An Open Flow for DNNs on Ultra-Low-Power RISC-V Cores

Dr. Francesco Conti
Assistant Professor
University of Bologna

#RISCVSUMMIT
Real-World DNNs at Extreme-Edge?

- Improve latency
- Preserve privacy
- Enhance energy efficiency
Real-World DNNs at Extreme-Edge?

How to bridge the algorithm <-> device divide?

specification & dataset selection

training

optimized HW & architecture
Real-World DNNs at Extreme-Edge?

1. Exploit hardware’s strengths

- specification & dataset selection
- training
- optimized DNN primitives
- optimized HW & architecture
Real-World DNNs at Extreme-Edge?

1. specification & dataset selection
2. training
3. graph optimization
4. memory-aware deployment
5. optimized DNN primitives
6. optimized HW & architecture

2. Keep compute units fed with data from on-chip memories; minimize off-chip access
Real-World DNNs at Extreme-Edge?

3. Produce DNNs that are tuned to embedded hardware (e.g., int8)
Real-World DNNs at Extreme-Edge?

- Specification & dataset selection
- Training
- Quantization & pruning
- Graph optimization
- Memory-aware deployment
- Optimized DNN primitives
- Optimized HW & architecture

Open-Source flow for DNN Deployment on PULP devices

- NEMO: NEural Minimization for pyTorch
- DORY: Deployment Oriented to memoRY
- PULP-NN: Parallel ULP Neural Network library
Real-World DNNs at Extreme-Edge?

THIS TALK

- Specification & dataset selection
- Training
- Quantization & pruning
- Graph optimization
- Memory-aware deployment
- Optimized DNN primitives
- Optimized HW & architecture

NEMO
NEural Minimization for pytOrch

DORY
Deployment Oriented to memoRY

PULP-NN
Parallel ULP
Neural Network library
PULP-NN: optimized back-end

- specification & dataset selection
- training
- quantization & pruning
- graph optimization
- memory-aware deployment
- optimized DNN primitives
- optimized HW & architecture

PULP-NN
Parallel ULP
Neural Network library
GreenWaves GAP8 is a RISC-V ultra-low power processor based on the open-source PULP (Parallel Ultra-Low-Power) paradigm.

- A fast microcontroller with autonomous I/O capability, support for off-chip memory (HyperRAM DRAM / Flash)
- RV32IMC in-order, 4 pipeline stage core (RI5CY, now OpenHW Group CV32E40P)
GreenWaves GAP8 is a RISC-V ultra-low power processor based on the open-source PULP (Parallel Ultra-Low-Power) paradigm:

- a programmable cluster of RI5CY cores sharing memory at L1

- high-bandwidth, low-latency L1 scratchpad + DMA for manual memory management

- xPULPv2 ISA extension for enhanced DSP capabilities: hardware loops, address post-increment, SIMD scalar product
GreenWaves GAP8 Architecture

**Off-Chip**
- **L3 Memory**
  - 8 MB RAM
  - 64 MB Flash
  - 250 MB/s DDR

**I/O DOMAIN**
- PMU
- DC/DC
- RTC
- HYPER
- UART
- SPI
- I2S
- I2C
- GPIO
- JTAG
- DBG
- Instr Cache
- I/O RISC-V
- I/O L1
- ROM
- CLK

**CLUSTER DOMAIN**
- **Shared Multi-Bank L1 Memory - 64 kB**
  - 16 GB/s @ 250 MHz
- Logarithmic Interconnect
- Shared Instruction Cache

**L2 Memory**
- 512 kB
- 4 GB/s @ 250 MHz
PULP-NN: optimized computational back-end

Target int8 execution of CONV, FC, ... primitives
1) maximize data reuse in register file 2) improve kernel regularity 3) exploit parallelism

```
lp.setup
  p.lw  w0, 4(W0!)
p.lw  w1, 4(W1!)
p.lw  w2, 4(W2!)
p.lw  w3, 4(W3!)
p.lw  x1, 4(X0!)
p.lw  x2, 4(X1!)
pv.sdotsp.b  acc1, w0, x0
pv.sdotsp.b  acc2, w0, x1
pv.sdotsp.b  acc3, w1, x0
pv.sdotsp.b  acc4, w1, x1
pv.sdotsp.b  acc5, w2, x0
pv.sdotsp.b  acc6, w2, x1
pv.sdotsp.b  acc7, w3, x0
pv.sdotsp.b  acc8, w3, x1
end
```

Load 16 weights (8-bit)
4 out chan, 4 in chan
address post-increment

PULP-NN: optimized computational back-end

Target int8 execution of CONV, FC, ... primitives
1) maximize data reuse in register file 2) improve kernel regularity 3) exploit parallelism

```
lp.setup
  p.lw  w0, 4(W0!)
p.lw  w1, 4(W1!)
p.lw  w2, 4(W2!)
p.lw  w3, 4(W3!)
p.lw  x1, 4(X0!)
p.lw  x2, 4(X1!)
```

