# Accelerating DNNs using Weight Clustering on RISC-V Custom Functional Units

Muhammad Sabih, Batuhan Sesli, Frank Hannig, and Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
{muhammad.sabih, batuhan.sesli, frank.hannig, juergen.teich}@fau.de

***Abstract*—Weight clustering is typically used to compress a Deep Neural Network (DNN) by reducing the number of unique weight values, which can be encoded using a few bits. However, using weight clustering for acceleration remains an unexplored area. In this work, we propose a design using Custom Functional Units (CFUs) to accelerate DNNs with weight clustering on a RISC-V-based SoC. We evaluate our accelerator on resource-constrained ML use cases and are able to report considerable speedups of up to 8 times with minimal overhead in the utilization of FPGA resources.**

## I. Introduction

As deep neural networks continue to grow in complexity, there is a greater need for neural network solutions that are small, low-latency, and energy efficient. Pruning, quantization, Neural Architecture Search (NAS), custom accelerator design, and efficient compilation are some of the ways to address the complexity problem of DNNs. RISC-V has become an attractive choice as a target for DNNs due to its customizability.

In this work, we propose Custom Functional Units (CFUs) to accelerate the execution of DNNs using *weight clustering* [1] on RISC-V processors. Weight clustering limits the number of unique values in a DNN layer, e.g., allowing them to be stored with fewer bits using a codebook. At the same time, the decoded weights remain in higher precision (e.g., FP32). Originally and still common, weight clustering has been utilized for compression [1]. In contrast, our presented approach not only utilizes weight clustering for compression but also enables acceleration. We employ *CFU Playground* [2] to implement our proposed concept as a RISC-V instruction set extension. CFU Playground is a framework that enhances the agility of hardware/software co-design and domain-specific optimizations.

## II. Background

*1) Weight Clustering:* The concept of weight clustering was initially aimed at reducing the storage requirements of DNNs with minimal accuracy loss. Its main idea is to restrict the number of unique weights in each layer to a smaller set of so-called *clusters*, as shown in Figure 1.

*2) Custom Function Units:* The RISC-V instruction set architecture allows for tailored instruction set extensions and accelerator design. Customization of the RISC-V processor architecture is achieved with so-called CFUs. These CFUs refer to custom logic added in hardware that is tightly coupled with the processor to provide specialized functionality. A CFU is addressed by the RISC-V ISA using the *R-type* instruction. These instructions utilize two 32-bit input registers (*rs1* and *rs2*) and store the output in one 32-bit register (*rd*). The custom instruction's type is specified by a 7-bit *opcode* (*funct7* in Fig. 2). Notably, CFUs do not have direct access to the main memory.

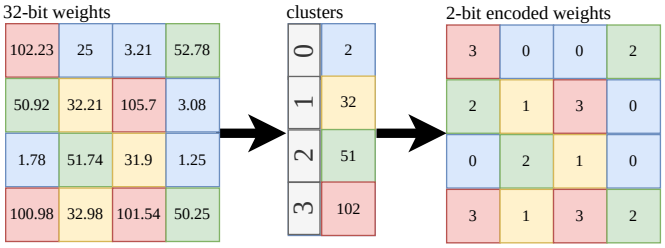


Fig. 1: An illustration of weight clustering: matrix with $n = 16$ different weights before clustering (left), clustering into $k = 4$ clusters with cluster center $c_i$, $i = 1,...,4$ (middle) and resulting 2-bit encoded weights (right).

*3) CFU Playground:* CFU Playground uses an open-source implementation of a RISC-V processor known as VexRiscv [3]. A LiteX SoC configuration is placed within VexRiscv. The CPU-CFU interface provides a very tight coupling between the CPU and the added custom functionality. During FPGA synthesis, place, and route, the interface disappears, and the CFU essentially becomes a component of the CPU pipeline. The new instructions can be utilized in C or C++ through an inline assembly macro provided. No adjustments to the RISC-V GCC toolchain are necessary.

