PULP-TrainLib: On-Device Learning on Parallel Ultra-Low-Power MCUs

Davide Nadalini

d.nadalini@unibo.it

Overview

- PULP-TrainLib
- GVSoC Simulator and Performance Metrics
- Analyzing and Generating On-Device Learning Code
- Parallelizing FP32 Code
- Optimizing FP16 Code with SIMD
- Comparison with the State-of-the-Art

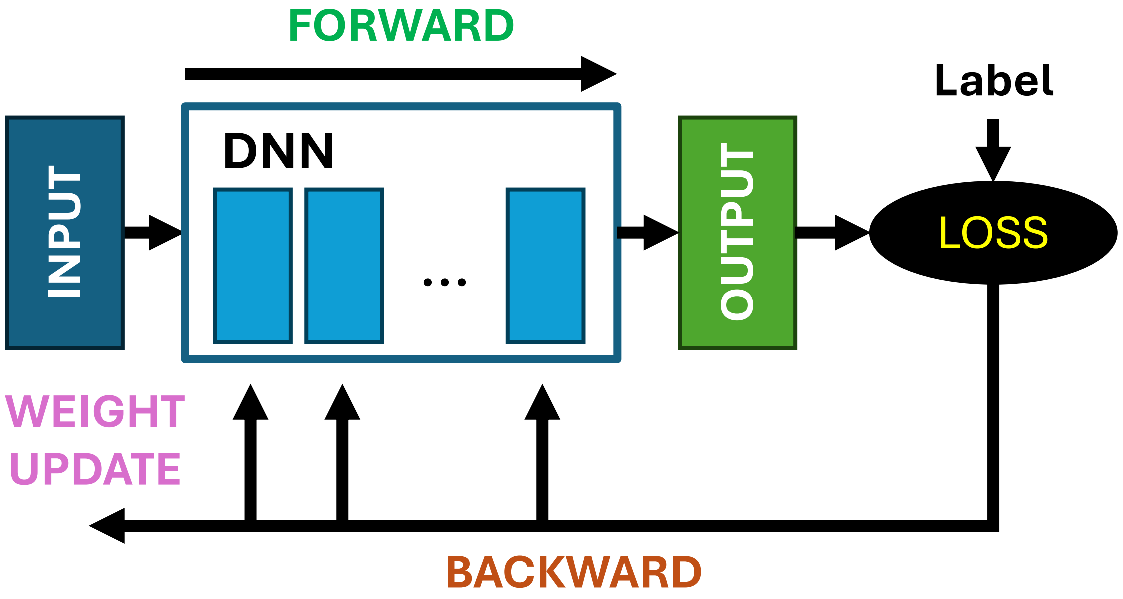
PULP-TrainLib

github.com/pulp-platform/pulp-trainlib

The first **latency-optimized open-source training library for RISC-V** multi-core **MCUs**

FUNCTION SIGNATURE

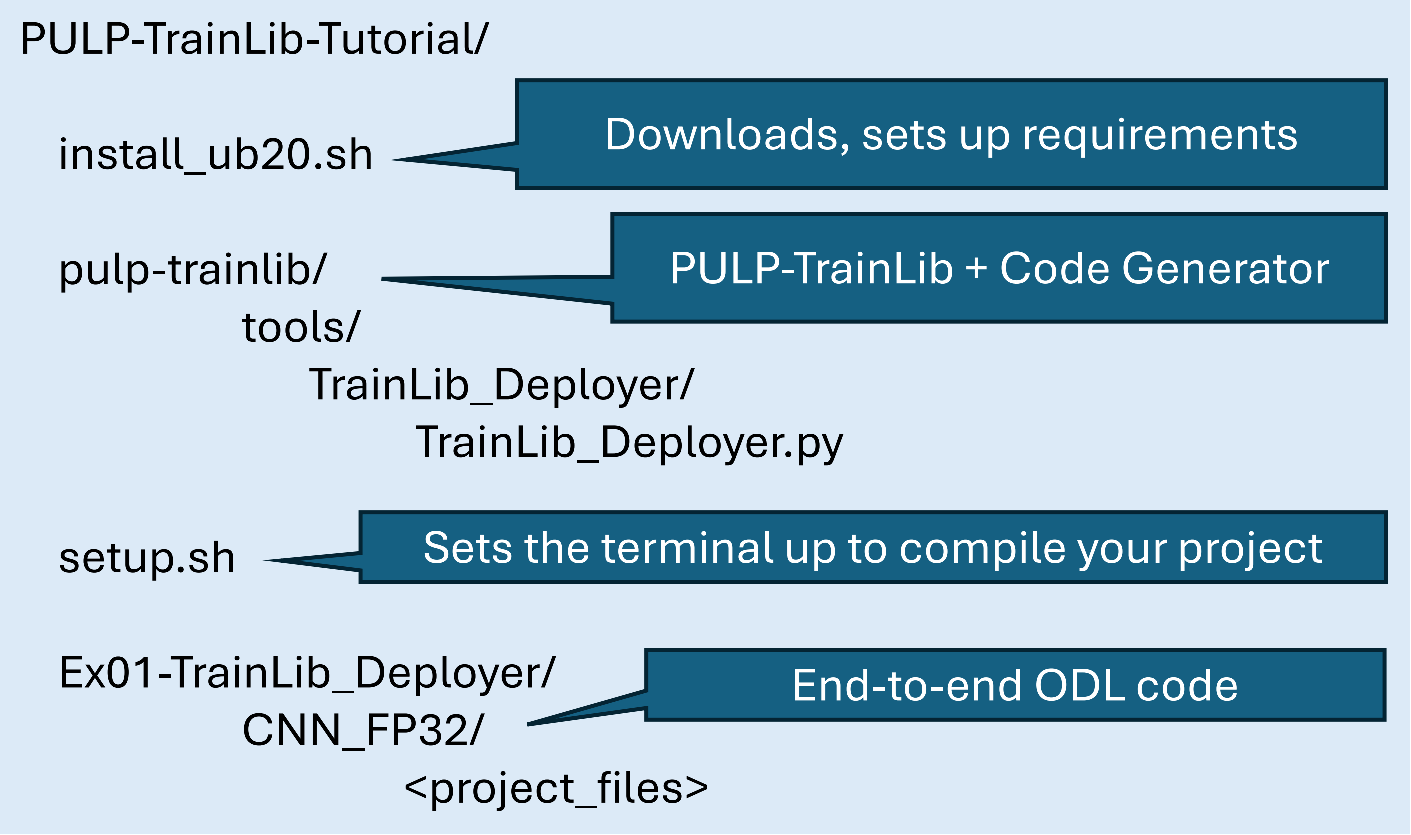
pulp_<layer/function>_<data_type>_<step>_<parallel?>();



```
pulp_linear_fp32_fw_cl(&lin_args);

pulp_CrossEntropyLoss(&loss_args);

pulp_linear_fp32_bw_param_grads_cl(&lin_args);
pulp_linear_fp32_bw_input_grads_cl(&lin_args);

pulp_gradient_descent_fp32(&wgts);
```

Repository Overview

github.com/dnadalini/PULP-TrainLib-Tutorial



```
PULP-TrainLib-Tutorial/
    install_ub20.sh
    pulp-trainlib/
            tools/
                TrainLib_Deployer/
                    TrainLib_Deployer.py
    setup.sh
    Ex01-TrainLib_Deployer/
            CNN_FP32/
                    <project_files>
```

