EasyChair Preprint

№ 15741

Evaluating Multiplier-Less CNNs in RISC-V Architecture

Bruno Henrique Spies, Mathias Cirolini Michelotti,
Leonardo Londero de Oliveira and Everton Alceu Carara

January 20, 2025

Evaluating Multiplier-Less CNNs in RISC-V Architecture

Bruno Henrique Spies, Mathias Cirolini Michelotti, Leonardo Londero de Oliveira, Everton Alceu Carara
Federal University of Santa Maria (UFSM)
Santa Maria, RS, Brazil
bruno.spies@ecomp.ufsm.br, mathias.michelotti@ecomp.ufsm.br, leonardo@ecomp.ufsm.br, carara@ufsm.br

***Abstract*—In recent years Convolutional Neural Network (CNN) emerged as Machine Learning (ML) became a popular approach to solve problems in distributed area computations such as mobile devices and Internet of Things (IoT). It is well known that local computation at edge devices is preferable over transmitting a huge amount of data to run ML algorithms at a central node. In this sense, RISC-V has the research community's attention as a flexible architecture and royalty-free alternative for embedded processors and IoT devices. Although the latest research on RISC-V and CNNs has been instruction set architecture (ISA) customization to speed up the convolution process, this work investigates the impact on inference execution time when replacing multiplication instructions by shift in multiply and accumulate (MAC) operations. Compared to slow multi-cycle multiplication instructions, our experiments showed inference throughput speedup ranging from 1.45x to 1.95x with negligible impact on memory footprint and employing only the base integer RISC-V ISA (RV32I).**

***Index Terms*—RISC-V, Convolution Neural Networks, shift, MAC operation, power-of-two Quantization.**

I. Introduction

Devices to support Augmented Reality (AR) and Virtual Reality (VR) are examples of what tends to be the next computing platform. They require computation as data is acquired and thus establish the need for a concrete edge AI concept. Convolutional Neural Networks (CNNs) have emerged as Machine Learning (ML) basis in various fields where AI is needed, particularly in image and signal processing tasks. As computation moves closer to the source of data, there is an increasing emphasis on optimizing CNNs inference for deployment on resource-constrained edge devices, which demands efficiency and low power.

In mobile edge inference, such as Android devices, the CPU handles all ML computations without any additional accelerators [1]. RISC-V is an open-source instruction set architecture (ISA) that allows customization and extension, where researchers have explored creating domain-specific instruction sets tailored for CNNs. The energy efficiency of RISC-V-based CNN inference depends on factors such as quantization level, hardware architecture, and optimization techniques. Custom instruction sets, like those designed for CNN computation, can greatly reduce execution latency and energy consumption but may require time and engineering efforts to validate the ISA and compilation for a new silicon-proven architecture.

CNNs typically have a huge amount of parameters (weights and biases) represented as 32-bit floating point values that demand a lot of memory. Furthermore, the convolution process is computationally intensive as it involves the CNN parameters and plenty of repetitive multiply and accumulate operations (MAC). Through a quantization process, it is possible to reduce the memory requirement and avoid the floating point unit (FPU), however, multiplication is still needed. Multiplication is a very power-consuming task in computational systems, and replacing it with more computationally lightweight alternatives such as shift operations may significantly reduce complexity, power consumption, and in some cases execution time. In this sense, multiplier-less CNNs development is a promising approach whether in hardware or software implementation.

In this paper, we investigate shift advantages over multiplication when RISC-V processors execute convolution neural networks, without any kind of hardware accelerator. The primary contributions of this paper are as follows:

1) We present the execution time impact and a lower-level analysis of the shift advantages over multiplication in the context of CNNs;
2) We highlight the leverage inference throughput without any hardware requirement beyond the base integer RISC-V ISA (RV32I).

