Extreme Edge AI on Open Hardware

ACCML@HIPEAC Bologna 20.01.2020

Luca Benini$^{1,2}$

$^1$Department of Electrical, Electronic and Information Engineering

$^2$Integrated Systems Laboratory
Cloud → Edge → Extreme Edge AI aka TinyML

Extreme edge AI challenge
AI capabilities in the power envelope of an MCU: 100mW peak (1mW avg)

#1 Customer Question on Amazon.com (out of 1,000+):
1. I don’t want any of my (private, personal) videos on any servers not in my control. Is this possible?

#2 Customer Question on Amazon.com (out of 1,000+):
2. How long does the battery charge last?

E. Gousev, Qcomm research

LOGISTICS $28BN
MANUFACTURING / INDUSTRIAL AUTOMATION $22BN
SMART CITIES/ BUILDING $12BN
RETAIL $8BN

AVERAGE CAGR 27.3%
70B$ in 5Y

Source: www.amazon.com/ask/questions/a21801539/5V3K3W4H81
AI Workloads from Cloud to Edge (Extreme?)

- **GOP+**
- **MB+**

High OP/B ratio
Massive Parallelism
MAC-dominated
Low precision OK
Model redundancy
Energy efficiency is THE Challenge

High performance MCUs

Low-Power MCUs

1pJ/OP=1TOPS/W

InceptionV4 @1fps in 10mW

Cool… But, HOW??
As VDD decreases, operating speed decreases.

However, efficiency increases → more work done per Joule.

Until leakage effects start to dominate.

Put more units in parallel to get performance up and keep them busy with a parallel workload.

Better to have N PEs running at lower voltage than one core at nominal voltage!

ML is massively parallel and scales well (P/S ↑ with NN size).
The workhorse: A simple RISC-V pipeline + ISA Extensions

3-cycle ALU-OP, 4-cyle MEM-OP → IPC loss: LD-use, Branch

ISA is extensible by construction (great!)

V1  Baseline RISC-V RV32IMC
    HW loops

V2  Post modified Load/Store Mac

V3  SIMD 2/4 + DotProduct + Shuffling
    Bit manipulation unit
    Lightweight fixed point (EML centric)

XPULP extensions: 25KG → 40KG (1.6x)
PULP-NN: Xpulp ISA exploitation

8-bit Convolution

- 9x less instructions than RV32IMC
- Pooling & ReLu:
  - HW loop
  - LD/ST with post-increment
  - 8-bit SIMD max, avg INSNS

```
addi    a0, a0, 1
addi    t1, t1, 1
addi    t3, t3, 1
addi    t4,t4,1
lbu     a7,-1(a0)
lbu     a6,-1(t4)
lbu     a5,-1(t3)
lbu     t5,-1(t1)
mul     s1,a7,a6
mul     a7,a7,a5
add     s0,s0,s1
mul     a6,a6,t5
add     t0,t0,a7
mul     a5,a5,t5
add     t2,t2,a6
add     t6,t6,a5
bne     s5,a0,1c000bc
lp.setup
p.lw  w1, 4(a0!)
p.lw  w2, 4(a1!)
p.lw  x1, 4(a2!)
p.lw  x2, 4(a3!)
pv.sdotp.b s1, w1, x1
pv.sdotp.b s2, w1, x2
pv.sdotp.b s3, w2, x1
pv.sdotp.b s4, w2, x2
end
```
PULP-NN: Data Reuse in the Register File

8-bit Convolution

CMSIS-NN based Matrix Multiplication Layout: 2x2  PULP-NN Matrix Multiplication Layout: 4x2

RegisterFile of the RI5CY core: 32 general purpose registers

2x2: 43% utilization

4x2: 69% utilization

More Data Reuse &
Higher utilization of the RF

Peak Performance (8 cores)

2x2: 12.8 MAC/cyc
4x2: 15.5 MAC/cyc
Multiple RI5CY Cores (1-16)
Low-Latency Shared TCDM
DMA for data transfers from/to L2
Shared Icashed with private “loop buffer”
Results: RV32IMCXpulp vs RV32IMC