install_ub20.sh — Downloads, sets up requirements

pulp-trainlib/ — PULP-TrainLib + Code Generator

setup.sh — Sets the terminal up to compile your project

Ex01-TrainLib_Deployer/CNN_FP32/ — End-to-end ODL code

Download requirements & build PULP-SDK

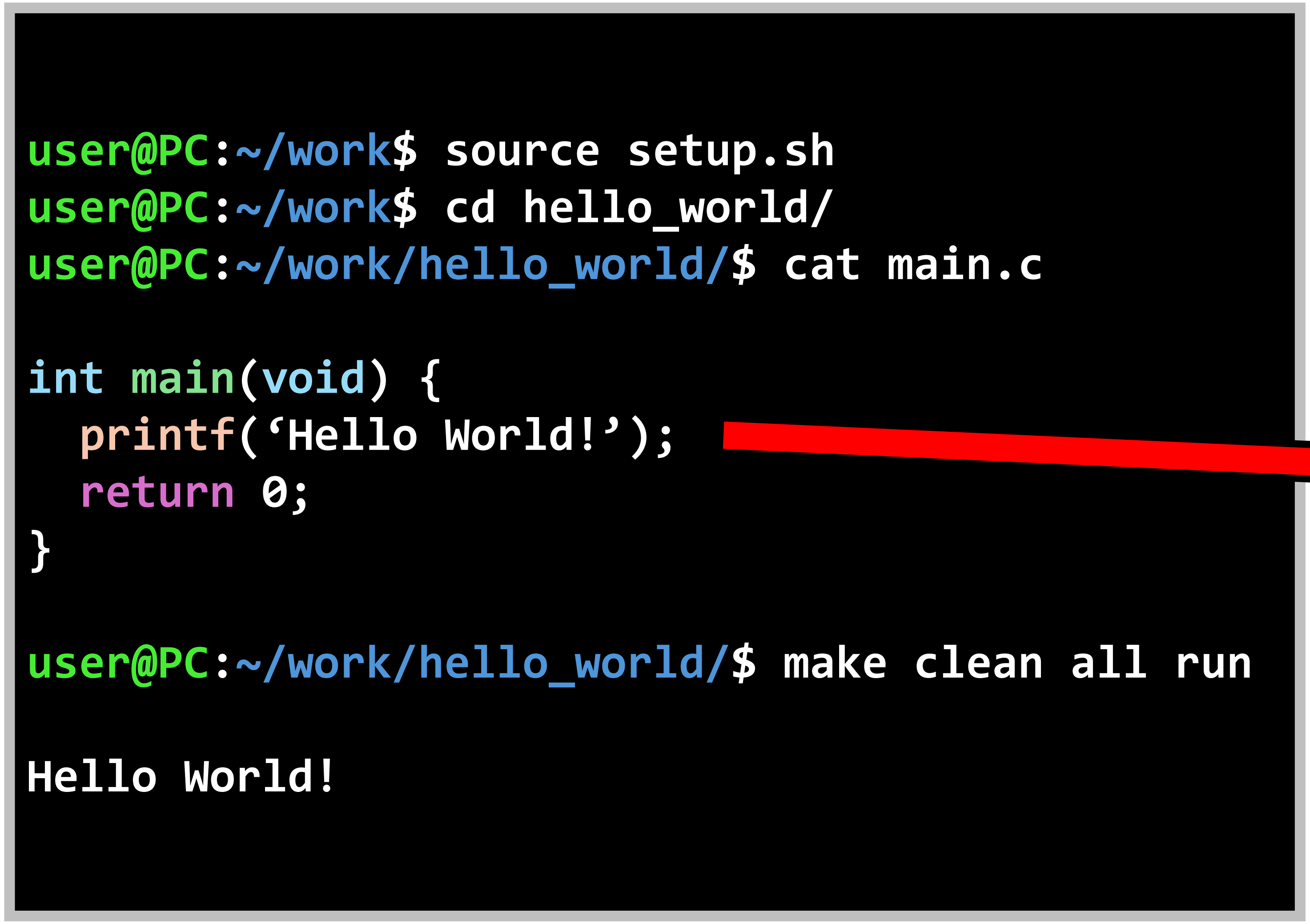
```
source install_ub20.sh
```

IN EACH NEW TERMINAL RUN

```
source setup.sh
```

You can now compile you project!!

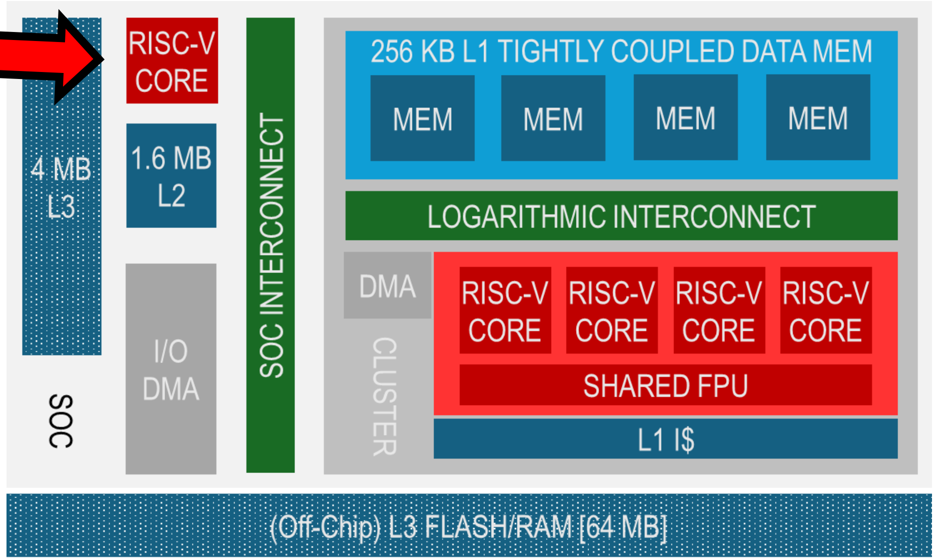
GVSoC simulator and performance metrics



```
user@PC:~/work$ source setup.sh
user@PC:~/work$ cd hello_world/
user@PC:~/work/hello_world/$ cat main.c

int main(void) {
  printf(‘Hello World!’);
  return 0;
}

user@PC:~/work/hello_world/$ make clean all run

Hello World!
```

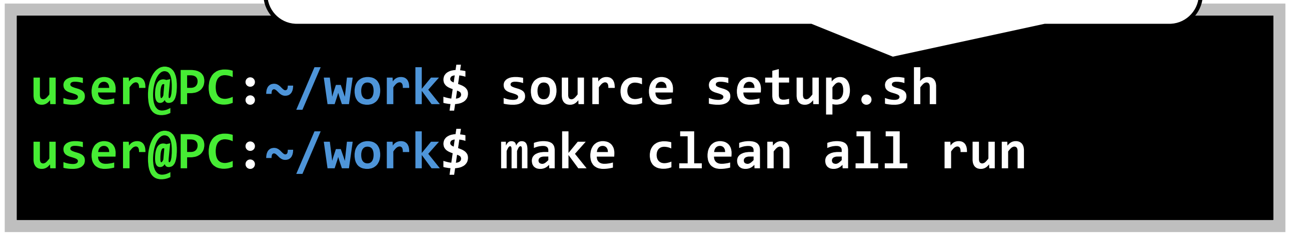
Configurable Behavioural Simulator for the PULP Plaftorm

Included in PULP-SDK!

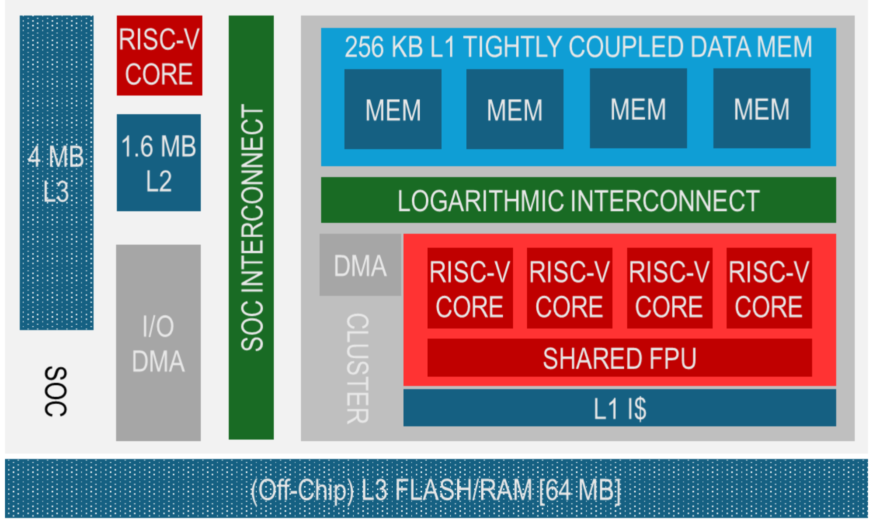


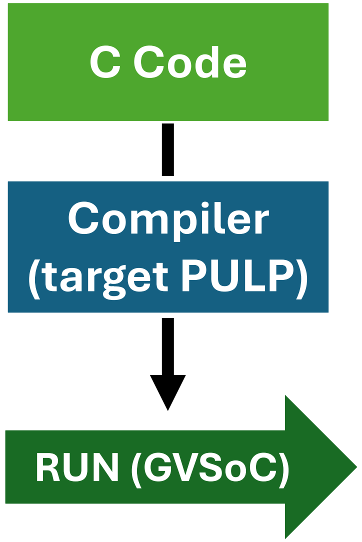
GVSoC simulator and performance metrics

Sets up PULP-SDK & compiler toolchain



```
user@PC:~/work$ source setup.sh
user@PC:~/work$ make clean all run
```

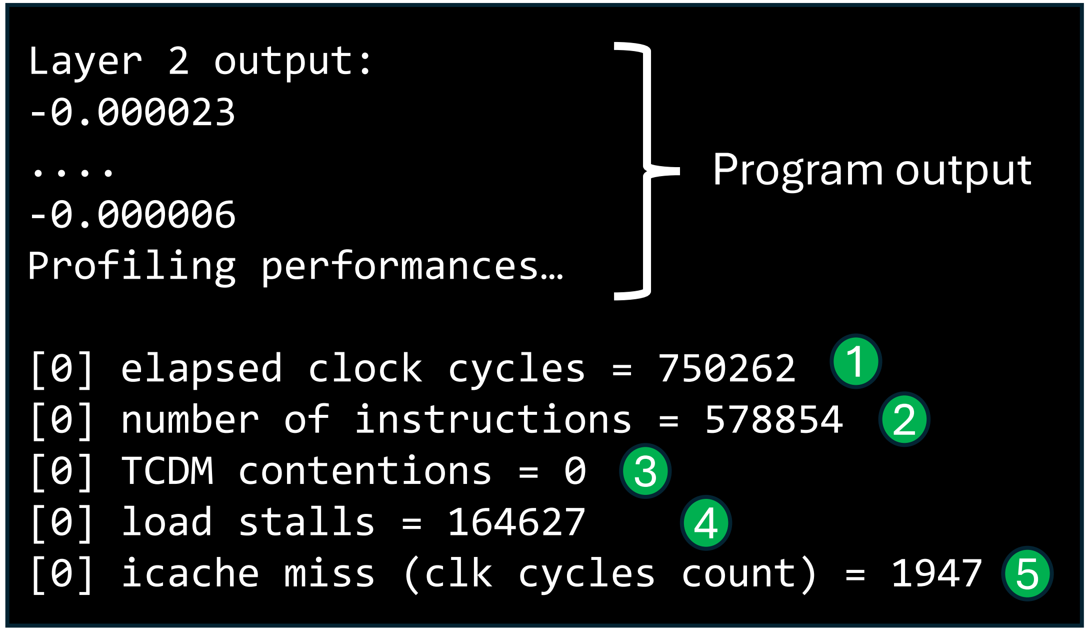




C Code

Compiler (target PULP)

