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?

	muRISCV-NN	PULP-NN	XNNPACK	OneDNN
Extensions	Vector "V" Packed "P"	Xpulp [1]	Vector "V"	Vector "V"
Precision	Integer	Subbyte-quantized integer	Floating-point	Floating-point
Kernels	Softmax, Pooling, Conv, LSTM, SVD, ReLu, Sigmoid	Add, Pooling, Linear, MatMul	Sqrt, Sqr, Abs, Neg, HSwish, Clamp	Pooling

The missing parts:

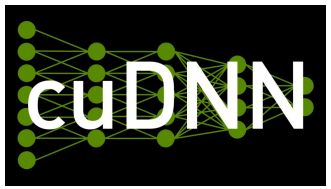
- Optimized for **performant** floating-point
- Focus on **training** and inference

[1] M. Gautschi *et al.*, "**Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices**," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700-2713, Oct. 2017, doi: 10.1109/TVLSI.2017.2654506.

Deep Learning API specifications

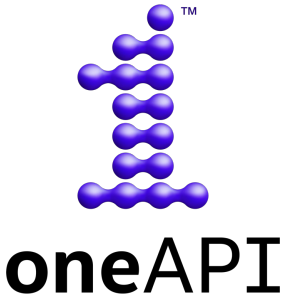
NVIDIA cuDNN

- NVIDIA GPU



OneDNN (part of Intel oneAPI)

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



Common kernels:

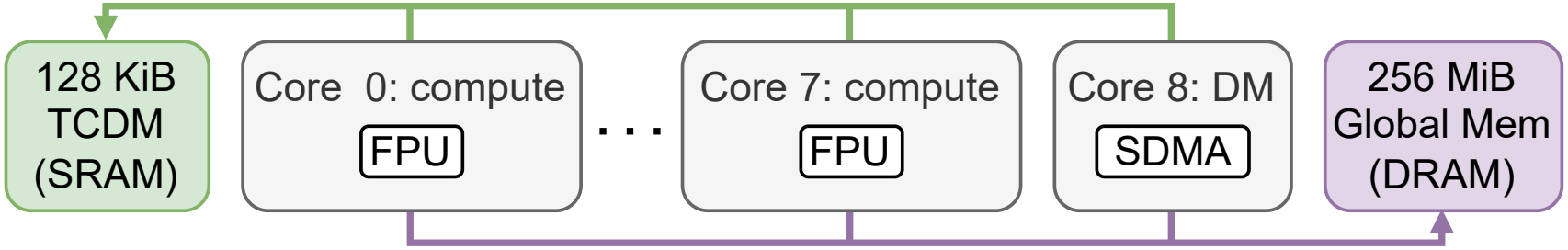
- Convolution
- Matrix multiplication
- Pooling
- Statistical normalization
- Activation functions

ONNX: Open Neural Network Exchange

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

Target platform

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



[1] F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "**Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads**," in *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845-1860, 1 Nov. 2021, doi: 10.1109/TC.2020.3027900.

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**

[1] F. Schuiki, F. Zaruba, T. Hoefler and L. Benini, "**Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores**," in *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212-227, 1 Feb. 2021, doi: 10.1109/TC.2020.2987314.