8-bit Convolution Results

Overall Speedup of 75x

PULP-NN: an open Source library for DNN inference on PULP cores

3pJ/OP in fdx22

10x Speedup w.r.t. RV32IMC (ISA does matter 😊)
An additional controller is used for I/O
Hardware Processing Engines (HWPEs)

- **Memory**
  - on the data plane, memory "sees" HWPEs as a set of SW cores

- **RISCY**
  - on the control plane, cores control HWPEs as a memory mapped peripheral (e.g. a DMA)

- **HWPE**
  - "Virtual" Core #N+1
  - "Virtual" Core #N+2
  - "Virtual" Core #N+3

- **"Virtual" peripheral (e.g. DMA)**

**Diagram:**
- Data plane
- Control plane
- HWPE wrapper
- Address generation
- TCDM interconnect
- Shared memory
- Cores
- Register file
- Peripheral interconnect
- Control logic
- "Virtual" Core #N+1
- "Virtual" Core #N+2
- "Virtual" Core #N+3

"Virtual" Core #N+1

"Virtual" Core #N+2

"Virtual" Core #N+3
1. Perform convolve-accumulate in streaming fashion.

2. Decouple the streaming domain from the shared memory domain; convert streams in 3D-strided memory accesses.

3. Allow "jobs" to be offloaded to the HWCE by the RISC-V cores.

4. Weights for each convolution filter are stored privately.

5. Fine-grain clock gating to minimize dynamic power.

HWCE Sum-of-Products

Each input channel pixel is read $N_{outFeatures}$ times.

Each weight is read once.

Each output channel's partial sum pixel is read $(N_{inFeatures} - 1)$ times.

**For each output channel**

```plaintext
foreach (out_feature y):
    foreach (in_feature x):
        y += conv(W, x)
```

**Linebuffer:** min-bandwidth sliding window

$W$: 5x5 16-bit filters

1x 16-bit pixel

ADDER TREE

$y_{in}$: 1x 16-bit pixel

$y_{out}$: 1x 16-bit pixel

Cluster performance and energy efficiency on a 64x64 CNN layer (5x5 conv)

Scaled to ST FD-SOI 28nm @ Vdd=0.6V, f=115MHz

**PERFORMANCE**

- 1 core 4 cores vectorized 16b weights 8b weights 4b weights

- SW
- HWCE

13 GOp/s
218x

**ENERGY EFFICIENCY**

- 1 core 4 cores vectorized 16b weights 8b weights 4b weights

- SW
- HWCE

3250 GOp/s/W
84x
5x

0.3 pJ/Op

Now coming: HWCEv5.0 – improves scalability & flexibility @ 3TOPS/W
PULP cluster+MCU+HWCE(V1) $\rightarrow$ GWT’s GAP8 (55 TSMC)

Two independent clock and voltage domains, from 0-133MHz/1V up to 0-250MHz/1.2V

**MCU Function**
- Extended RISC-V core
- Extensive I/O set
- Micro DMA
- Embedded DC/DC converter
- Secured execution

**Computation engine**
- 8 extended RISC-V clusters
- Fully programmable
- Efficient parallelization
- Shared instruction cache
- Multi channel DMA
- HW synchronization
- HW convolution Engine

---

<table>
<thead>
<tr>
<th>What</th>
<th>Freq MHz</th>
<th>Exec Time ms</th>
<th>Cycles</th>
<th>Power mW</th>
</tr>
</thead>
<tbody>
<tr>
<td>40nm Dual Issue MCU</td>
<td>216</td>
<td>99.1</td>
<td>21 400 000</td>
<td>60</td>
</tr>
<tr>
<td>GAP8 @1.0V</td>
<td>15.4</td>
<td>99.1</td>
<td>1 500 000</td>
<td>3.7</td>
</tr>
<tr>
<td>GAP8 @1.2V</td>
<td>175</td>
<td>8.7</td>
<td>1 500 000</td>
<td>70</td>
</tr>
<tr>
<td>GAP8 @1.0V w HWCE</td>
<td>4.7</td>
<td>99.1</td>
<td>460 000</td>
<td>0.8</td>
</tr>
</tbody>
</table>