RUN (GVSoC)



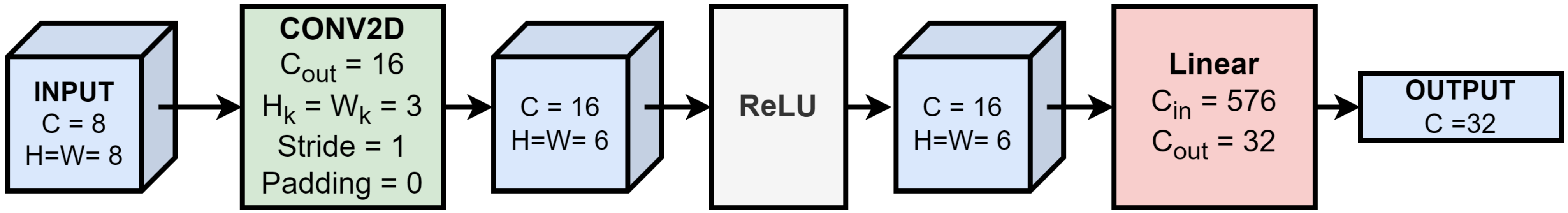
```
Layer 2 output:
-0.000023
....
-0.000006
Profiling performances…

[0] elapsed clock cycles = 750262 ①
[0] number of instructions = 578854 ②
[0] TCDM contentions = 0 ③
[0] load stalls = 164627 ④
[0] icache miss (clk cycles count) = 1947 ⑤
```

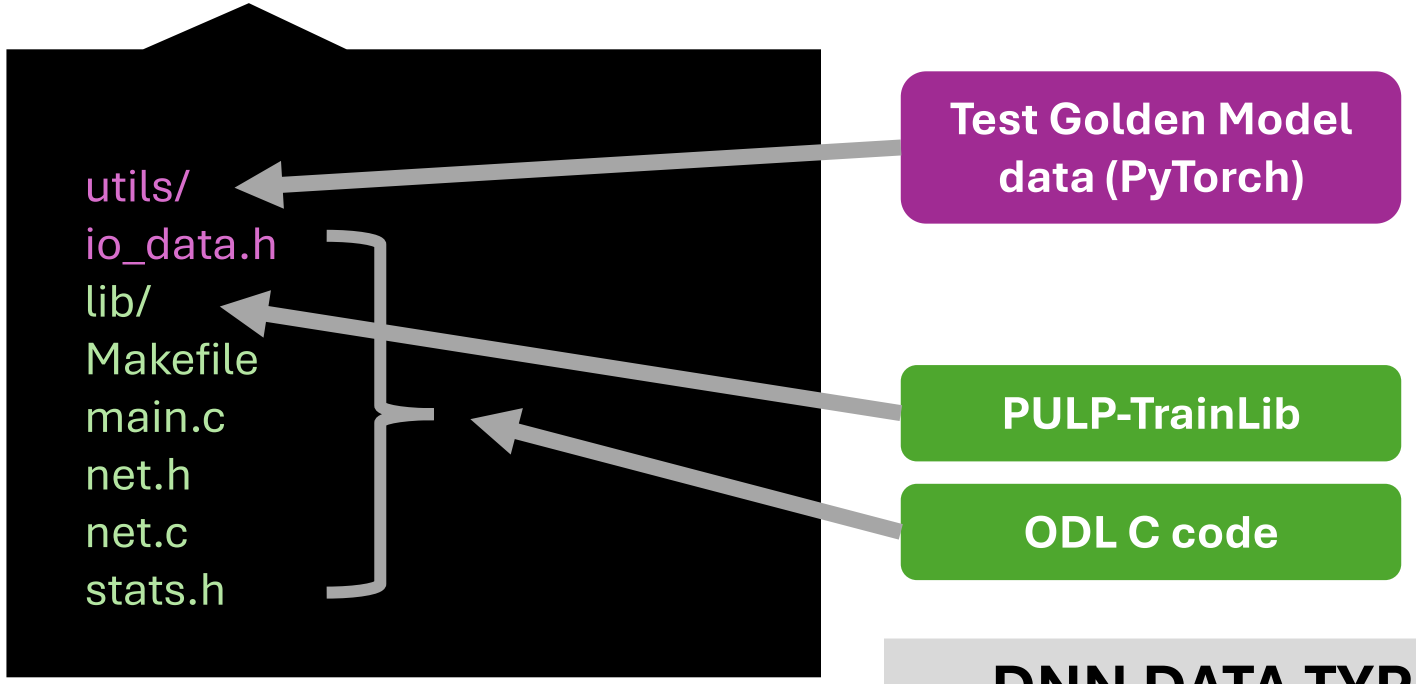
Program output

PROFILING INFORMATION
Performance Counters (clock cycles)

1. **Latency of the program**
2. **Number of instructions**
3. **Memory contentions in L1**
4. **Stalls while loading data from L1**
5. **Instruction cache misses**

Training a model with PULP-TrainLib




utils/
io_data.h
lib/
Makefile
main.c
net.h
net.c
stats.h

Test Golden Model data (PyTorch)

PULP-TrainLib

ODL C code

Set up environment variables

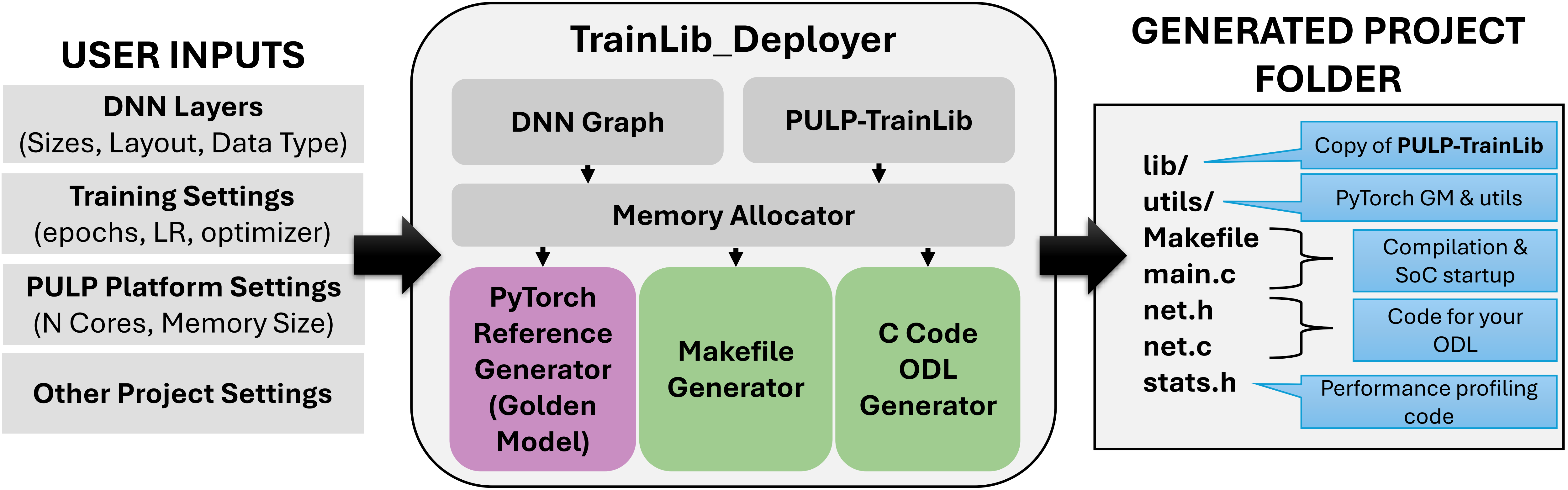
```
source setup.sh
```

Compile and run your ODL project

```
cd Ex01-TrainLib_Deployer/CNN_FP32/
make clean get_golden all run
```