II. Case Study CNN Models

In this work we trained three different CNN models using four popular datasets: (i) MNIST, (ii) GTSRB, (iii) CIFAR10 and (iv) SVHN. Since there is no well-established methodology to design CNN architectures, we tried some kind of manual NAS (Network Architecture Search) [2] to find small models with good accuracy. Those models are described as follows:

- **MNIST_CNN**: it contains a succession of (3x3 convolution, 2x2 maxpool, ReLU) layers repeated two times, followed by one fully connected classification layer containing 10 neurons (Figure 1). Each convolution layer applies 4 filters;
- **GTSRB_CNN**: same architecture as MNIST_CNN, with convolution layers applying 16 filters and classification layer containing 43 neurons;
- **CIFAR10_SVHN_CNN**: it contains a succession of (3x3 convolution, ReLU, 3x3 convolution, ReLU, 2x2 maxpool) layers repeated three times, followed by one fully

Figure 1. CNN model for MNIST and GTSRB datasets.

connected classification layer containing 10 neurons. Each convolution layer applies 26 filters and padding = 1.

Figure 2. CNN model for CIFAR10 and SVHN datasets.

Important to note that our models have only one fully connected layer, which differs from typical models where some hidden layers present dozens or hundreds of neurons. Also, models to classify more complex datasets such as CIFAR10, employ hundreds of filters in convolutional layers. However, edge computing with low memory and computing budgets, can't afford such requirements.

Our training process was supported by the PyTorch library [3] and has as main goal to find weights which are power-of-two values through quantization-aware training (QAT). Since we want to be able to execute CNNs using integer datapaths, once QAT is finished, all network parameters (weights and biases) are converted to integer values multiplying them all by a scale factor which is also a power-of-two value. The resulting weights are integer power-of-two values that can be multiplied using shift operations instead of multiplication. The integer CNN accuracy depends directly on the applied scale factor value, which also impacts the bit-width required to store weights.

CNN models operating on color datasets (GTSRB, CIFAR10 and SVHN) were designed to process (train/infer) only the images green channel to reduce weights and computations in the first convolution layer. Table I presents the accuracy results of floating point and power-of-two quantized CNNs. The integer CNNs require 5-bit weights and have an almost negligible accuracy reduction with regard to their float counterparts.

Table I
ACCURACY COMPARISON.

Dataset	Float (32-bits)	Integer (5-bits)
MNIST	98.54%	98.01%
GTSRB	95.1%	94.49%
SVHN	95.18%	94.25%
CIFAR10	83.98%	81.55%

III. RESULTS

To evaluate multiplier-less CNNs in RISC-V architecture we developed an integer C code composed basically of functions that implement the colored blocks in Figures 1 and 2. We have a Python script that automatically generates C language parameters (weights and biases) from trained integer PyTorch models and also specifies the sequence in which these functions are applied for each model.

Even though weights bit-width is 5-bits, they are stored in 8-bits using char type. This way it is possible to avoid weight unpacking instructions, due to the lack of support, at the ISA level, for sub-byte data types. Regardless dataset, the program memory footprint (no data) is about 4.3 KiB, with a negligible difference depending on the MAC operation (*shift* or *multiplication*). Data memory footprint depends on the model architecture (Figures 1 and 2) and are listed in table II. We compiled using GCC and -O3 optimization flag in order to get the minimum number of executed instructions.

Table II
DATA MEMORY FOOTPRINT.

Integer CNN	Weights/Biases	Feature maps/Vars
MNIST_CNN	1.18 KiB	7.11 KiB
GTSRB_CNN	26.72 KiB	30.51 KiB
CIFAR10_SVHN_CNN	34.32 KiB	61.32 KiB

Following C code listings (1 and 2) present the MAC operation implemented in convolution and fully connected layers using *shift* and *multiplication*, respectively. Each one is inside a loop that iterates according to the current set of weights and input features. In convolution layers *kernel* points to the current convolution kernel whereas in fully connected layers it points to the current neuron weights. Input layer features are pointed by *receptive_field*. Such snippets highlight the unique difference between the *shift* and *multiplication* based CNNs: the MAC implementation.

