecture and its tremendous flexibility in optimizing for specific applications.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Fig 4. Custom Instruction

B. Parallel Convolutional Arithmetic Methods

We utilize a specialized parallel processing method to accelerate convolution operations. This approach involves treating the data within a single 32-bit register as a vector comprising four separate 8-bit integers, and performing element-wise multiplication with another similarly structured register, where each 8-bit integer from one register is multiplied in parallel with the corresponding 8-bit integer from another 32-bit register. The results are then summed to complete the convolution operation. This is a typical Single Instruction, Multiple Data (SIMD) operation, commonly used in image processing, signal processing, machine learning, and other applications requiring extensive numerical computations.

This process allows multiple data points to be processed within a single CPU cycle, significantly enhancing processing efficiency. Below is a more detailed explanation of the process:

- Data Representation:The raw data is stored in two 32-bit registers, such as `a` and `b`. Each register can be viewed as containing four 8-bit integers, meaning each register simultaneously represents four independent data points. For example, register `a` contains `a(0), a(1), a(2), a(3)`, where each `a(i)` is an 8-bit integer.
- Parallel Multiplication Operations:The corresponding 8-bit numbers are each subjected to multiplication operations. For instance, the 8-bit numbers from registers `a` and `b` are paired for multiplication: `a(0)` multiplied by `b(0)`, `a(1)` by `b(1)`, and so on up to `a(3)` by `b(3)`. These multiplication operations occur simultaneously, as they are independent operations.
- Intermediate Result Storage:Each 8-bit by 8-bit multiplication produces a 16-bit result due to the potential increase in bit width resulting from the multiplication. These 16-bit results are stored in an intermediate storage location or register. For example, the multiplication results could be denoted as `mult_result(0), mult_result(1), mult_result(2), mult_result(3)`.
- Accumulation Operation:The multiplication results are then summed to form the final convolution result. This accumulation operation involves adding the 16-bit numbers, potentially requiring a larger register to store the final sum. For instance, summing the four 16-bit results might yield a 32-bit convolution result.
- Result Storage:The final 32-bit convolution result is stored in a destination register for subsequent processing. This result can be directly used for further processing or serve as a function’ s return value.

The detailed process is depicted in Figure 5.

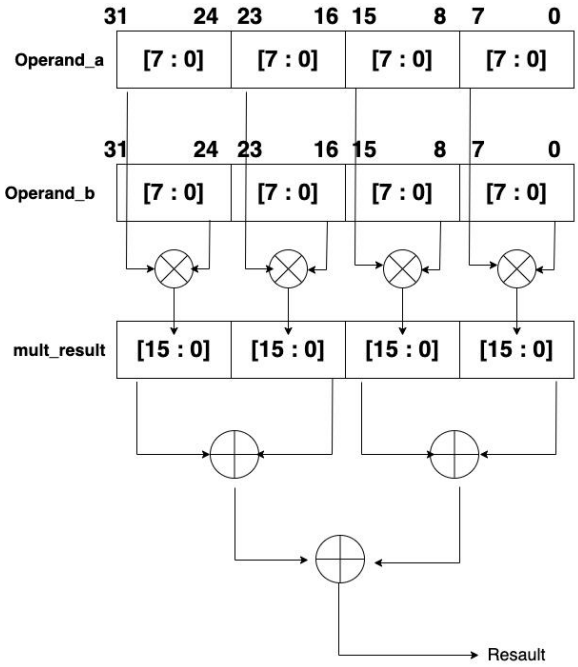

Fig 5. Schematic

IV. TEST AND RESULT

A. Integration of Custom Vector Instructions to Enhance Computational Efficiency

To fully utilize the custom `conv` vector instruction for specific machine learning tasks, such as MNIST handwritten digit recognition, this study modifies the official MNIST code provided by competition organizers to employ the custom instruction set `conv`, thereby achieving hardware acceleration.

B. ICode Implementation and Optimization

In the implementation, the data packing process is first defined to ensure that the data format matches the

requirements of the custom instruction. The following code snippet demonstrates how to pack individual data inputs into a 32-bit format so that the `conv` instruction can process four 8-bit data points in a single operation:

```
uint32_t packed_inputs = 0;
packed_inputs |= (uint32_t)inputs[iter + 0] << 0;
packed_inputs |= (uint32_t)inputs[iter + 1] << 8;
packed_inputs |= (uint32_t)inputs[iter + 2] << 16;
packed_inputs |= (uint32_t)inputs[iter + 3] << 24;
```

```
uint32_t packed_weights = 0;
packed_weights |= (uint32_t)(uint8_t)weights[iter + 0] << 0;
packed_weights |= (uint32_t)(uint8_t)weights[iter + 1] << 8;
packed_weights |= (uint32_t)(uint8_t)weights[iter + 2] << 16;
packed_weights |= (uint32_t)(uint8_t)weights[iter + 3] << 24;
```

Subsequently, the `asm volatile` directive is used to embed the custom `conv` instruction, performing the convolution operation on the packed input and weight data, and accumulating the results:

```
uint32_t result;
asm volatile (
   "conv %[result], %[inputs], %[weights]\n\t"
   : [result] "=r" (result)
   : [inputs] "r" (packed_inputs), [weights] "r" (packed_weights)
);
*weightedSum += (SUM_T)result;
```

C. Test result

This section of the paper presents a detailed comparative analysis of the performance and resource utilization metrics for our optimized RISC-V based processor designed specifically for the MNIST handwritten digit recognition application. The following table summarizes key parameters measured before and after applying several architectural and processing optimizations to the processor design. The data highlights the impact of these changes on system performance, processing efficiency, and hardware resource allocation.