DNN DATA TYPE = FP32, MEMORY = 207 kB

TrainLib-Deployer: ODL code generator



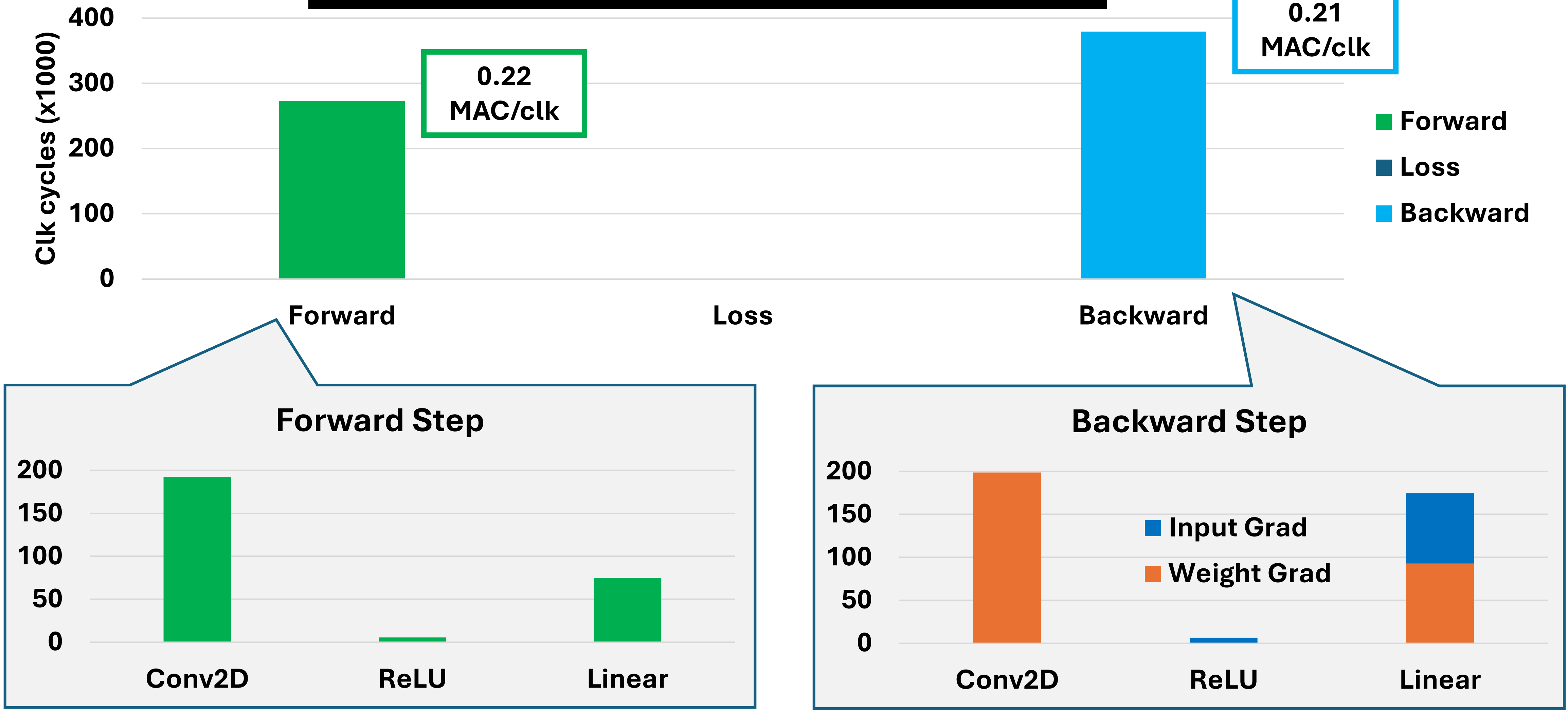
Generate project with TrainLib_Deployer

```
cd ../pulp-trainlib/tools/TrainLib_Deployer
python TrainLib_Deployer.py
```

WHAT ABOUT THE CODE?

ODL: latency of each step (FP32) – single core

```
cd Ex01-TrainLib_Deployer/CNN_FP32/
make clean get_golden all run NUM_CORES=1
```



0.22 MAC/clk

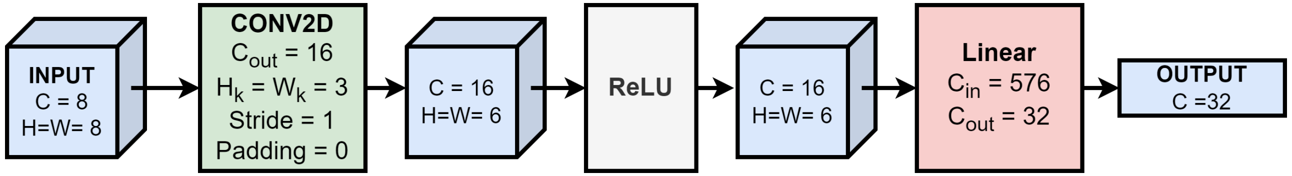
0.21 MAC/clk

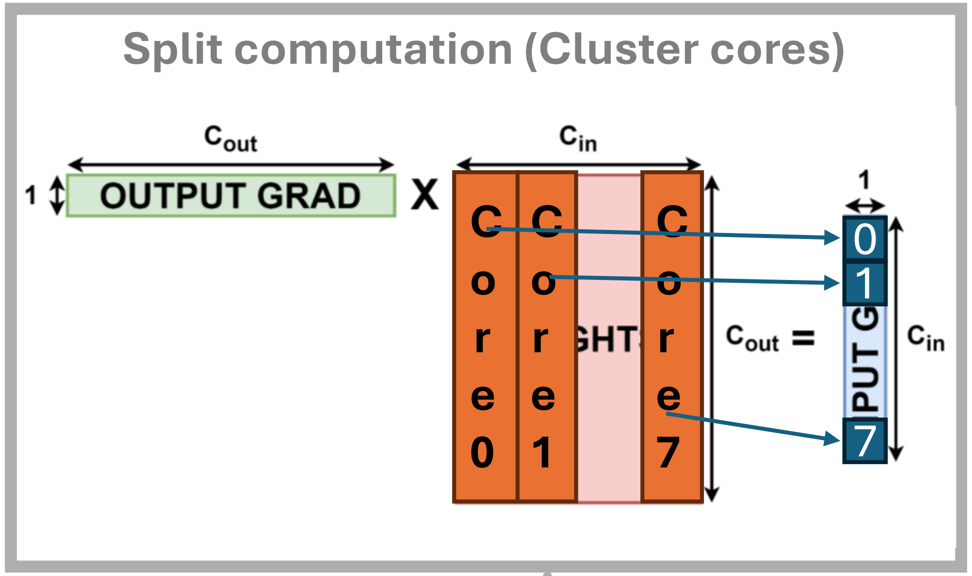
Forward Step

Backward Step

Optimization 1: Parallelization

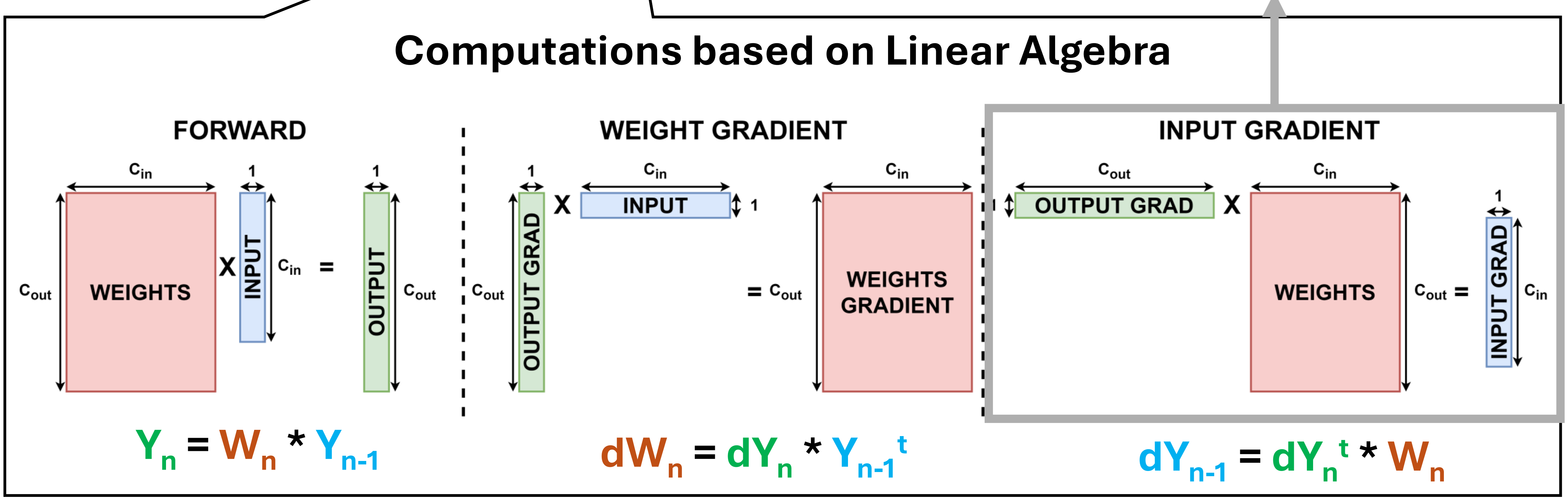
Parallelizing ODL code





Split computation (Cluster cores)