Listing 1. Multiplication based MAC

```
weight = *kernel;            // Current weight
feature = *receptive_field; // Current input feature

/* Multiplication based MAC */
acc += (feature * weight) >> SHIFT_SCALE;

kernel++;
receptive_field++;
```

Listing 2. Shift based MAC

```
weight = *kernel;            // Current weight
feature = *receptive_field; // Current input feature

/* Shift based MAC */
if (weight > 0)
  acc += (feature << weight) >> SHIFT_SCALE;

else if (weight < 0)
  acc -= (feature << -weight) >> SHIFT_SCALE;

kernel++;
receptive_field++;
```

Multiplication-based MAC is trivial, weights are power-of-two values directly multiplied by the feature. Since weights

and features are already scaled by SHIFT_SCALE, the resulting multiplication must be re-scaled. Scale factor is a power-of-two value applied with the left shift, likewise re-scaling is applied with right shift. Shift-based MAC requires weights coded as signed exponents, where sign indicates the resulting shift sign and magnitude indicates bits to shift. Such weights coding implies the if/else structure in listing 2 to determine whether to add or subtract during accumulation and change the weight sign when negative to allow shift execution.

A. Instruction count

The Imperas RISC-V instruction set simulator [4] was used to generate instruction level accurate execution traces. This first evaluation is in terms of ISA, where the performance is based on the number of executed instructions. In addition to RISC-V base integer ISA (RV32I), present in any implementation, we also consider M extension (RV32IM), including instructions for integer multiplication and division. We simulate the three integer CNNs (Table II) with MAC operation implemented in three versions:

- **mult_sw**: RV32I ISA does not contain multiplication instructions, so multiplication subroutines created by GCC are used instead;
- **shift**: RV32I ISA contains shift instructions, which are used rather than multiplication instructions (Listing 2);
- **mult_hw**: this version requires RV32IM ISA, which support hardware multiplication (Listing 1).

Figures 3 and 4 present the number of executed instructions for the three MAC operation versions in different CNNs. CNNs for MNIST and GTSRB datasets (Figure 3) have few layers/filters with instruction count varying in the order of tens of millions, whereas CNN for CIFAR10 and SVHN datasets (Figure 4) are complex with instruction count varying in the order of hundreds of millions. As expected, software-implemented multiplication (mult_sw) has by far the lowest performance. Shift-based MAC has about 3% to 7% more instructions executed than hardware multiplication (mult_hw) due to the *if/else* structure seen in Listing 2 and not present in Listing 1. Bearing in mind that MAC-based shift does not require any ISA extension, the presented drop in performance can be considered acceptable.

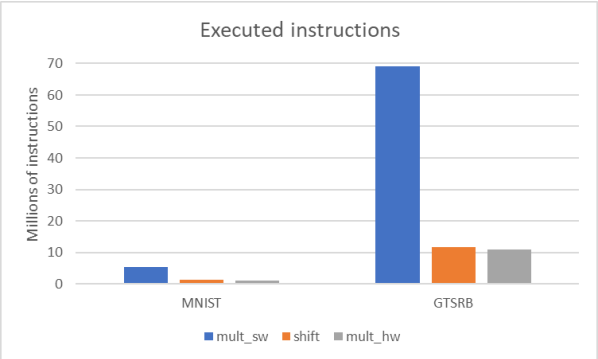

Figure 3. Instruction count for MNIST and GTSRB.

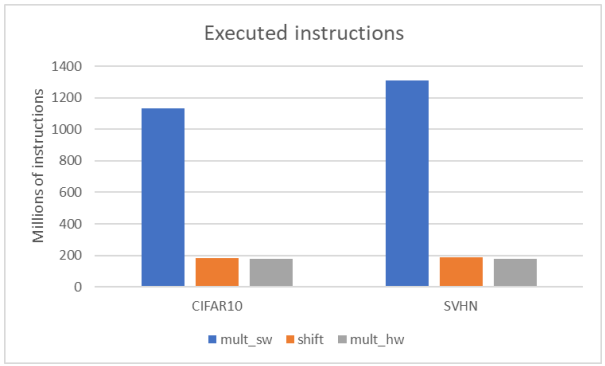

Figure 4. Instruction count for CIFAR10 and SVHN.

B. Execution time

This subsection explores at a lower abstraction level, the performance comparison between shift and multiplication-based MAC through different RISC-V implementations. Multiplication-based MAC showed higher performance in terms of executed instructions. However, multiplier circuits present latency ranging from 1 to n clock cycles, where n is the factor bit width.