## III. Proposed Approach

In this section, we elaborate on our approach to utilize weight clustering for DNN acceleration using RISC-V CFUs.

*1) Weight Clustering Accelerator Custom Functional Unit (WCACFU) Design:* The main idea for obtaining promising acceleration numbers through weight clustering is to utilize the internal memory of the FPGA to store the resulting codebooks. Moreover, we consider codebook sizes of 2, 4, and 16 clusters in the following, corresponding to 1, 2, and 4 bits, respectively.

The codebook needs to be initialized once for each layer. After that, the packed weights are pushed into the CFU memory. The CFU memory can be realized either using FPGA flip-flops or Block RAMs (BRAMs). The synthesis tool chooses flip-flops if the required memory is small enough; otherwise, it instantiates BRAMs. The CFU internally decodes the weights and multiplies them with input values (these are activations or the input to the model). In order to implement our approach, three types of instructions need to be defined: (i) *setting the codebook*, (ii) *pushing weights*, and (iii) *performing ALU operations*. We use the most significant four bits of *funct7* to decode a total of 9 instructions grouped into three types. The mapping is shown in Figure 2.

979-8-3503-4859-0/DATE24/© 2024 EDAA

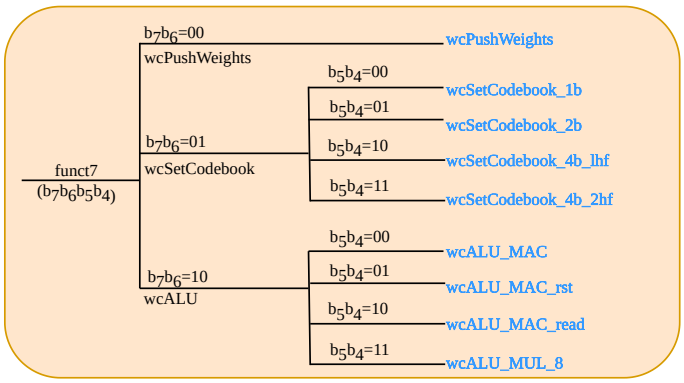


Fig. 2: Instructions defined and supported by WCACFU and encoding using four bits of the field *funct7*.

*Kernel Optimization:* We show a pseudocode of the kernel of a convolutional layer and demonstrate the acceleration using the weight clustering approach of the innermost loop. An unoptimized kernel can perform 32 multiply-and-accumulate operations with 32 calls of the shown *accumulate* instruction. Whereas, using WCACFU, it can be done in five C instructions as shown in Figure 3. Thus, the expected speedup when using four clusters (2-bit codebook) for the convolutional layer is $6-7\times$. Kernels of other layers such as a fully connection layer, a depthwise convolutional layer, etc., can also be easily adapted.

```
for (batches) {
 for (output_height) {
  for (output_width) {
   for (out_channel) {
    for (int idx=0; idx<32; idx++) {
     accumulate(input_arr[idx], filter_arr[idx]);}}}}}
```

(a) Pseudocode of an unoptimized convolutional kernel.

```
wcSetCodebook_2b(codebook); // configures codebook
for (batches) {
 for (output_height) {
  for (output_width) {
   for (out_channel) {
    // push 32 weights, each 2 bits
    wcPushWeights(filt_weights[0], filt_weights[1]);
    wcALU_MAC(inp_arr[0], inp_arr[1]);      // 8 MACs
    wcALU_MAC(inp_arr[2], inp_arr[3]);      // 8 MACs
    wcALU_MAC(inp_arr[4], inp_arr[5]);      // 8 MACs
    wcALU_MAC(inp_arr[6], inp_arr[7]); }}}} // 8 MACs
```

(b) Pseudocode of a convolutional kernel optimized for WCACFU.

Fig. 3: A comparison of an unoptimized convolutional kernel and a WCACFU optimized kernel.