4x More efficiency at less than 10% area cost
New Application Frontiers: DroNET on NanoDrone

Pluggable PCB: PULP-Shield
- ~5g, 30×28mm
- GAP8 SoC
- 8 MB HDRAM
- 16 MB HFlash
- QVGA ULP HiMax camera
- Crazyflie 2.0 nano-drone (27g)

Only onboard computation for autonomous flight + obstacle avoidance
no human operator, no ad-hoc external signals, and no remote base-station!
More Efficiency (2): Extreme Quantization

Low(er) precision: 8→4→2

<table>
<thead>
<tr>
<th>Model</th>
<th>Bit-width</th>
<th>Top-1 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 ref</td>
<td>32</td>
<td>31.73%</td>
</tr>
<tr>
<td>INQ</td>
<td>5</td>
<td>31.02%</td>
</tr>
<tr>
<td>INQ</td>
<td>4</td>
<td>31.11%</td>
</tr>
<tr>
<td>INQ</td>
<td>3</td>
<td>31.92%</td>
</tr>
<tr>
<td>INQ</td>
<td>2 (ternary)</td>
<td>33.98%</td>
</tr>
</tbody>
</table>

SOA INQ retraining

2.2% loss → 0% with 20% larger net

MULT → MUX

RISC-V ISA Extensions for extreme quantization

RI5CY microarchitectural extensions

- Overheads (28nm FDX PULPissimo impl.):
  - Area: ~11% (vs. Ri5CY)
  - Timing Overhead: negligible (integrated in PULPissimo)
  - 8-bit MatMul power overhead: 1.8% (integrated in PULPissimo)
  - GP-app power overhead: 3.5% (integrated in PULPissimo)
From +/-1 Binarization to XNORs

\[ y(k_{out}) = \text{binarize}_{\pm 1} \left( b_{k_{out}} + \sum_{k_{in}} \left(W(k_{out}, k_{in}) \otimes x(k_{in})\right) \right) \]

\[ \text{binarize}_{\pm 1}(t) = \text{sign} \left( \gamma \frac{t - \mu}{\sigma} + \beta \right) \]

\[ \text{binarize}_{0,1}(t) = \begin{cases} 
1 & \text{if } t \geq -\kappa/\lambda = \tau, \text{ else } 0 \quad \text{(when } \lambda > 0) \\
1 & \text{if } t \leq -\kappa/\lambda = \tau, \text{ else } 0 \quad \text{(when } \lambda < 0) 
\end{cases} \]

\[ y(k_{out}) = \text{binarize}_{0,1} \left( \sum_{k_{in}} \left(W(k_{out}, k_{in}) \otimes x(k_{in})\right) \right) \]

Thresholding

Multi-bit accumulation

**Binary product \rightarrow XOR**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Main unit: binary dot-product and thresholding
Quentin: a XNE-accelerated microcontroller

Quentin in GlobalFoundries 22FDX

XNE area is \(~14000\) \(\text{um}^2\) (71 KGE, 72% Riscy+FPU)

Private memory is 448 KB SRAM + 3r2w 8 KB SCM

Shared memory is 56 KB SRAM + 8 KB SCM

With SRAMs, max eff @ 0.65V 8.7 Top/s/W

With SCMs, max eff @ 0.5V 46.3 Top/s/W

Accuracy Loss is high even with retraining (10%+) \(\rightarrow\) mixed precision
TWN & TCN are also a very appealing alternative (under design)
Binary-based Quantization (BBQ)

Normal NN layer: \[ y(k_{out}) = b(k_{out}) + \sum_{k_{in}} (W(k_{out}, k_{in}) \otimes x(k_{in})) \]

Inspired by ABC-Net:

BBQ NN layer: \[ y(k_{out}) \approx b(k_{out}) + \sum_{i=0..Q_W} \sum_{j=0..Q_A} \sum_{k_{in}} \alpha_i \beta_j W_{\text{bin}}(k_{out}, k_{in}) \otimes x_{\text{bin}}(k_{in}) \]

\( Q_W \): weight quantization level
\( Q_A \): activation quantization level

One quantized NN can be emulated by superposition of power-of-2 weighted \( Q_A \times Q_W \) binary NN
Reconfigurable Binary Engine

\[ y(k_{out}) \approx b(k_{out}) + \sum_{i=0..Q_W} \sum_{j=0..Q_A} \sum_{k_{in}} \alpha_i \beta_j (W_{bin}(k_{out}, k_{in}) \otimes x_{bin}(k_{in})) \]
Mixed-Precision Quantized Networks

Apply minimum tensor-wise quantization to fit the memory constraints with very-low accuracy drop.