Combining execution traces from previous evaluations with features of specific RISC-V implementations, it is possible to infer execution time at the clock cycle level or even in seconds. We start with RVCoreP-32IM [5], which is an extension of the 5-stages pipeline RVCoreP [6] soft processor (implementing RV32I ISA only) to support RISC-V M extension. It is described in Verilog HDL and focused on FPGA prototyping. M extension is supported using a multiplier unit able to choose between iterative radix-4 booth multiplier and DSP-based multiplier. Multiplication operation latency for the execution stage is 18 and 2 clock cycles for radix-4 and DSP multiplier, respectively. Besides multiplication stalls, the pipeline also stalls in case of load-use dependency (1-cycle penalty) due to load instructions and missed branch prediction (3-cycle penalty). Load-use dependency is easily detected analyzing the executed instructions trace. Regarding branch prediction, it is not possible to extract from the execution traces hit or missed predictions, also its accuracy is not mentioned in [5] [6]. Even modern branch predictors predict the vast majority of conditional branch instructions with near-perfect accuracy [7], we set hit prediction accuracy to 90%, meaning 10% of total branch instructions will be penalized in 3 cycles. Increased hit prediction accuracy favours shift-based MAC due to the *if/else* structure since it is assembly implemented with branch instructions, which are a bottleneck to pipelining.

Figure 5 presents a performance comparison between shift and multiplication-based MAC. Bars represent shift performance normalized with regard to MAC using radix-4 multiplier (blue bars) and MAC using DSP multiplier (grey bars). The horizontal red line is a multiplication-based MAC reference (radix-4 and DSP). Since the radix-4 multiplier has 18-cycles latency, shift-based MAC clearly surpasses it, which was hidden by previous ISA-based comparisons. Concerning

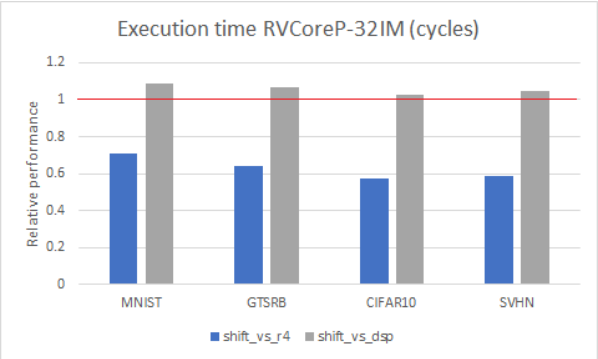

Figure 5. Shift performance w.r.t. radix-4 and DSP multipliers

DSP-based MAC, shift increases the execution time by about 2% to 8%, similar to previous ISA-based comparisons. However, note that hardware multiplication requires RISC-V M extension (RV32IM).

The second evaluation takes two SiFive cores E20 (2-stages pipeline) [8] and E51 (5-stages pipeline) [9], both coupled with 5-cycles latency multiplier. Besides the M extension, both cores also implement A (atomic instructions) and C (compressed instructions) extensions (RV32IMAC). In reason of pipeline depth, such cores differ in load latency. E51 has 2-cycle latency executing LW instructions and 3-cycles latency executing LH, LHU, LB and LBU instructions. E20 has 2-cycles latency for all load instructions. Both cores stall in case of load-use dependency (1-cycle penalty). Branch and jump prediction is implemented only in E51, with 3-cycles penalty in case of missed prediction. Hit prediction accuracy was also set to 90%. In E20 branches are predicted not-taken, incurring no penalty in case of hit. Taken branches and unconditional jumps incur 1-cycle penalty. Figure 6 reveals that considering SiFive cores E51 and E20, shift-based MAC is the best approach. All shift performance bars are below the reference red line with execution time reduction of about 2% to 15%. Execution time experiments point out that from about 4-cycles multiplications latency onwards, shift-based MAC may starts to perform better.