## IV. Evaluation

For evaluation, we consider a $1\times1$ convolutional kernel of varying input and output depths and a fully connected layer from three different applications: KWS (Key Word Spotting), ANOD (Anomaly Detection), and PDTI8 (Person Detection INT8). These applications are included in the CFU Playground. The acceleration using 2-bit and 4-bit clustering is shown in Figure 4. The FPGA resource usage is shown in Table I. We can observe a significant acceleration for a minimal overhead in terms of FPGA resources. Additionally, our design can be extended for other types of layers and clustering of activations as well.

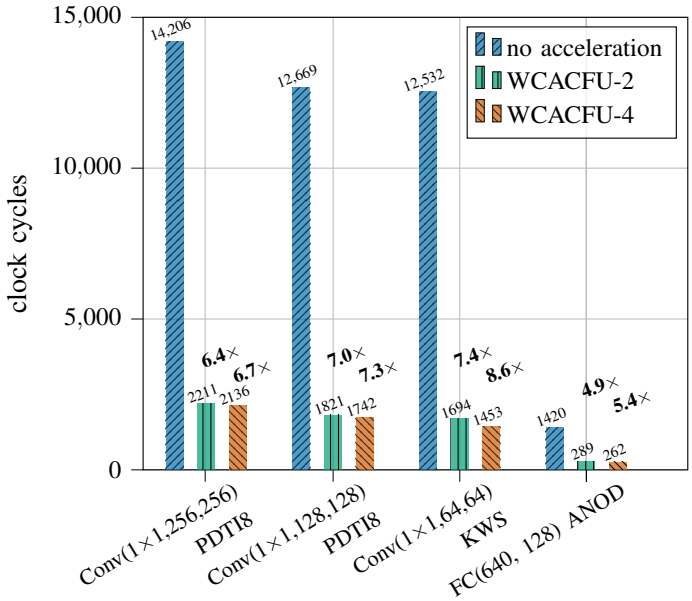


Fig. 4: Acceleration with WCACFU. WCACFU-4 represents the 4-bit (16 clusters) case and WCACFU-2 represents the 2-bit (4 clusters) case.

TABLE I: FPGA utilization with and without our WCACFU.

| | **RISC-V** | | **Cost** | **FPGA Utilization** [%] | | |
|---|---|---|---|---|---|---|
| | w/o CFU | WCACFU | [%] | w/o CFU | WCACFU | Overhead |
| LUTs | 2,650 | 3,058 | 15.39 | 12.74 | 14.70 | 1.96 |
| Slice FF | 2,077 | 2,141 | 3.08 | 4.99 | 5.15 | 0.16 |
| BRAMs | 15 | 15 | 0 | 30 | 30 | 0 |
| DSPs | 4 | 4 | 0 | 4.44 | 4.44 | 0 |

## V. Conclusion and Future Work

Weight clustering has previously been used only for compression during DNN optimization and mapping. In this paper, we proposed implementing weight clustering as a custom instruction of a RISC-V CPU to achieve significant acceleration in DNN inference with minimal overhead in terms of FPGA resources. For future work, it would also be interesting to quantize activations and incorporate techniques from *explainable AI* [4] to guide mixed-precision weight clustering.

*Acknowledgement:* This work was partly supported by the Fraunhofer Institute for Integrated Circuits IIS, Erlangen, Germany, and partly by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 524986327 (NA$^3$Os).

## References

[1] S. Han, H. Mao, and W. J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. 2016. arXiv: 1510.00149 [cs.CV].

[2] S. Prakash et al. "CFU Playground: Want a faster ML processor? Do it yourself!" In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE, 2023, pp. 1–2. DOI: 10.23919/DATE56975.2023.10137093.

[3] SpinalHDL. *VexRiscv Core*. URL: https://github.com/SpinalHDL/VexRiscv (visited on 12/14/2023).

[4] M. Sabih, F. Hannig, and J. Teich. "Utilizing Explainable AI for Quantization and Pruning of Deep Neural Networks". In: *The Computing Research Repository (CoRR)* (Aug. 20, 2020). arXiv: 2008.09072 [cs.CV].