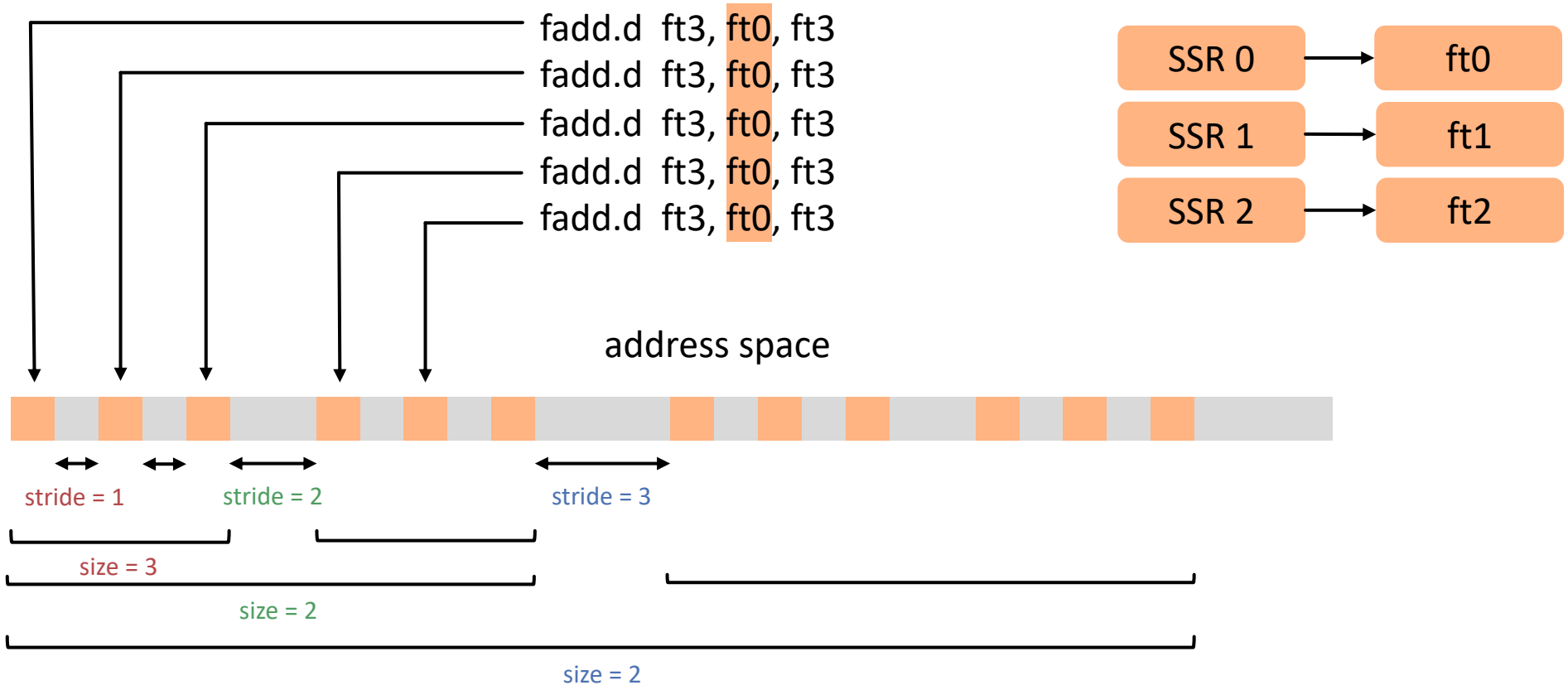
[2] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu and L. Benini, "**Design and Evaluation of SmallFloat SIMD extensions to the RISC-V ISA**," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 654-657, doi: 10.23919/DATE.2019.8714897.

Extension: SSR

Configuration:

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

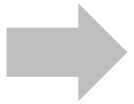
Motivation: remove explicit loads and stores from instruction flow