Computations based on Linear Algebra



$$Y_n = W_n * Y_{n-1}$$

$$dW_n = dY_n * Y_{n-1}^t$$

$$dY_{n-1} = dY_n^t * W_n$$

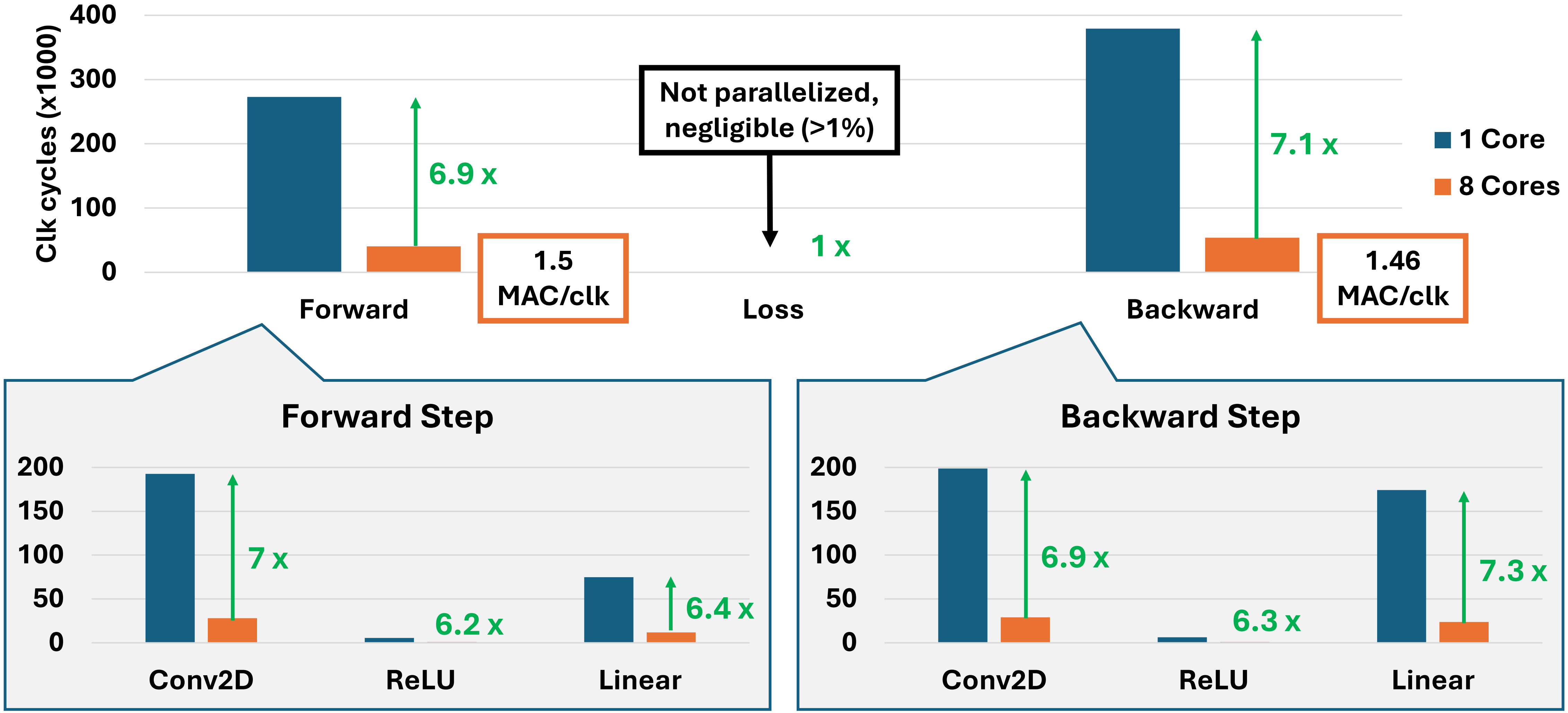
WHAT ABOUT THE CODE?

Parallelizing ODL code (cont’d)

IDEAL: 8x speedup

REAL: TCDM contentions + parallelization overhead

```
cd Ex01-TrainLib_Deployer/CNN_FP32/
make clean get_golden all run NUM_CORES=8
```



Not parallelized, negligible (>1%)

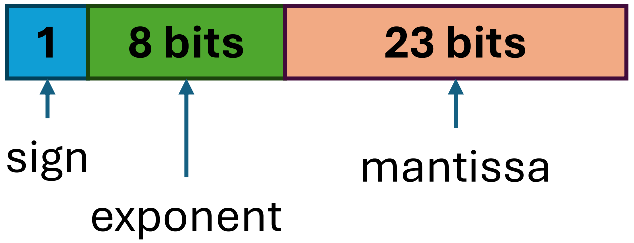
1.5 MAC/clk

1.46 MAC/clk

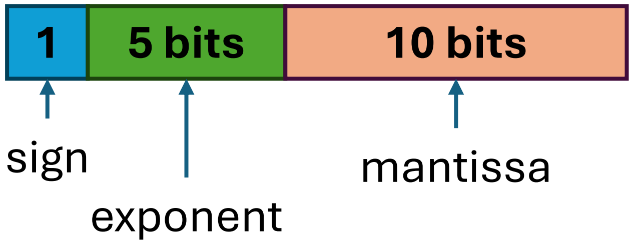
Optimization 2:
SIMD Vectorization (FP16)

FP16 Optimization

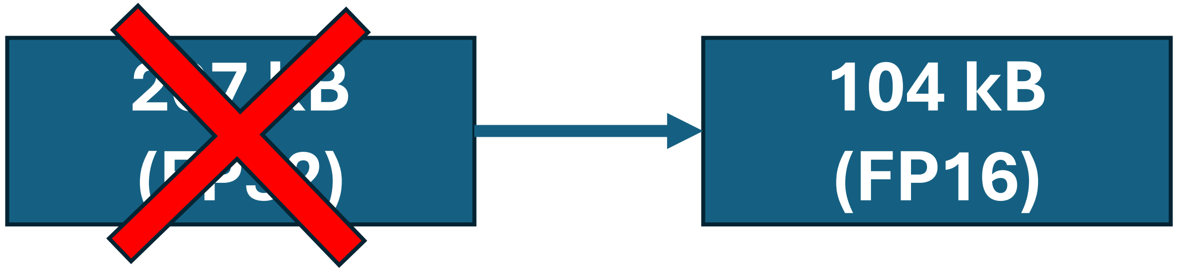
Floating-Point 32 (FP32)



Floating-Point 16 (FP16)*

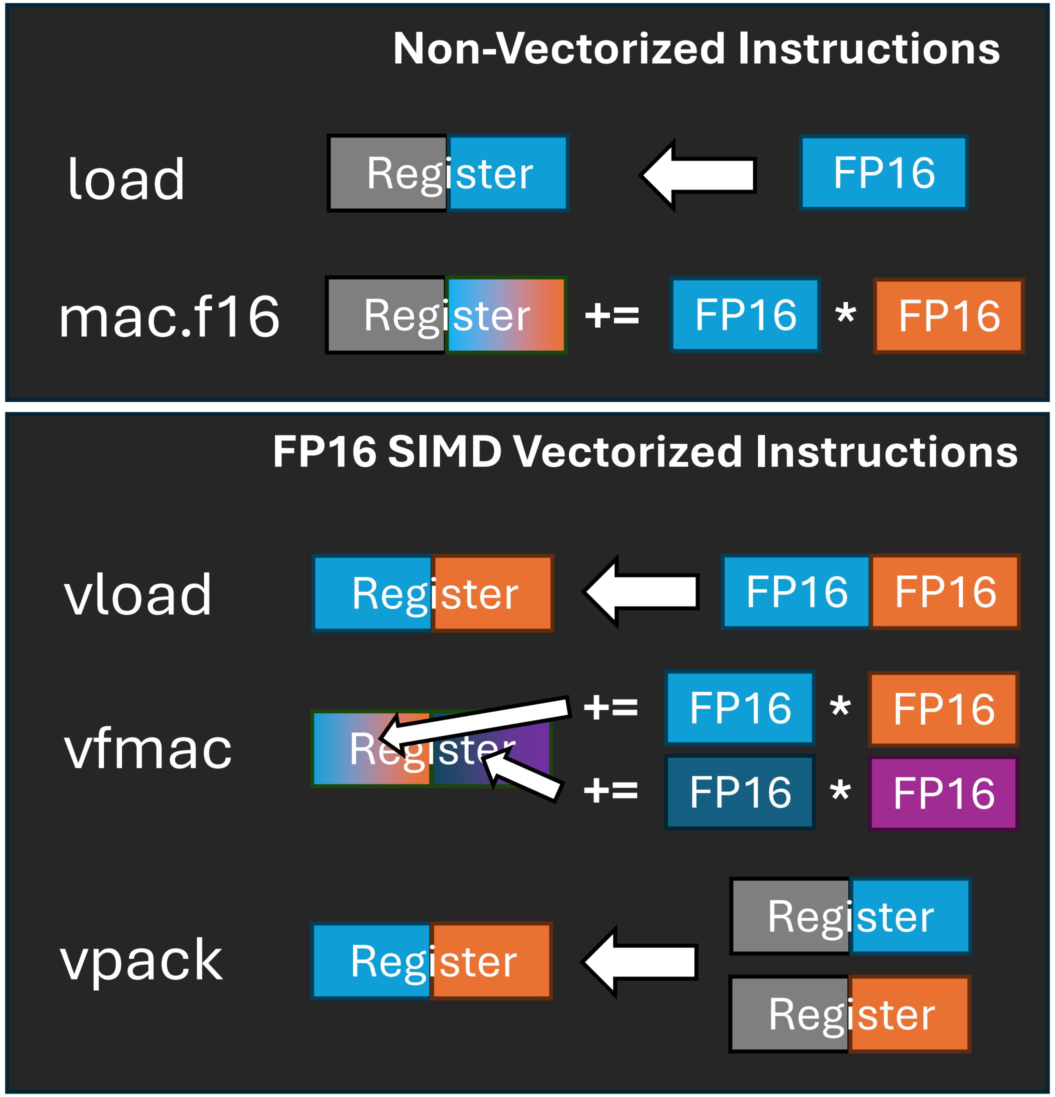


Memory Occupation (DNN):