Load 8 pixels
2 rows, 4 in chan
address post-increment

PULP-NN: optimized computational back-end

Target int8 execution of CONV, FC, ... primitives
1) maximize data reuse in register file 2) improve kernel regularity 3) exploit parallelism

```
lp.setup
  p.lw w0, 4(W0!)
p.lw w1, 4(W1!)
p.lw w2, 4(W2!)
p.lw w3, 4(W3!)
p.lw x1, 4(X0!)
p.lw x2, 4(X1!)
pv.sdotsp.b acc1, w0, x0
pv.sdotsp.b acc2, w0, x1
pv.sdotsp.b acc3, w1, x0
pv.sdotsp.b acc4, w1, x1
pv.sdotsp.b acc5, w2, x0
pv.sdotsp.b acc6, w2, x1
pv.sdotsp.b acc7, w3, x0
pv.sdotsp.b acc8, w3, x1
end
```


Compute 32 MAC over 8 accumulators dot-product instructions
PULP-NN: optimized computational back-end

Target \texttt{int8} execution of CONV, FC, ... primitives

1) maximize data reuse in register file  
2) improve kernel regularity  
3) exploit parallelism

```plaintext
lp.setup
  p.lw w0, 4(W0!)
p.lw w1, 4(W1!)
p.lw w2, 4(W2!)
p.lw w3, 4(W3!)
p.lw x1, 4(X0!)
p.lw x2, 4(X1!)
  pv.sdotsp.b acc1, w0, x0
  pv.sdotsp.b acc2, w0, x1
  pv.sdotsp.b acc3, w1, x0
  pv.sdotsp.b acc4, w1, x1
  pv.sdotsp.b acc5, w2, x0
  pv.sdotsp.b acc6, w2, x1
  pv.sdotsp.b acc7, w3, x0
  pv.sdotsp.b acc8, w3, x1
end
```

PULP-NN [Garofalo 19]  
https://arxiv.org/abs/1908.11263
PULP-NN: optimized computational back-end

Target int8 execution of CONV, FC, ... primitives
1) maximize data reuse in register file
2) improve kernel regularity
3) exploit parallelism

lp.setup
p lw w0, 4(W0!)
p lw w1, 4(W1!)
p lw w2, 4(W2!)
p lw w3, 4(W3!)
p lw x1, 4(X0!)
p lw x2, 4(X1!)
pv.sdotsp.b acc1, w0, x0
pv.sdotsp.b acc2, w0, x1
pv.sdotsp.b acc3, w1, x0
pv.sdotsp.b acc4, w1, x1
pv.sdotsp.b acc5, w2, x0
pv.sdotsp.b acc6, w2, x1
pv.sdotsp.b acc7, w3, x0
pv.sdotsp.b acc8, w3, x1
end

Parallelize over 8 cores

32-bit accum

PULP-NN: optimized computational back-end

Target int8 execution of CONV, FC, ... primitives

1) maximize data reuse in register file
2) improve kernel regularity
3) exploit parallelism

Peak Performance (8 cores)

**15.5 MAC/cycle**

DORY: Tiling & Code Generation

- specification & dataset selection
- training
- quantization & pruning
- graph optimization
- memory-aware deployment
- optimized DNN primitives
- optimized HW & architecture

DORY Deployment Oriented to memoRY
DORY: the tensor tiling problem

1.0-MobileNet-128
(59% top-1 accuracy on ImageNet)
DORY: the tensor tiling problem

L3 / L2 tiling
64 MB / 512 kB

small memory

big memory
DORY: the tensor tiling problem

**L3 / L2 tiling**
- 64 MB / 512 kB

**L2 / L1 tiling**
- 512 kB / 64 kB

small memory

big memory
DORY: Tiling as optimization problem

How to define the sizes of various tiles? \(\rightarrow\) abstracted as **Constraint Programming** problem

\[
\text{cost} = \max \text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}})
\]

\text{MEMORY } \Rightarrow \text{s.t. } \text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}}) < \text{SizeSmallMem}

**Basic idea:** maximize tile size while fitting the overall memory size constraint

---

Google ORTools

Integer DNN + tile sizes
DORY: Tiling as optimization problem

How to define the sizes of various tiles? → abstracted as Constraint Programming problem

\[\text{cost} = \max \text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}})\]

\[\text{MEMORY } \text{s.t. } \text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}}) < \text{SizeSmallMem}\]

\[\text{GEOMETRY } \text{s.t. } \{y_{\text{tile}}[c_{\text{out}}] = W_{\text{tile}}[c_{\text{out}}], \ldots\}\]

Size of tiles for in tensors / out tensors / weights tied by geometric constraints typical of a given layer
DORY: Tiling as optimization problem

How to define the sizes of various tiles? → abstracted as **Constraint Programming** problem

\[
\text{cost} = \max \text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}})
\]

**MEMORY** → s.t. \(\text{Size}(W_{\text{tile}}) + \text{Size}(x_{\text{tile}}) + \text{Size}(y_{\text{tile}}) < \text{SizeSmallMem}\)

**GEOMETRY** → s.t. \(\{y_{\text{tile}}[ch_{\text{out}}] = W_{\text{tile}}[ch_{\text{out}}], \ldots\}\)

**EFF. HEURISTICS** → \(\text{cost}' = \text{cost} + \{y_{\text{tile}}[ch_{\text{out}}] \text{ divisible by 4}, \ldots\}\)