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
```



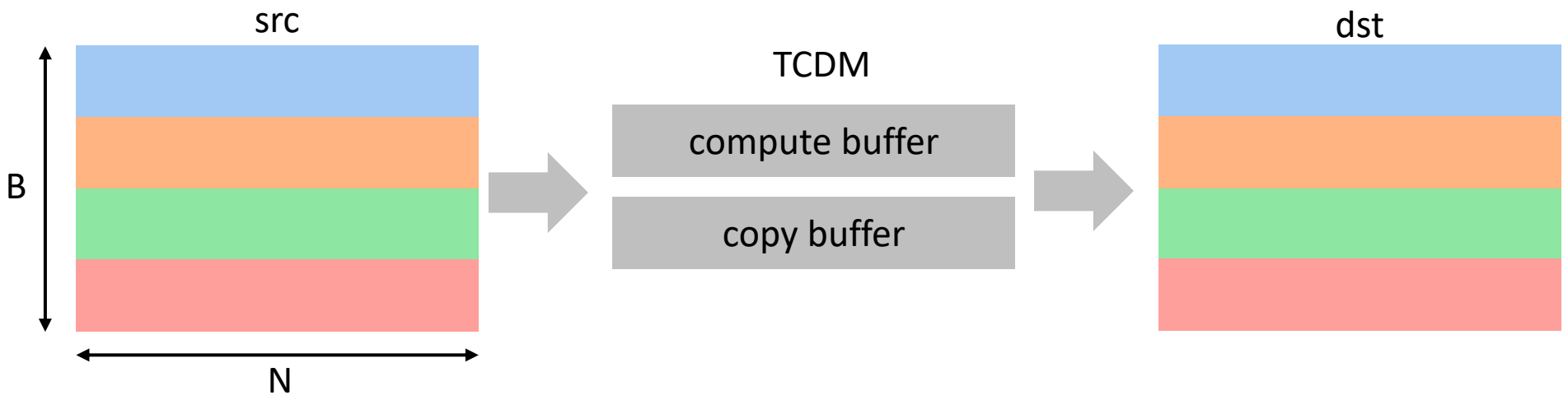
"repeat 1 instruction 5 times"

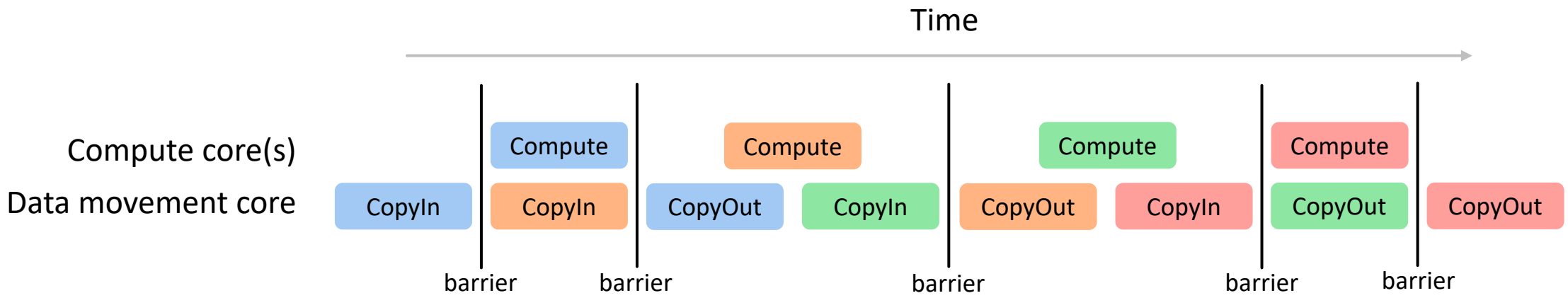
```
frep.o  5, 1, 0, 0
fadd.d  ft3, ft0, ft3
```

Example: Layer Normalization

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

Tiling with asynchronous copies





Example: Layer Normalization

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

4-stage computation

```
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];
    }
}
```

stage 1: compute mean

stage 2: find difference

stage 3: compute denominator

stage 4: compute result

Example: Layer Normalization

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

D = dst-src // distance between src and dst array starts in memory
P = beta-gamma

	SSR 0	SSR 1	SSR 2
shapes:	[B, 2, 2, N]	[B, N, 2]	[B, 2, N]
strides:	[N, D, 0, 1]	[N, 1, P]	[N, 0, 1]

```
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];
    }
}
```

read src

read src — write dst

read dst

read dst — read γ — write dst

read β

Example: Layer Normalization

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

```
for (size_t b = 0; b < B; b++) {
    mu[b] = 0;
    for (size_t n = 0; n < N; n++) {          → frep
        mu[b] += src[b * N + n];
    }
    mu[b] /= N;
    sigma[b] = 0;
    for (size_t n = 0; n < N; n++) {          → frep
        dst[b * N + n] = src[b * N + n] - mu[b];
    }
    for (size_t n = 0; n < N; n++) {          → frep
        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++) {          → frep
        dst[b * N + n] = gamma[n] * dst[b * N + n] * sigma[b] + beta[n];
    }
}
```

FREP: repeat loop body N times

Example: Layer Normalization

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

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

```
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
```

3 cycle delay

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