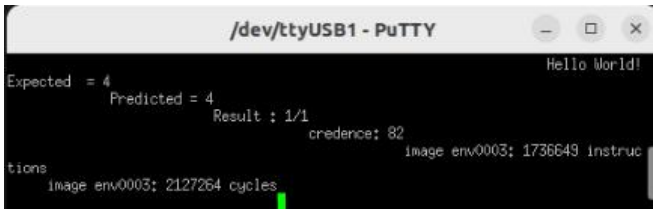

Fig 6. Testing Result

Table Summary of Performance and Resource Utilization:

Aspect	Before Optimization	After Optimization	Change
Recognized Digit	4	4	No change
Credence	82%	82%	No change
Number of Cycles on FPGA Board	2,316,653 cycles	2,127,264 cycles	Decrease of 8.175%
Number of Cycles in Simulation	2,316,653 cycles	2,090,182 cycles	Decrease of 9.78%
Maximal Frequency	51.54 MHz	48.54 MHz	Decrease of 5.82% (within acceptable limit of 20%)
Number of LUTs	8,582	9,221	Increase of 639 LUTs
Number of Flip-Flops (FFs)	4,611	4,627	Increase of 16 FFs
Number of RAM36 Blocks	16	16	No change
Efficiency & Speed	Standard efficiency and processing speed	Enhanced efficiency and processing speed	Improved processing efficiency and speed

Fig 7. Result Table

The data in the table illustrates several critical outcomes from our optimization efforts. Notably, the optimizations maintained the integrity of digit recognition, as evidenced by the unchanged recognized digit and credence values. These optimizations significantly improved the processor's operational efficiency, as demonstrated by the reduction in processing cycles by 8.175% on the FPGA board, and an even more substantial reduction of 9.78% in simulation cycles. This further enhancement in simulation cycles, from 2,316,653 cycles before optimization to 2,090,182 cycles after, directly contributes to faster data processing and reduced model training times, vital for real-time applications.

Moreover, there was a slight reduction in the maximal frequency of the processor, which decreased by 5.82%. Despite this decrease, the change remains within the acceptable performance degradation limit of 20%, indicating that the enhancements did not adversely affect the processor's operational speed beyond a negligible threshold.

The increase in the number of Look-Up Tables (LUTs) and Flip-Flops (FFs) suggests a higher complexity in the FPGA configuration but is justified by the corresponding gains in processing capabilities and efficiency. The constant number of RAM36 blocks demonstrates efficient memory management despite the broader scope of processing tasks.

These results underscore the effectiveness of our design modifications in achieving a balanced enhancement across various performance metrics, including a notable reduction in simulation cycles which highlights the processor's improved efficiency under simulated operational conditions. Future work will focus on exploring further optimizations to enhance these metrics without significantly impacting the system's frequency and resource efficiency. This ongoing development is expected to refine our approach and extend the processor's applicability to more complex AI-driven tasks, ensuring it can meet the increasing demands of advanced applications.

V. CONCLUSION

The integration of parallel processing technology into the MNIST handwritten digit recognition task utilizing convolutional neural networks (CNNs) has demonstrated substantial improvements in image feature extraction and model training processes. This technology enables

simultaneous processing of multiple convolution kernels and image regions, significantly increasing data throughput and processing speed. Such advancements contribute to reduced model training times and bolstered real-time processing capabilities, essential for dynamic recognition systems that operate in variable environments.

Furthermore, the implementation of parallel processing techniques notably decreases CPU load and conserves energy, making this approach particularly beneficial for applications deployed on resource-constrained devices. The scalability and flexibility of this technology empower system designers to customize processing unit configurations to meet specific application demands, optimizing resource utilization and enhancing processing accuracy. This adaptability is critical for maintaining a competitive edge in the rapidly evolving field of machine learning.

From a theoretical perspective, if a system is capable of processing four data points per cycle, it could potentially quadruple performance. However, the actual performance gains are contingent upon the efficiency of algorithm parallelization, hardware execution, and the mitigation of system bottlenecks such as memory bandwidth and I/O latency. Advanced processors equipped with SIMD instruction sets or GPU acceleration are poised to achieve higher degrees of parallelism, which could amplify performance by significant multiples, depending on the degree of task parallelization and the sophistication of hardware optimization.

Achieving and quantifying these performance enhancements necessitates rigorous performance evaluations and empirical testing. Such assessments are vital for substantiating the practical benefits of parallel processing in image and signal processing tasks. This conclusive evidence will not only validate the effectiveness of current implementations but also guide future enhancements to further refine the technology's application in diverse machine learning scenarios. The ongoing development and optimization of parallel processing techniques will continue to be instrumental in advancing the capabilities of AI technologies, ensuring they meet both current and future computational demands.

REFERENCES

[1] Y. Lee, A. Ou, and A. Magyar, “Z-scale: Tiny 32-bit risc-v systems,” in Open-RISC Conf., Geneva, Switzerland, 2015.

[2] Z. Lia, W. Hua, and S. Chena, "Design and Implementation of CNN Custom Processor Based on RISC-V Architecture," in **College of Computer Science and Technology, Wuhan University of Science and Technology; Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, China**.

[3] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, "A ‘New Ara’ for Vector Computing: An Open Source Highly Efficient RISC-V V 1.0 Vector Processor Design," in **Integrated Systems Laboratory, ETH Zurich, Zurich, Switzerland; Computing Systems Laboratory, Huawei Zurich Research Center, Zurich, Switzerland; University of Bologna, Zurich, Switzerland.**