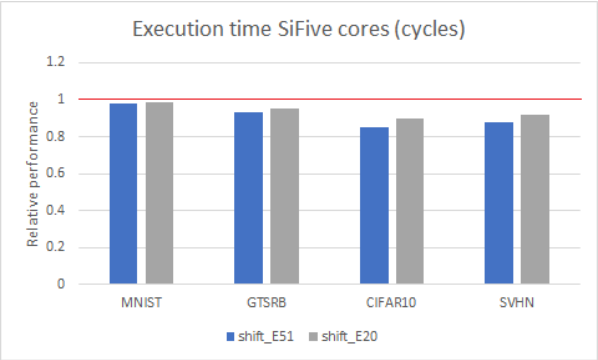

Figure 6. Shift performance w.r.t. E51 and E20 cores multiplier

IV. Discussion

Despite this work dealing only with multiplication, M extension requires extra circuitry in order to support integer multiplication and division instructions. In [5] authors prototyped in FPGA (Xilinx Artix-7) the RVCoreP implementing RV32I and RV32IM ISAs aiming to evaluate the area impact of M extension. They report that LUT resource utilization increases around 52% for RV32IM (radix-4) and around 40% for RV32IM (DSP) compared to RV32I implementation. On top of that, RV32IM (DSP) uses 4 FPGA DSP blocks. RV32I achieves a bit higher frequency than RV32IM (regardless of multiplier type). It is noteworthy that in subsection III-B, execution times were reported in clock cycles. Since M extension support reduces maximum frequency, the performance of shift-based MAC running on RV32I implementation compared to multiplication-based MAC on RV32IM can be even better when measuring execution time in seconds. In order to highlight that, we took FPGA implementation reports from [5] and [6]. Table III compares inference throughput (inferences/second) between shift-based MAC running on RV32I at 174MHz [6] and multiplication-based MAC (radix-4) running on RV32IM at 169MHz [5]. CIFAR10 and SVHN present almost the same throughput because the same CNN model is used (Figure 2). Note that speedup gains concerns only software changes, with a negligible memory footprint impact on integer CNN inference code due to MAC implementation (Listing 2).

Table III
Inference throughput.

	shift [6] (RV32I@174MHz)	mult [5] (RV32IM@169MHz)	Speedup
MNIST	125.3	86.1	1.45x
GTSRB	12.5	7.8	1.6x
CIFAR10	0.8	0.4	2x
SVHN	0.78	0.4	1.95x

V. Conclusion

The direct replacement of multiplication (*) by shift ($\ll$) in a high-level language, without including any type of extra statements, likely will result in the execution time reduction. Shift instructions typically have lower latency, but there are exceptions, such as the Xtensa architecture [10], which does not provide single-instruction shifts where the shift amount is a register operand. However, when such a replacement requires changes to the source code, such as adding flow control statements, it is difficult to predict what happens to performance without further investigation.

This work addressed a lower-level analysis to unveil the shift advantages over multiplication when RISC-V processors execute convolution neural networks, which in many cases are hidden by instruction count. Thanks to the flexibility of the RISC-V architecture, we showed that only the ISA base integer (RV32I) is enough to obtain good results in terms of execution time and accuracy of convolution neural networks.

REFERENCES

[1] G. Akkad, A. Mansour, and E. Inaty, "Embedded deep learning accelerators: A survey on recent advances," *IEEE Transactions on Artificial Intelligence*, vol. 1, pp. 1–19, sep 5555.

[2] G. Kyriakides and K. G. Margaritis, "An introduction to neural architecture search for convolutional networks," *CoRR*, vol. abs/2005.11074, 2020.

[3] A. Paszke, S. Gross, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[4] Imperas, "riscvovpsim - free imperas risc-v instruction set simulator," 2024. Accessed 03/27/2024.

[5] M. A. Islam, H. Miyazaki, and K. Kise, "Rvcorep-32im: An effective architecture to implement mul/div instructions for five stage risc-v soft processors," 2020.

[6] H. Miyazaki, T. Kanamori, M. A. Islam, and K. Kise, "Rvcorep: An optimized risc-v soft processor of five-stage pipelining," *IEICE Transactions on Information and Systems*, vol. E103.D, p. 2494–2503, Dec. 2020.

[7] C. K. Lin and S. J. Tarsa, "Branch prediction is not a solved problem: Measurements, opportunities, and future directions," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 228–238, 2019.

[8] SiFive Inc, "Sifive e20 core complex manual v1p0," 2018. Accessed 08/18/2024.

[9] SiFive Inc, "Sifive e51 core complex manual v2p0," 2018. Accessed 08/18/2024.

[10] Cadence Design Systems Inc, "Xtensa instruction set architecture (isa) summary," 2022. Accessed 08/18/2024.