```
fadd.d  ft3, ft0, ft3
fadd.d  ft4, ft0, ft4
fadd.d  ft5, ft0, ft5
fadd.d  ft6, ft0, ft6
fadd.d  ft3, ft0, ft3
```

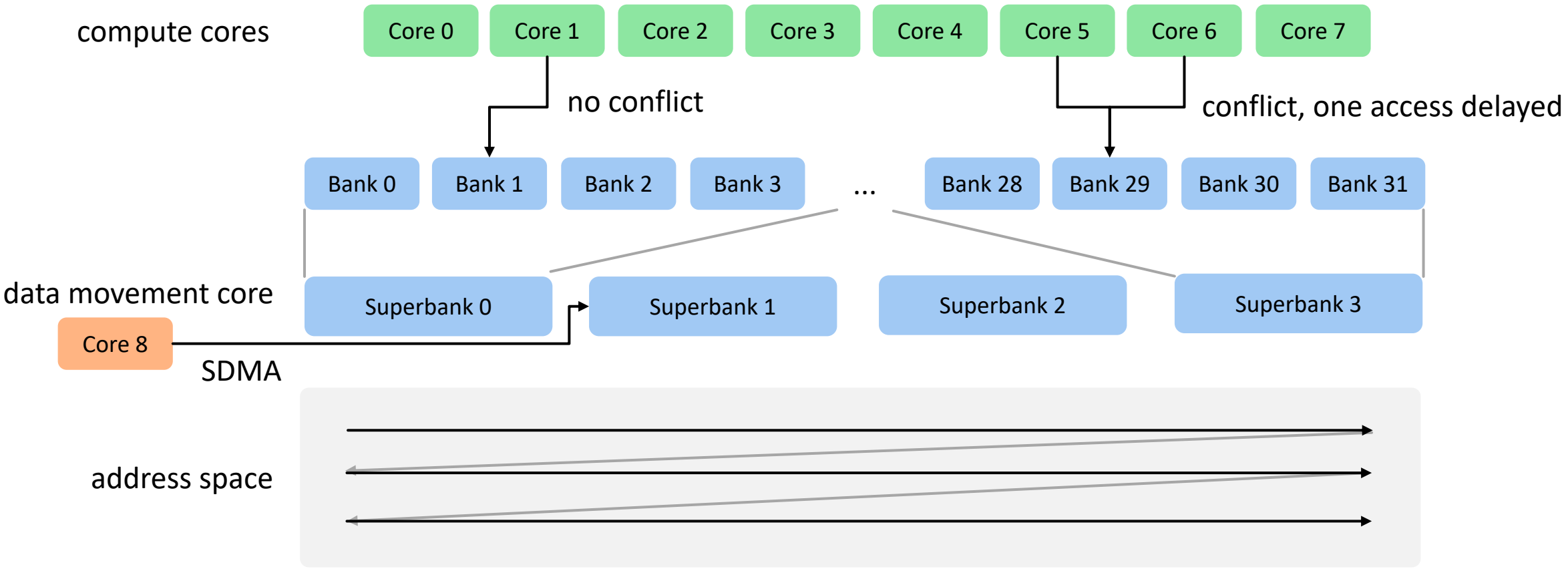
no delay

SSR 0

[1] S. Mach, F. Schuiki, F. Zaruba and L. Benini, "**FPnew: An Open-Source Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing**," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774-787, April 2021, doi: 10.1109/TVLSI.2020.3044752.

Memory bank conflicts

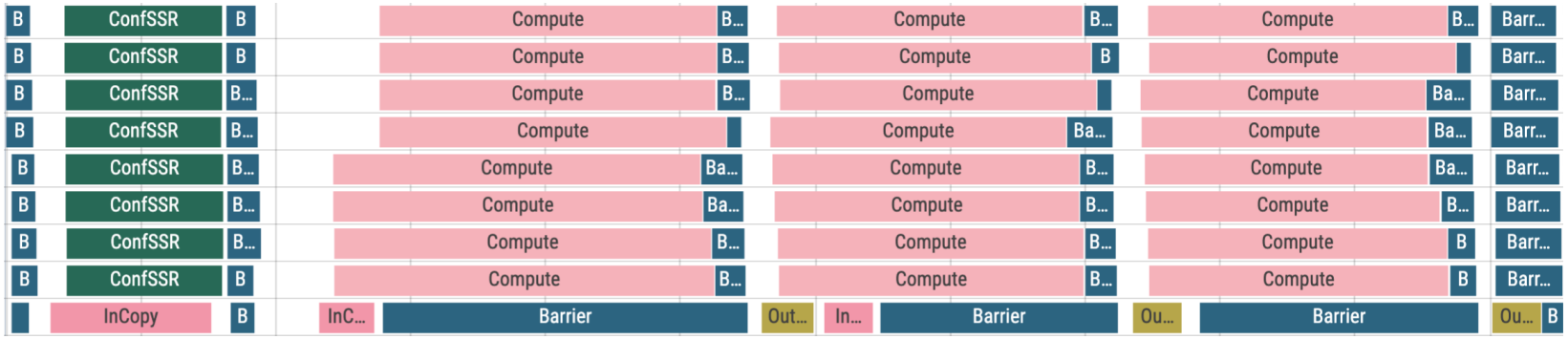
- **TCDM: 32 banks 4 KiB each**
- **4 superbanks: group of 8 consecutive banks**



LayerNorm execution timeline

SSR configuration: stride, size

Layernorm Batch=96 N=32

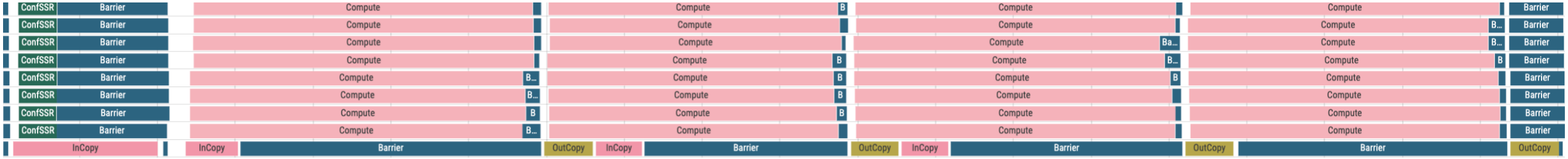


Barrier synchronization

Longer first iteration:
cold instruction cache

Copy in and out of TCDM

Layernorm Batch=128 N=128



Evaluation

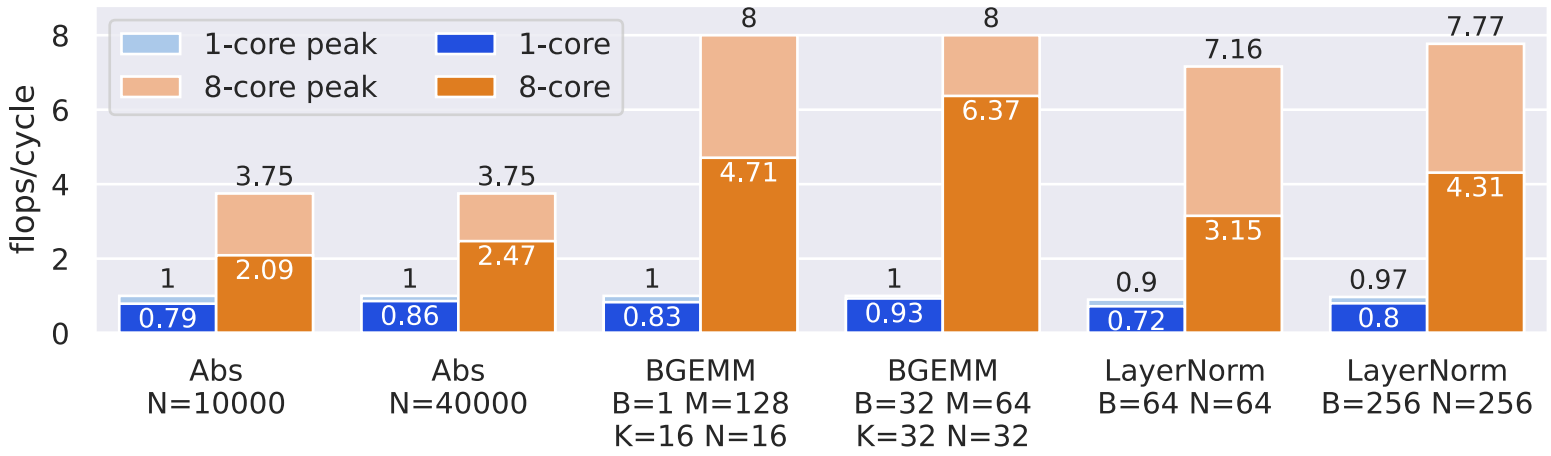
Functional block	Operation	Peak ops/cycle	latency [cycles]
ADDMUL	fma, add, mul	1	4
DIVSQRT	sqrt, div	0.05	22
COMP	min, max, abs	1	1
SDMA	byte transfer	60	166