*Performance is maximum for configurations that use PULP-NN primitives more efficiently (e.g., full parallelism)*

---

Integer DNN

Google

ORTools

Integer DNN + tile sizes
DORY: Tile data movement

Input feature map \( I \)

Output feature map \( O \)

Filters weights \( W \)

L1 buffer 1

x TILE 1

y TILE 1

W TILE 1

L1 buffer 2

x TILE 2

y TILE 2

W TILE 2

L2 memory

CONVOLUTIONAL PIPELINE

DMA ch. 0-1

DMA ch. 2

Cluster computation

In.

copy

\( t_0, t_1, t_2, t_3, \ldots, t_n \)
DORY: Tile data movement

**L2 memory**

- Input feature map $I$:
  - Height $h_M$
  - Width $w_M$
  - Height $i_M$
  - Width $w_M$

- Filters weights $W$:
  - Height $h_M$
  - Width $w_M$

- Output feature map $O$:
  - Height $o_M$
  - Width $w_M$

**L1 memory**

- L1 buffer 1:
  - X TILE 1
  - Y TILE 1
  - W TILE 1

- L1 buffer 2:
  - X TILE 2
  - Y TILE 2
  - W TILE 2

**CONVOLUTIONAL PIPELINE**

- DMA ch. 0-1
- DMA ch. 2
- Cluster computation

In.
copy

CONVOL. kernel

In.
y TILE 1

W TILE 1

x TILE 2

y TILE 2

W TILE 2

$\theta_0 \theta_1 \theta_2 \theta_3 \ldots \theta_n$
DORY: Tile data movement

CONVOLUTIONAL PIPELINE

illiseconds: 0-1 DMA ch. 2
Cluster computation

Input feature map $I$
Output feature map $O$
Filters weights $W$
L1 buffer 1
x TILE 1
y TILE 1
W TILE 1
L1 buffer 2
x TILE 2
y TILE 2
W TILE 2
L2 memory

$O_M$, $i_M$, $h_M$, $w_M$
$O_M$, $i_M$, $h_M$, $w_M$
$O_M$, $i_M$, $h_M$, $w_M$
$O_M$, $i_M$, $h_M$, $w_M$
DORY: Tile data movement

Input feature map $I$

Output feature map $O$

Filters weights $W$

$L1$ buffer 1
- $x$ TILE 1
- $y$ TILE 1
- $W$ TILE 1

$L1$ buffer 2
- $x$ TILE 2
- $y$ TILE 2
- $W$ TILE 2

$L2$ memory

CONVOLUTIONAL PIPELINE

DMA ch. 0-1
DMA ch. 2

Cluster computation

In. copy
Convol. kernel
Out. copy

DMA ch. 0-1
DMA ch. 2
DORY: Tile data movement

L2 memory

Input feature map $I$

Filters weights $W$

Output feature map $O$

L1 memory

L1 buffer 1
- $x$ TILE 1
- $y$ TILE 1
- $W$ TILE 1

L1 buffer 2
- $x$ TILE 2
- $y$ TILE 2
- $W$ TILE 2

CONVOLUTIONAL PIPELINE

$\begin{array}{cccc}
  t_0 & t_1 & t_2 & t_3 \\
  \text{In. copy} & \text{Convol. kernel} & \text{Out. copy} & \text{In. copy} \\
  \text{In. copy} & \text{Convol. kernel} & \text{Out. copy} & \text{In. copy} \\
  \text{In. copy} & \text{Convol. kernel} & \text{Out. copy} & \text{In. copy} \\
\end{array}$

Cluster computation

DMA ch. 0-1

DMA ch. 2
DORY: code generation example

Integer Network + tile sizes

Code Generation from templates

Network-level C code
- L3/L2 transfer boilerplate
- double buffering for weights
- calls to layer-level code

Layer-level C code
- L2/L1 transfer boilerplate
- calls to PULP-NN backend library

1.0-MobileNet-128
(59% top-1 accuracy on ImageNet)

100 MHz, 4 fps, 12.5 mJ/frame
260 MHz, 10 fps, 25 mJ/frame
An example of our flow in use!

https://www.youtube.com/watch?v=xBd2nAFNuWY

Credits: Nicky Zimmermann¹, Hanna Müller², Jerome Guzzi¹, Alessandro Giusti¹, Daniele Palossi¹,²

¹IDSIA Lugano, ²ETH Zürich
Where can I get it?

Our deployment flow is fully open-source – as is our full hardware platform.

**PULP-NN library:** [https://github.com/pulp-platform/pulp-nn](https://github.com/pulp-platform/pulp-nn)

**DORY deployment tool:** [https://github.com/pulp-platform/dory](https://github.com/pulp-platform/dory)

**NEMO network minimization tool:** [https://github.com/pulp-platform/nemo](https://github.com/pulp-platform/nemo)
Thanks for your attention!

Credits: Angelo Garofalo, Alessio Burrello, Nazareno Bruschi, Manuele Rusci, Giuseppe Tagliavini, Davide Rossi, Luca Benini & the PULP team