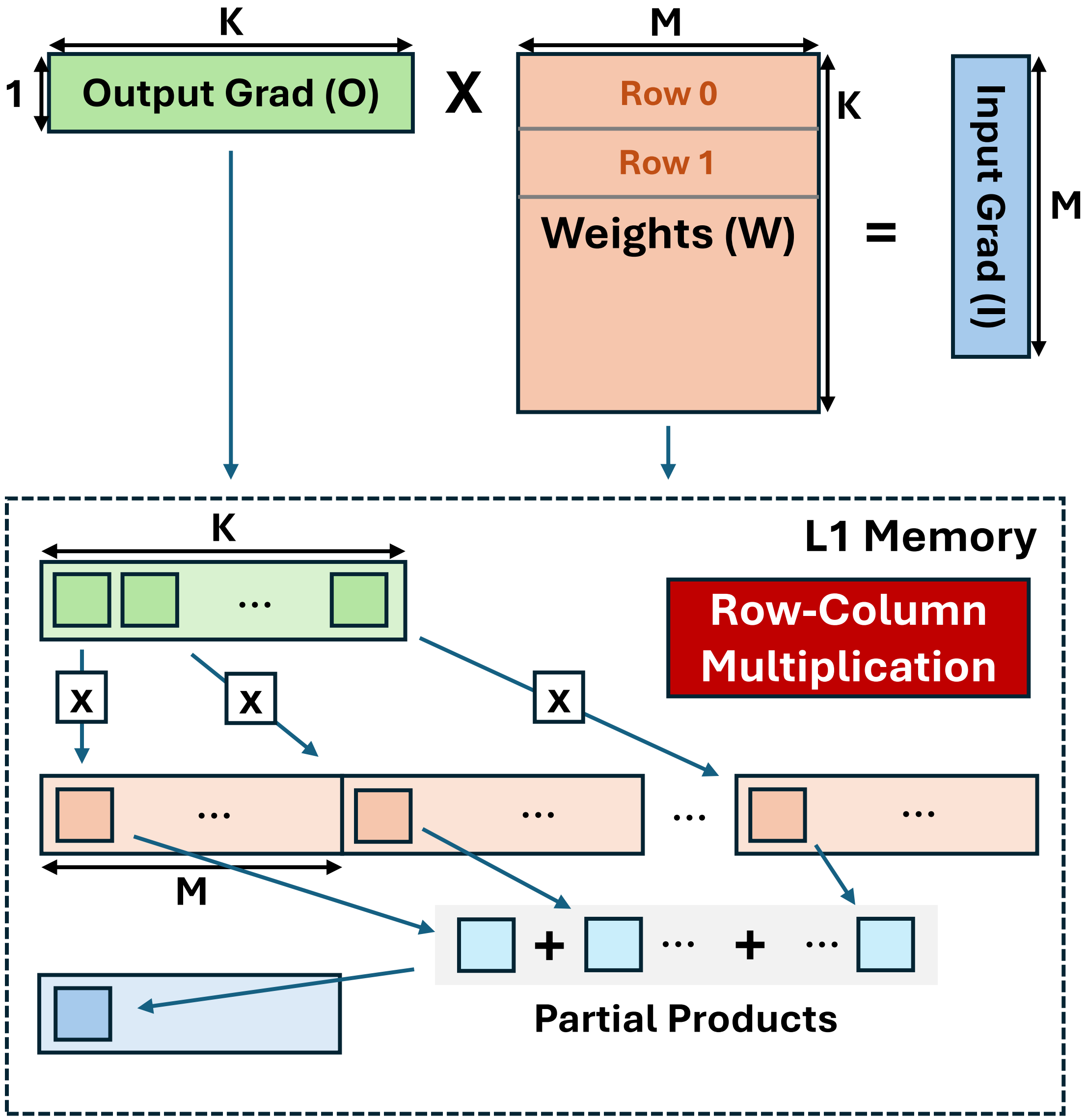
What about the latency?

Pseudo-Assembly Instructions



*IEEE or BFLOAT (user-selectable, platform support for both)

Linear Input Grad: Naive Vector-Matrix (FP16)



No SIMD: same performance as FP32

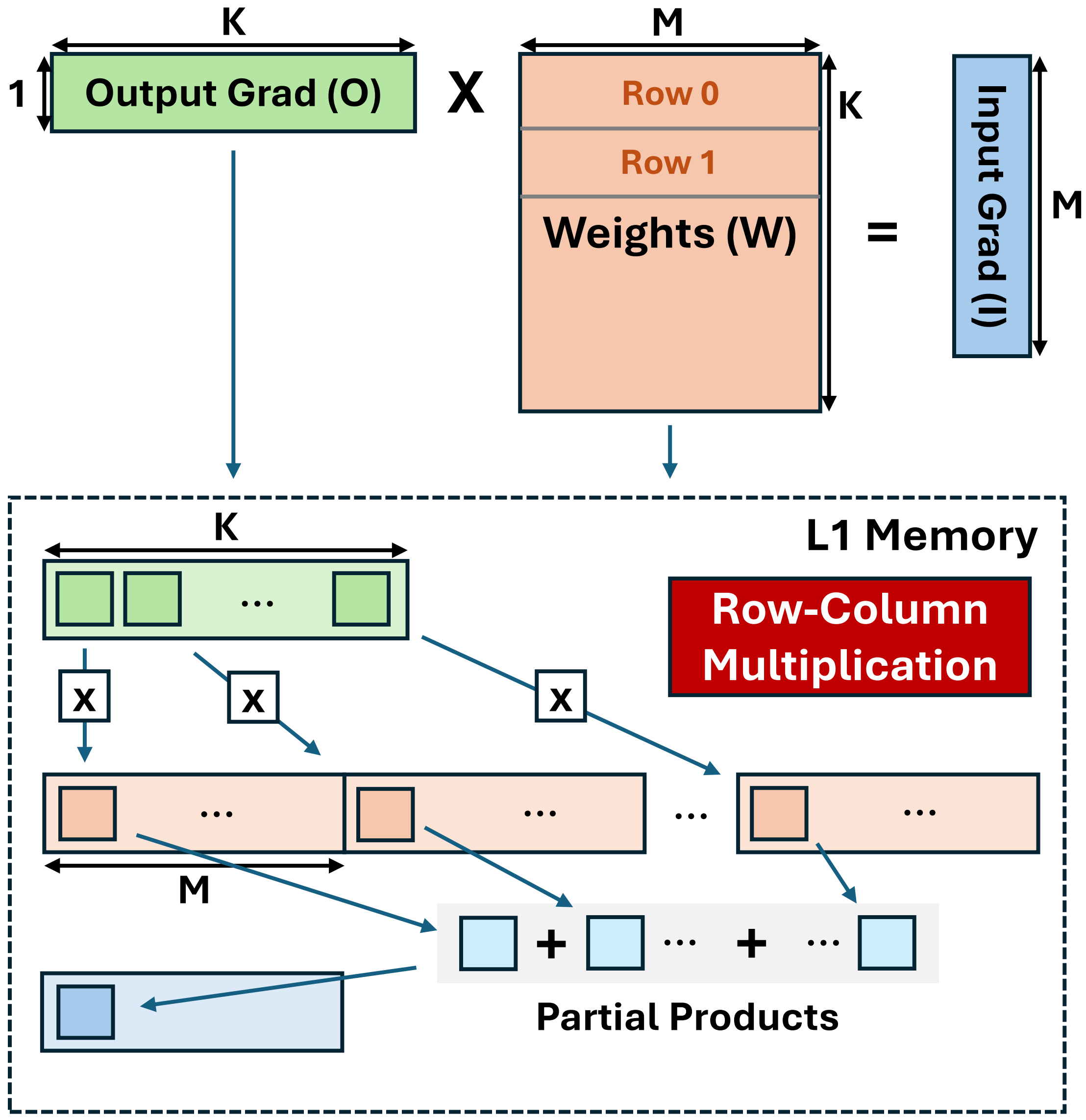
PSEUDO-ASSEMBLY (naive VM)

Loop K times

```
load      ra, O[k]
load      rb, W[k*M+j]

mac.f16   rd1, ra, rb

add       rd, rd1
store     I[j], rd
```

3 instructions / MAC

FP16: maximizing SIMD performance



SIMD Units: we want vectorized instructions

PSEUDO-ASSEMBLY (naive SIMD)