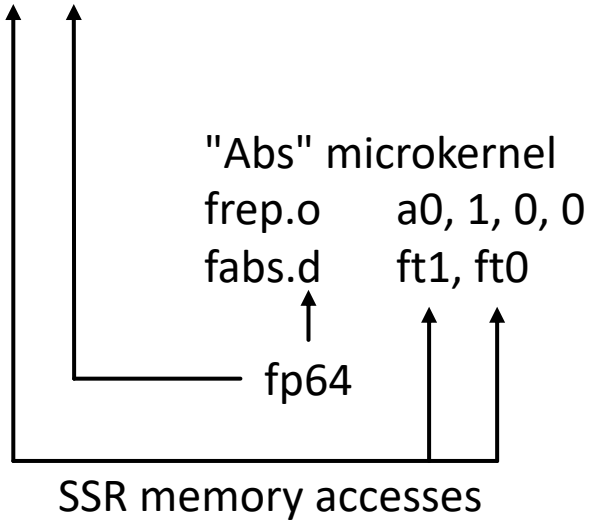
Peak compute [cycles]

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

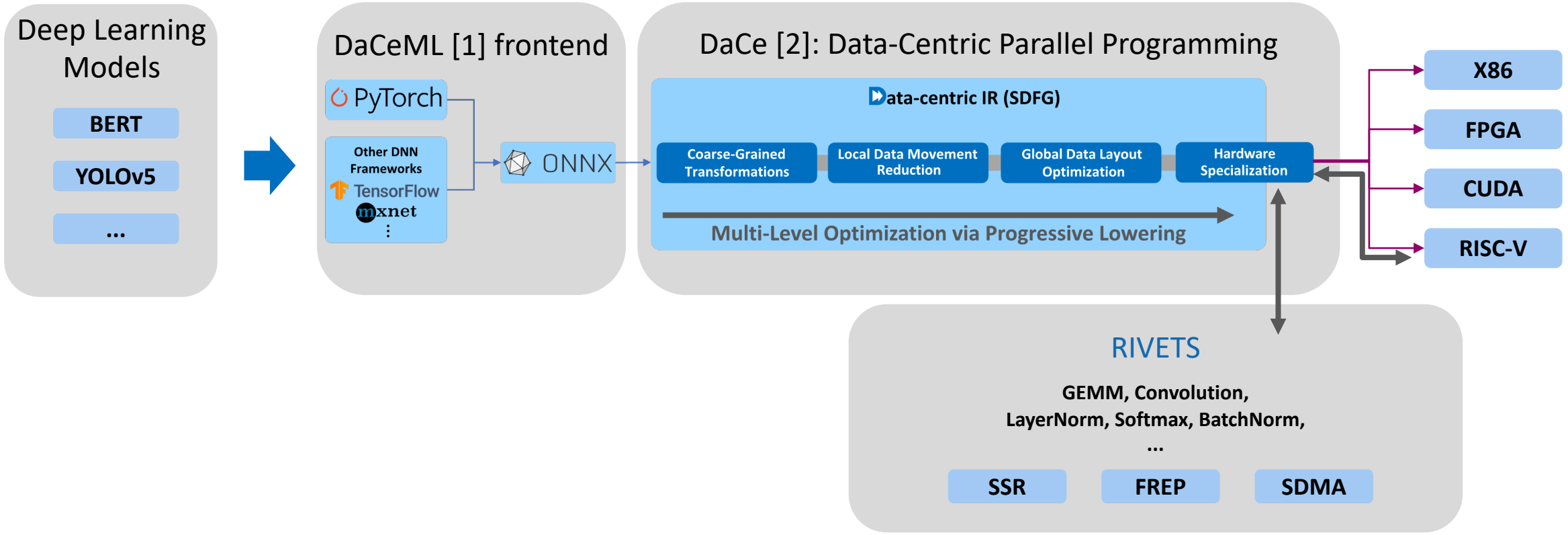
"Abs" I/O requirements [bytes/cycle]:

- Required: $8 \cdot 2 \cdot 8 > 60$
- Available: $3.75 \cdot 2 \cdot 8 = 60$





Moving forward: end-to-end model support



[1] Oliver Rausch, Tal Ben-Nun, Nikoli Dryden, Andrei Ivanov, Shigang Li, and Torsten Hoefler. 2022. **A data-centric optimization framework for machine learning**. In Proceedings of the 36th ACM International Conference on Supercomputing (ICS '22). Association for Computing Machinery, New York, NY, USA, Article 36, 1–13.
[2] Tal Ben-Nun, Johannes de Fine Licht, Alexandros N. Ziogas, Timo Schneider, and Torsten Hoefler. 2019. **Stateful dataflow multigraphs: a data-centric model for performance portability on heterogeneous architectures.** In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19). Association for Computing Machinery, New York, NY, USA, Article 81, 1–14.

Conclusions

