RIVETS: An Efficient Training and Inference Library for RISC-V with Snitch Extensions

Andrei Ivanov[1]*, Timo Schneider[1], Luca Benini[2] and Torsten Hoefler[1]

[1]Department of Computer Science, ETH Zurich
[2]Department of Information Technology and Electrical Engineering, ETH Zurich

Abstract

The openness and customizability of RISC-V makes it a compelling platform for executing deep learning applications. We present a library of efficient deep learning kernels for RISC-V hardware, addressing the challenge of achieving optimal performance in both training and inference. The library adopts the Snitch extensions to RISC-V, adheres to the OneDNN interface, and offers portable baseline implementations as well as platform-specific optimizations. Our optimizations that leverage Snitch extensions allow us to achieve up to 0.87 flops per clock cycle. RIVETS is a valuable tool for deep learning practitioners and researchers using RISC-V, providing portability, compatibility with other frameworks, and a baseline for performance comparison.

Introduction

The growing interest in running machine learning workloads on edge devices has led to the development of processor architecture variations geared toward power efficiency. The ARM architecture has long been a popular choice for energy-efficient computing. Recently, there has been noticeable attention to exploring the applicability of open architectures such as RISC-V [1].

While today's RISC-V cores used in practice are low-power, inference-oriented [2], there is rising interest in using this architecture in more demanding workloads, including training deep learning models [3].

Deep learning libraries The need to run neural networks on edge devices caused the development of various libraries targeting specific embedded and low-power processor architectures.

Popular frameworks for deep learning are TensorFlow and PyTorch. TensorFlow Lite is a toolkit for deploying TensorFlow models on edge devices. It also has an extension, TensorFlow Lite for Microcontrollers targeting highly constrained devices. These libraries have support for RISC-V, ranging from being compatible with the toolchain to having kernels optimized for specific hardware. ONNXRuntime is a lower-level tool that supports training and inference of models created in the aforementioned frameworks. It also has a port to RISC-V, but its optimizations are mainly focused on integer-precision kernels.

The frameworks discussed above typically support RISC-V through backends that utilize kernels from hardware-specific libraries. OneDNN contains experimental support for 64-bit RISC-V architectures, but it currently only has a maximum pooling operator optimized with vector extensions. XNNPACK is a library of kernels used by popular ML frameworks, including TensorFlow and PyTorch. It supports only 6 vectorized floating-point element-wise kernels specifically targeting RISC-V. CMSIS-NN is a neural network kernel library for the ARM architecture. It only supports integer arithmetic. There are different CMSIS-NN library ports for RISC-V architectures. One of them, PULP-NN, is written for the GAP-8 SoC and uses non-portable platform-specific features. The other port, muRISCV-NN, targets vector and SIMD extensions, but only works with integer data.

There are multiple popular deep learning APIs. **CuDNN** contains a library of forward and backward primitives but its API is not standardized as an open specification. In contrast, **ONNX** is an open standard, but its backpropagation specification is not as comprehensive. **OneDNN** defines an open specification with explicit care of inference and training. The design of both OneDNN and CuDNN includes two kinds of APIs: a legacy operator-level API and a graph API that potentially allows inter-operator optimization. In our library, we follow the OneDNN operator-level API to simplify adoption. We maintain a simple build system that provides RIVETS as a static library to simplify integration on new platforms. Unlike many existing implementations, we pay special attention to floating-point computation.

RISC-V extensions One of the extensions that entered the RISC-V standard early is the Packed SIMD extension. It is focused on integer SIMD operations. Vector floating point instructions are supported by recently standardized RISC-V Vector extension.

In addition to these standard processor hardware extensions, there are others which try to remove the instruction fetching bottleneck occurring in the hardware. The 32-bit RISC-V core, Snitch [4], adds Stream Semantic Registers (**SSR**), floating-point re-

*Corresponding author: `{firstname}.{lastname}@inf.ethz.ch`

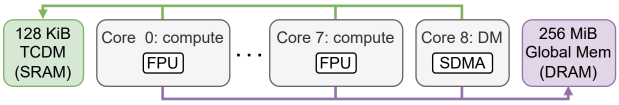


Figure 1: *Overview of the Snitch cluster environment.*

peat (**FREP**), and asynchronous data movement (**SDMA**) extensions. Snitch also has non-standard **smallFloat** extension to support 8, 16, and 32-bit floating point operations on 64-bit wide vectors.

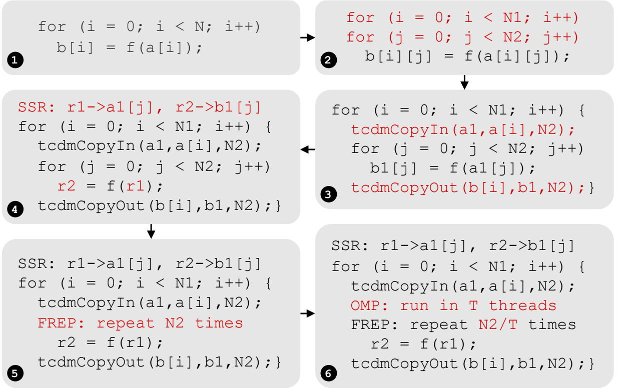
A Snitch cluster is composed out of eight compute and one data movement core as shown in Figure 1. It has access to global memory and faster shared memory (TCDM). We chose Snitch as a reference optimization and performance evaluation platform for RIVETS because of the significant number of floating-point extensions applicable to DNN kernels.