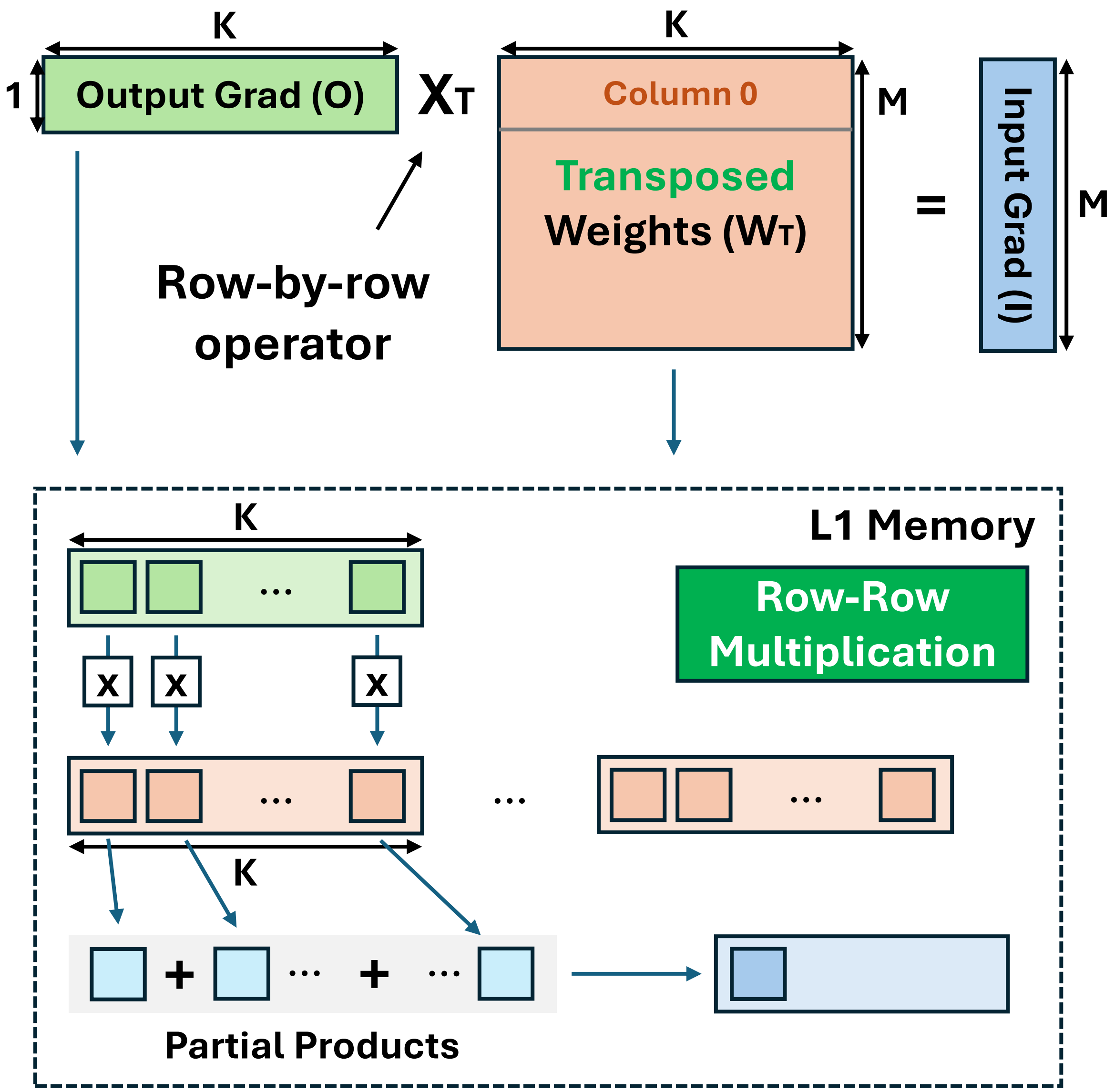
Loop K/2 times

```
vload     ra, O[k], O[k+1]
load      rs2, W[k*M+j]
load      rs3, W[(k+1)*M+j]

vpack     rb, rs2, rs3
vfmac     rd1, ra, rb

add       rd, rd1, rd2
store     I[j], rd
```

Vectorization overhead!!

5 instructions / 2 MAC

FP16: maximizing SIMD performance (cont'd)



SIMD Units: we want vectorized instructions

PSEUDO-ASSEMBLY (full SIMD)

Loop K/2 times

```
vload     ra, O[k], O[k+1]
vload     rb, WT[k+j*M], WT[(k+1)+j*M]

vfmac     rdv, ra, rb

add       rd, rdv(0), rdv(1)
store     I[j], rd
```

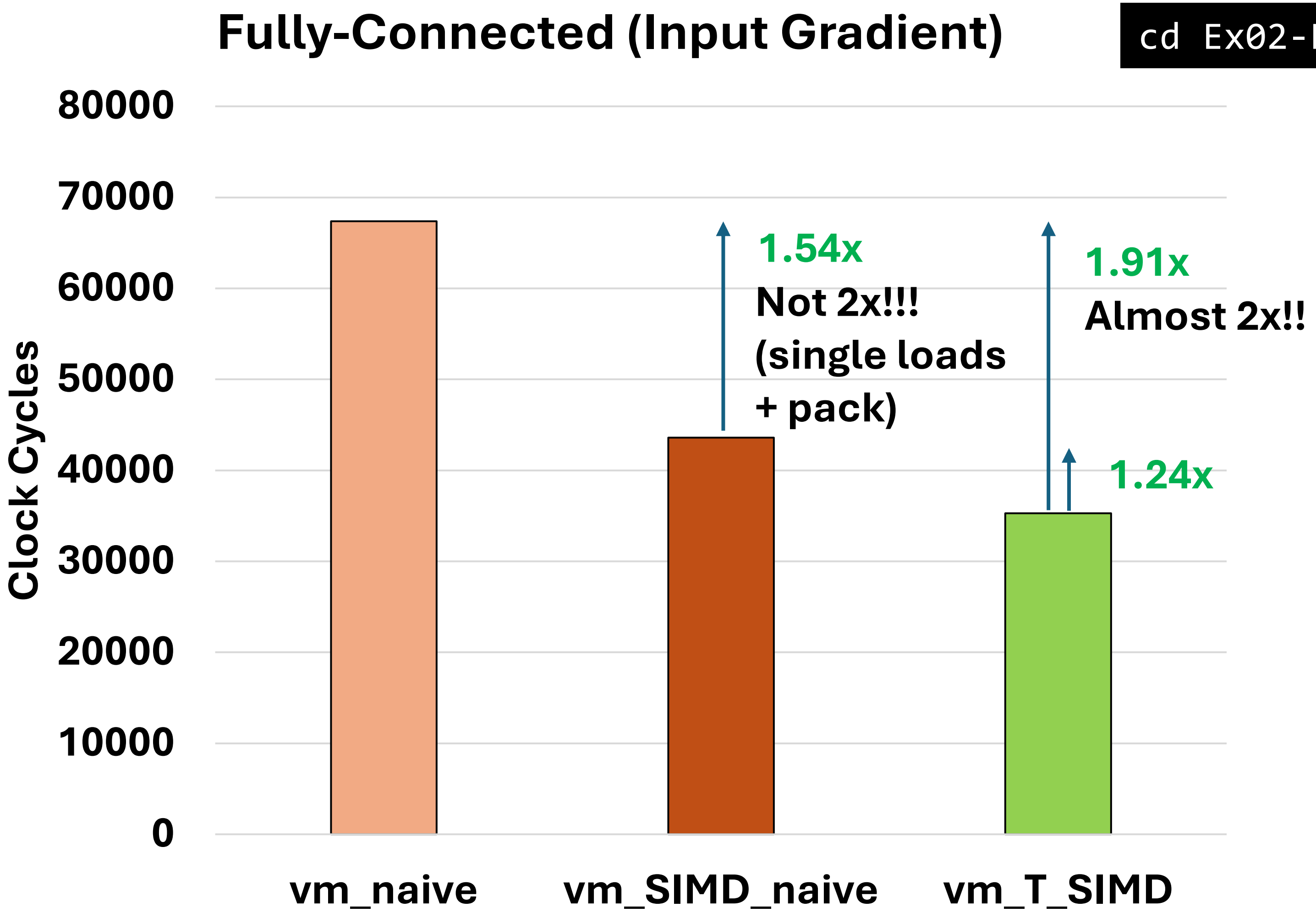
No overhead!

3 instructions / 2 MAC

WHAT ABOUT THE CODE?

FP16: maximizing SIMD performance (cont’d)

```
cd Ex02-Fully-Connected-FP16/test_linear_fp16/
```



Fully-Connected (Input Gradient)

1.54x
Not 2x!!!
(single loads
+ pack)

1.91x
Almost 2x!!

1.24x

Vector-Matrix (no SIMD)

```
make clean get_golden all run
MATMUL_TYPE=0 TRANSPOSE_WEIGHTS=0
```

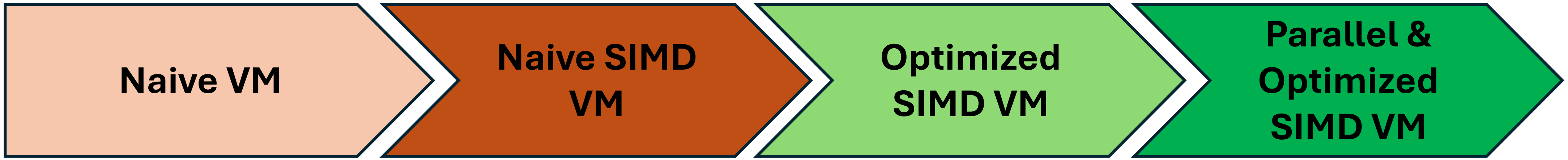
Naive SIMD

```
make clean get_golden all run
MATMUL_TYPE=1 TRANSPOSE_WEIGHTS=0
```

SIMD with transposed Weights

```
make clean get_golden all run
MATMUL_TYPE=2 TRANSPOSE_WEIGHTS=1
```

Stacking Optimizations (Linear Input Grad)