Problem

Deep Learning libraries for RISC-V?

	muRISCV-NN	PULP-NN	XNNPACK	OneDNN
Extensions	Vector "V" Packed "P"	Xpulp [1]	Vector "V"	Vector "V"
Precision	Integer	Subbyte-quantized integer	Floating-point	Floating-point
Kernels	Softmax, Pooling, Conv, LSTM, SVD, ReLu, Sigmoid	Add, Pooling, Linear, MatMul	Sqrt, Sqr, Abs, Neg, HSwish, Clamp	Pooling

The missing parts:

- Optimized for **performant** floating-point
- Focus on **training** and inference

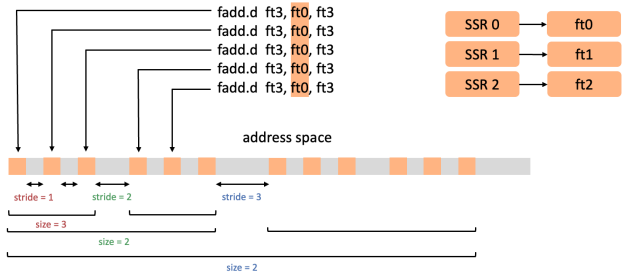
[1] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700-2713, Oct. 2017, doi: 10.1109/TVLSI.2017.2654506.

Extension: SSR

Configuration:

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

Motivation: remove explicit loads and stores from instruction flow



Example: Layer Normalization

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

D = dst-src // distance between src and dst array starts in memory
P = beta-gamma

	SSR 0	SSR 1	SSR 2
shapes:	[B, 2, 2, N]	[B, N, 2]	[B, 2, N]
strides:	[N, D, 0, 1]	[N, 1, P]	[N, 0, 1]

