ANDREI IVANOV, TIMO SCHNEIDER, LUCA BENINI, TORSTEN HOEFLER

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

Deep Learning libraries for RISC-V?

<table>
<thead>
<tr>
<th></th>
<th>muRISCV-NN</th>
<th>PULP-NN</th>
<th>XNNPACK</th>
<th>OneDNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensions</td>
<td>Vector &quot;V&quot;</td>
<td>Xpulp [1]</td>
<td>Vector &quot;V&quot;</td>
<td>Vector &quot;V&quot;</td>
</tr>
<tr>
<td></td>
<td>Packed &quot;P&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>Integer</td>
<td>Subbyte-quantized integer</td>
<td>Floating-point</td>
<td>Floating-point</td>
</tr>
<tr>
<td>Kernels</td>
<td>Softmax, Pooling, Conv, LSTM, SVD, ReLu, Sigmoid</td>
<td>Add, Pooling, Linear, MatMul</td>
<td>Sqrt, Sqr, Abs, Neg, HSwish, Clamp</td>
<td>Pooling</td>
</tr>
</tbody>
</table>

The missing parts:
- Optimized for **performant** floating-point
- Focus on **training** and inference

---

Deep Learning API specifications

NVIDIA cuDNN
- NVIDIA GPU

OneDNN (part of Intel oneAPI)
- Intel CPU/GPU
- Arm 64-bit AArch64
- Experimental:
  - NVIDIA/AMD GPU
  - OpenPOWER (PPC64),
  - IBMz (s390x)
  - RISC-V

ONNX: Open Neural Network Exchange
- ONNXRuntime: relies on "Execution Providers"
- **Training**: NVIDIA/AMD GPU (cuDNN, ROCm)
- **Inference**: x86_32, x86_64, ARM32v7, ARM64, PPC64LE

Common kernels:
- Convolution
- Matrix multiplication
- Pooling
- Statistical normalization
- Activation functions
Target platform

- Snitch cluster [1]
- TCDM: Tightly Coupled Data Memory – Programmable scratchpad memory
- Extensions targeting fast floating point computations

Snitch Extensions

- SSR: Stream Semantic Registers [1]
- FREP: Floating-point repetition
- SDMA: Snitch asynchronous DM: 1D and 2D asynchronous copies
- SmallFloat [2]: Support of fp8, fp16, fp32, fp64 inside the 64-bit register


Extension: SSR

Configuration:
- size = 3, stride = 1
- size = 2, stride = 2
- size = 2, stride = 3

Motivation: remove explicit loads and stores from instruction flow

Configuration:
- size = 3, stride = 1
- size = 2, stride = 2
- size = 2, stride = 3

address space

fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3

SSR 0 → ft0
SSR 1 → ft1
SSR 2 → ft2
Extension: FREP

Motivation: remove branching and loop counter increments

```
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
fadd.d ft3, ft0, ft3
```

"repeat 1 instruction 5 times"

```
frep.o 5, 1, 0, 0
fadd.d ft3, ft0, ft3
```
Example: Layer Normalization

\[ \text{dst}(b, n) = \gamma(n) \cdot \frac{\text{src}(b, n) - \mu(b)}{\sqrt{\sigma^2(b)}} + \beta(n) \]

Tiling with asynchronous copies

```
TCDM
```

```
compute buffer
```

```
compute buffer
```

```
Time
```

```
Compute core(s)
```

```
Data movement core
```

```
CopyIn
```

```
CopyIn
```

```
CopyOut
```

```
CopyIn
```

```
CopyOut
```

```
CopyIn
```

```
CopyOut
```

```
CopyOut
```

```
CopyIn
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```

```
CopyOut
```
Example: Layer Normalization

\[
dst(b, n) = \gamma(n) \cdot \frac{src(b, n) - \mu(b)}{\sqrt{\sigma^2(b)} + \epsilon} + \beta(n)
\]

4-stage computation

```c
for (size_t b = 0; b < B; b++) {
    mu[b] = 0;
    for (size_t n = 0; n < N; n++) {
        mu[b] += src[b * N + n];
    }
    mu[b] /= N;
}
```

stage 1: compute mean

```c
sigma[b] = 0;
for (size_t n = 0; n < N; n++) {
    dst[b * N + n] = src[b * N + n] - mu[b];
}
```

stage 2: find difference

```c
for (size_t n = 0; n < N; n++) {
    sigma[b] += SQR(dst[b * N + n]);
}
```

stage 3: compute denominator

```c
sigma[b] = 1.0 / SQRT(sigma[b] / (N - 1) + eps);
for (size_t n = 0; n < N; n++) {
    dst[b * N + n] = gamma[n] * dst[b * N + n] * sigma[b] + beta[n];
}
```

stage 4: compute result
Example: Layer Normalization

$$
dst(b, n) = \gamma(n) \cdot \frac{src(b, n) - \mu(b)}{\sqrt{\sigma^2(b) + \epsilon}} + \beta(n)
$$

for (size_t b = 0; b < B; b++) {
    mu[b] = 0;
    for (size_t n = 0; n < N; n++) {
        mu[b] += src[b * N + n];
    }
    mu[b] /= N;
    sigma[b] = 0;
    for (size_t n = 0; n < N; n++) {
        dst[b * N + n] = src[b * N + n] - mu[b];
    }
    for (size_t n = 0; n < N; n++) {
        sigma[b] += SQR(dst[b * N + n]);
    }
    sigma[b] = 1.0 / SQRT(sigma[b] / (N - 1) + eps);
    for (size_t n = 0; n < N; n++) {
        dst[b * N + n] = gamma[n] * dst[b * N + n] * sigma[b] + beta[n];
    }
}
Example: Layer Normalization

\[ \text{dst}(b, n) = \gamma(n) \cdot \frac{\text{src}(b, n) - \mu(b)}{\sqrt{\sigma^2(b)} + \varepsilon} + \beta(n) \]