Naive VM

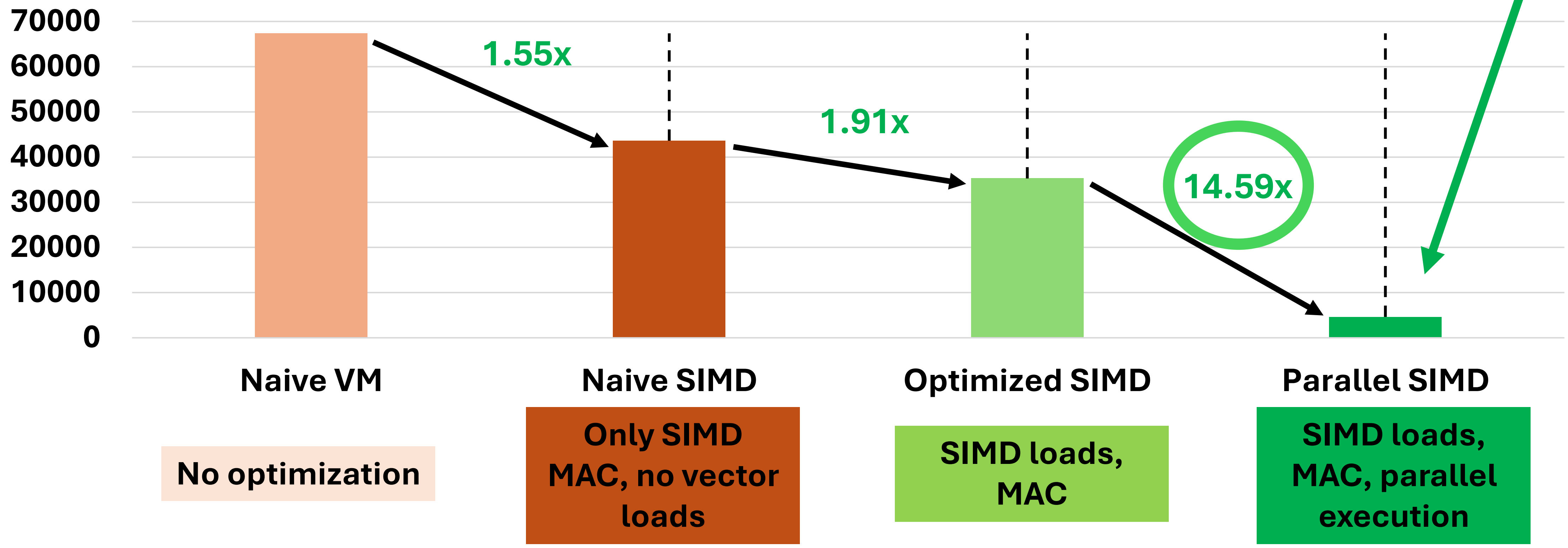
Naive SIMD VM

Optimized SIMD VM

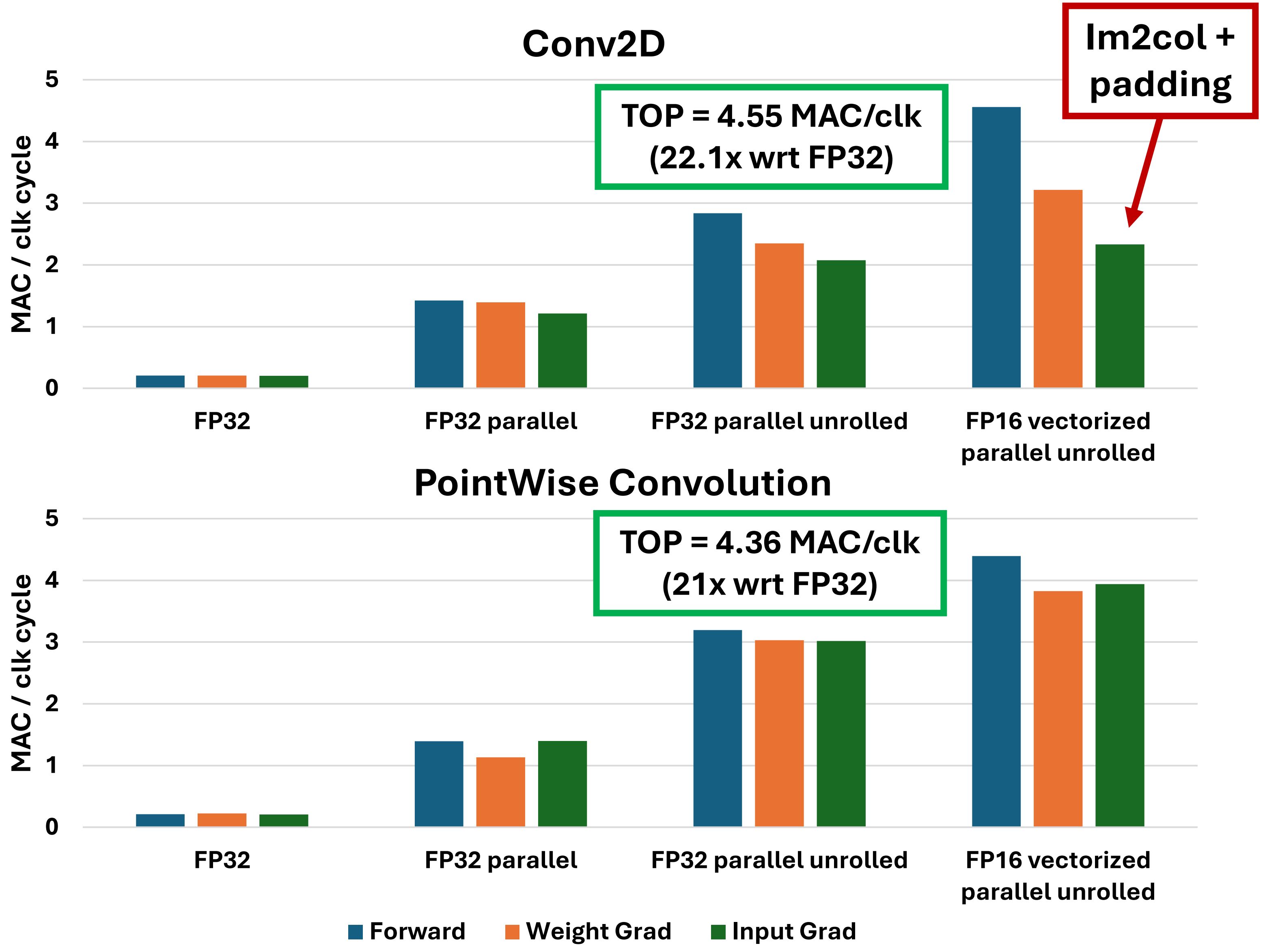
Parallel & Optimized SIMD VM

```
make clean get_golden all run MATMUL_TYPE=2
TRANSPOSE_WEIGHTS=1 NUM_CORES=8
```

Latency (Clock cycles, FP16)



PULP-TrainLib: Further Optimizations



TOP = 4.55 MAC/clk (22.1x wrt FP32)

Im2col + padding

TOP = 4.36 MAC/clk (21x wrt FP32)

OPTIMIZATIONS:

- FP32 Parallelization
- FP32 Unrolling + Parallelization
- FP16 Vectorization + Unrolling + Parallelization

SUPPORTED FEATURES:

- **Layers**: Conv2D, PointWise, DepthWise, Linear
- **Activations**: ReLU, SoftMax
- **Losses**: MSE, CrossEntropy
- **Optimizers**: SGD
- **Pooling**: MaxPool, AvgPool

Comparison with the State-of-the-Art

ODL Library / Method	Target Task	Target Device	Key Optimization	Retrainable Layers	Sparse Update	Data Type	Code Generation	Peak Training Performance (MAC/clk)
PULP-TrainLib [1, 2]	General Purpose	Multicore RISC-V MCUs	Parallelism, SIMD, unrolling	All (Convs, Linear)	No	FP32, FP16	Manual DNN definition, auto memory management	6.62[1]
TTE [3]	Image Classification / General Purpose	STM32	Quantized Sparse Update	All (Convs, Linear)	Layers, Weight Channels, Biases	INT8, FP32	Graph analysis with compile-time autodiff	~ 0.1[2]
AIfES [5]	General Purpose	General Purpose	Unrolling (MatMul)	All (Convs, Linear)	No	FP32, INT32, INT8	Graph Analysis (Pytorch, Keras)	0.15[3]

WE ARE OPEN FOR CONTRIBUTION!

[1] Profiled on hardware (Greenwaves GAP9)
[2] Extrapolation from data published in [3]
[3] Profiled on hardware (STM32L476RG)

References

[1] Nadalini, D., Rusci, M., Tagliavini, G., Ravaglia, L., Benini, L. and Conti, F., 2022, July. **PULP-TrainLib: Enabling on-device training for RISC-V multi-core MCUs through performance-driven Autotuning**. In *International Conference on Embedded Computer Systems* (pp. 200-216). Cham: Springer International Publishing.

[2] Nadalini, D., Rusci, M., Benini, L. and Conti, F., 2023. **Reduced Precision Floating-Point Optimization for Deep Neural Network On-Device Learning on MicroControllers**. *arXiv preprint arXiv:2305.19167*.

[3] Lin, J., Zhu, L., Chen, W.M., Wang, W.C., Gan, C. and Han, S., 2022. **On-device training under 256kb memory**. *Advances in Neural Information Processing Systems*, *35*, pp.22941-22954.

[4] Wulfert, L., Kühnel, J., Krupp, L., Viga, J., Wiede, C., Gembaczka, P. and Grabmaier, A., 2024. **AIfES: A Next-Generation Edge AI Framework**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Thank you for your attention

Davide Nadalini
d.nadalini@unibo.it