With the **SDMA** extension, the data movement core (DM) can send requests to the hardware DMA queue and checks the completion status of the request. This feature accelerates the transfer of strided data between global memory and TCDM. By supporting asynchronous transfers, it allows to overlap communication with computation. Compute cores send memory copy requests to the DM core via TCDM.

The **SSR** extension avoids the overhead of fetching data movement instructions in a hot loop: instead of incrementing a pointer to iterate over array elements the increment is performed implicitly and accesses are made through floating point registers instead of memory dereferencing.

The **FREP** instruction can repeat the next N instructions M times, saving the cycles that would otherwise be needed to increment the loop counter and branch to the start of the loop body.

Implementation



1
```
for (i = 0; i < N; i++)
  b[i] = f(a[i]);
```

2
```
for (i = 0; i < N1; i++)
for (j = 0; j < N2; j++)
  b[i][j] = f(a[i][j]);
```

3
```
for (i = 0; i < N1; i++) {
  tcdmCopyIn(a1,a[i],N2);
  for (j = 0; j < N2; j++)
    b1[j] = f(a1[j]);
  tcdmCopyOut(b[i],b1,N2);}
```

4
```
SSR: r1->a1[j], r2->b1[j]
for (i = 0; i < N1; i++) {
  tcdmCopyIn(a1,a[i],N2);
  for (j = 0; j < N2; j++)
    r2 = f(r1);
  tcdmCopyOut(b[i],b1,N2);}
```

5
```
SSR: r1->a1[j], r2->b1[j]
for (i = 0; i < N1; i++) {
  tcdmCopyIn(a1,a[i],N2);
  FREP: repeat N2 times
    r2 = f(r1);
  tcdmCopyOut(b[i],b1,N2);}
```

6
```
SSR: r1->a1[j], r2->b1[j]
for (i = 0; i < N1; i++) {
  tcdmCopyIn(a1,a[i],N2);
  OMP: run in T threads
  FREP: repeat N2/T times
    r2 = f(r1);
  tcdmCopyOut(b[i],b1,N2);}
```

Figure 2: *A sketch of the optimization sequence. Loop unrolling and asynchronous copies are omitted for brevity.*

Operation	peak ops/cycle	latency [cycles]
fma, add, mul	1	4
min, max, abs	1	1
sqrt, div	0.05	22
byte transfer	60	166

Table 1: *Operations per cycle in the Snitch core.*

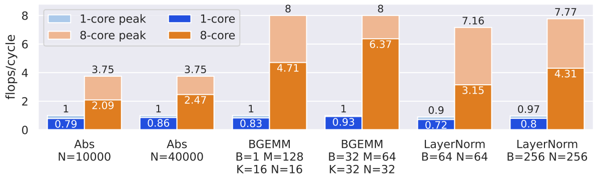


Figure 3: *Performance of kernels on Snitch platform.*

For optimal portability, we developed RIVETS in C and equipped it with a preconfigured environment for compilation and profiling on the Snitch platform. It supports both plain RISC-V kernels and kernels optimized to use the Snitch extensions, as shown in Figure 2. It works with a flexible number of cores, clusters and memory sizes. We evaluate our kernels using cycle accurate simulation in the default Snitch cluster configuration provided in Zaruba et al. [4].

Figure 3 reports the number of flops per clock cycle for each kernel. To assess how far these results are from the theoretical peak, we use the platform's throughput parameters shown in the Table 1 and the theoretical number of flops required to execute the kernel. For example, BGEMM performs `B·M·K·N` fma operations, while layer normalization does `B(5N+2)` fma- and `2B` long-latency (div, sqrt) operations. The peak numbers in Figure 3 refer to computational bandwidth only. The performance of element-wise kernels such as Abs is IO bound. For small inputs in multi-core implementations, the latency and bandwidth costs of transferring data into the TCDM memory significantly hampers performance.

Our library has been made available to the public[1].

Acknowledgement

This work has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 101034126 (EU-Pilot).

References

[1] Marcia Sahaya Louis et al. "Towards Deep Learning using TensorFlow Lite on RISC-V". In: 2019.

[2] Pasquale Davide Schiavone et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications". In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2017, pp. 1–8.

[3] Angelo Garofalo et al. "DARKSIDE: A Heterogeneous RISC-V Compute Cluster for Extreme-Edge On-Chip DNN Inference and Training". In: *IEEE Open Journal of the Solid-State Circuits Society* 2 (2022), pp. 231–243.

[4] Florian Zaruba et al. "Snitch: A tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads". In: *IEEE Transactions on Computers* (2020).

[1] `https://github.com/spcl/rivets`