```
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];                          read src
    }
    mu[b] /= N;
    sigma[b] = 0;
    for (size_t n = 0; n < N; n++) {
        dst[b * N + n] = src[b * N + n] - mu[b];          read src    write dst
    }
    for (size_t n = 0; n < N; n++) {
        sigma[b] += SQR(dst[b * N + n]);                  read dst
    }
    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];   read dst   read γ   write dst
    }                                                                      read β
}
```

Evaluation

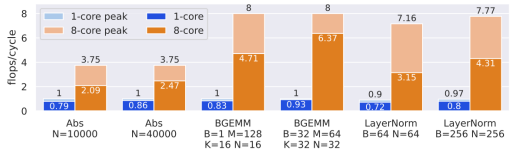
Functional block	Operation	Peak ops/cycle	latency [cycles]
ADDMUL	fma, add, mul	1	4
DIVSQRT	sqrt, div	0.05	22
COMP	min, max, abs	1	1
SDMA	byte transfer	60	166

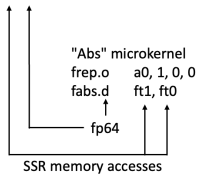
Peak compute [cycles]

- 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: $3.75 \cdot 2 \cdot 8 = 60$





More of SPCL's research:

youtube.com/@spcl — 150+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

github.com/spcl — 2K+ Stars

… or spcl.ethz.ch