```c
for (size_t b = 0; b < B; b++) {
    mu[b] = 0;
    for (size_t n = 0; n < N; n++) {
        mu[b] += src[b * N + n];
    }
    mu[b] /= N;
    sigma[b] = 0;
    for (size_t n = 0; n < N; n++) {
        dst[b * N + n] = src[b * N + n] - mu[b];
    }
    for (size_t n = 0; n < N; n++) {
        sigma[b] += SQR(dst[b * N + n]);
    }
    sigma[b] = 1.0 / SQRT(sigma[b] / (N - 1) + eps);
    for (size_t n = 0; n < N; n++) {
        dst[b * N + n] = gamma[n] * dst[b * N + n] * sigma[b] + beta[n];
    }
}
```
Example: Layer Normalization

<table>
<thead>
<tr>
<th>Functional block [1]</th>
<th>Operation</th>
<th>Peak ops/cycle</th>
<th>latency [cycles]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDMUL</td>
<td>fma, add, mul</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DIVSQRT</td>
<td>sqrt, div</td>
<td>0.05</td>
<td>22</td>
</tr>
<tr>
<td>COMP</td>
<td>min, max, abs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SDMA</td>
<td>byte transfer</td>
<td>60</td>
<td>166</td>
</tr>
</tbody>
</table>

```
for (size_t i = 0; i < N; i++) {
    x += y[i];
}
```

```
for (size_t i = 0; i < N; i++) {
    x0 += y0[i];
    x1 += y1[i];
    x2 += y2[i];
    x3 += y3[i];
}
```

```
3 cycle delay
```
```
no delay
```

Memory bank conflicts

- TCDM: 32 banks 4 KiB each
- 4 superbanks: group of 8 consecutive banks
LayerNorm execution timeline

SSR configuration: stride, size

LayerNorm Batch=96 N=32

- Longer first iteration: cold instruction cache
- Copy in and out of TCDM

LayerNorm Batch=128 N=128

Barrier synchronization
Evaluation

### Functional block operation comparison

<table>
<thead>
<tr>
<th>Functional block</th>
<th>Operation</th>
<th>Peak ops/cycle</th>
<th>latency [cycles]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDMUL</td>
<td>fma, add, mul</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DIVSQRT</td>
<td>sqrt, div</td>
<td>0.05</td>
<td>22</td>
</tr>
<tr>
<td>COMP</td>
<td>min, max, abs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SDMA</td>
<td>byte transfer</td>
<td>60</td>
<td>166</td>
</tr>
</tbody>
</table>

Peak compute [cycles]
- Abs: $N$ COMP
- BGEMM: $B \cdot M \cdot K \cdot N$ ADDMUL
- LayerNorm: $B(5N + 2)$ ADDMUL + $2B$ DIVSQRT

"Abs" I/O requirements [bytes/cycle]:
- Required: $8 \cdot 2 \cdot 8 > 60$
- Available: $3.75 \cdot 2 \cdot 8 = 60$

"Abs" microkernel
frep.o a0, 1, 0, 0
fabs.d ft1, ft0
fp64
SSR memory accesses
Moving forward: end-to-end model support

Deep Learning Models
- BERT
- YOLOv5
- ...

DaCeML [1] frontend
- PyTorch
- Other DNN Frameworks
- TensorFlow
- ONNX

DaCe [2]: Data-Centric Parallel Programming
- Data-centric IR (SDFG)
- Coarse-Grained Transformations
- Local Data Movement Reduction
- Global Data Layout Optimization
- Hardware Specialization

Multi-Level Optimization via Progressive Lowering

RIVETS
- GEMM, Convolution, LayerNorm, Softmax, BatchNorm,
  ...
- SSR
- FREP
- SDMA

Conclusions

Example: Layer Normalization

```plaintext
      0 = depth of a distance between src and dst array starts in memory
      P = lane grouping

  \[ \text{dst}(b, n) = y \times \text{src}(b, n) - \mu(b) \sqrt{\text{var}(b, n)} + \beta(b) \]

  for \( \text{for}( b = 0; b < B; b += P) \) { \text{for} \( (l = 0; l < L; l += P) \) { \text{for} \( (n = 0; n < N; n += P) \) } { \text{for} \( (d = 0; d < D; d += P) \) { \text{for} \( (l = 0; l < L; l += P) \) { \text{for} \( (n = 0; n < N; n += P) \) } \} \} \} \}

  \text{read src} \quad \text{write dst} \quad \text{read mem} \quad \text{write mem} \quad \text{write y} \quad \text{write f}

  \text{shaves:} \quad [N, 2, 2, N] \quad [N, 0, 1] \quad \text{shapes:} \quad [B, 2, Z, N] \quad [B, N, 2] \quad \text{strides:} \quad [B, N, 1] \quad [N, 1, P]

  \text{sizes:} \quad B, N, Z, L, D, P

  \text{observation:} \quad B = 4, N = 4, P = 8, D = 32, L = 1024

  \text{spcl.ethz.ch}
```

More of SPCL's research:

- [youtube.com/@spcl](youtube.com/@spcl) - 150+ Talks
- [twitter.com/spcl_eth](twitter.com/spcl_eth) - 1.2K+ Followers
- [github.com/spcl](github.com/spcl) - 2K+ Stars