Mixed-precision quantization rule-based bit precision selection based on memory constraints.

Only -2% wrt most accurate INT8 mobilenet (224_1.0) which does not fit on-chip.

+8% wrt most accurate INT8 mobilenet fitting on-chip (192_0.5).

+7.5% wrt most accurate INT4 mobilenet (224_1.0) fitting on chip.
Diversion: why not a fully hardwired engine?

However....

Flexibility is essential!

Amdhal’s effect!
+ Mixed precision
+ NN “zoo”

1000TOPS/W…
What about uW «sleep»?
Small always-on network → triggers alarm and video capture/streaming for cloud-based forensics

JPEG encoding on cluster:

- Fixed Point DCT: ~20fps@50Mhz
- Fast-DCT: ~20fps@50Mhz
- 8 Cores & Optimizations: ~20fps@50Mhz

Target 30 fps ...
Need uW always-on Intelligence

Always-on IO Accelerator!
Not Only CNNs: Hyper-Dimensional Computing

Low Dimensional Input Data (e.g. 32-bit int) → Mapping → [0 1 0 1 .............. 1]

High-dimensional → Holographic → Distributed → Pseudorandom with i.i.d. components

HD-Encoding

- Component-wise Majority
- XOR
- Permutation

Search Vector

Similarity Search (e.g. Hamming Distance)

Associate Memory

[0 1 0 1 .............. 1]
[1 1 1 0 .............. 1]
[1 1 0 0 .............. 0]
[0 1 1 1 .............. 1]
[1 1 1 1 .............. 1]
[1 1 0 1 .............. 1]
[0 1 0 1 .............. 1]
[0 1 0 1 .............. 1]

Prototype Vectors

Highly parallel, fault-tolerant, binary operators, assoc-min-distance search → Merge storage & computation, i.e. In-memory computing
More efficiency (3): HD-Based smart Wake-Up Module

Specifications

- Area: 670kGE
- Max. Frequency: 3 MHz
- SCM-Memory: 32 kBit
- Module Power Consumption (@ 1 kSPS/channel, 3 channels): ~ 15μW

Taped out in 22fdx
Spiking Convolutional Neural Network

- Activity-driven computation
- Event-like (sparse) feature representation
- No processing in absence of input events
- Lightweight pre-processing required on emerging Event-based sensor data (e.g. DVS cameras)
Leaky Integrate&Fire CSNN layer

Input spike stream
- 64 channel
- (8x8 matrix)
- 100 time intervals

Weighted stream
Conv. (3x3 kernel)

Membrane potential
LIF state variable

Output spikes
When Membrane potential exceed the threshold
SNE Accelerator Architecture

- Programmable streamer autonomous data fetching
- Data locality
  - Weight reuse and Neuron state stored locally
- Data re-ordering optimized for Convolutional SNN
- Digital Leaky Integrate&FIRE neuron model

**PULP SoC interconnect**

**Combinational crossbar**
- High data routing flexibility and low-latency communication

**High Modularity**
- Configurable number of LIF neuron blocks
  - Datapath “slices”
First silicon implementation

ROSETTA SoC

SoC physical implementation
- TSMC65 nm technology
- Chip area 4100μm x 3000μm
- Gates full SoC 6M

Accelerator physical implementation
- Area Accelerator 333μm²
- Gates Accelerator ~260k

Rosetta’s SNE configuration
- 4x64 time domain multiplexed LIF neurons per slice
- 8kB per data-path slice
- 2 data-path slices

<table>
<thead>
<tr>
<th>Number of Neurons</th>
<th>Memory (Accelerator)</th>
<th>Target Frequency</th>
<th>Performance (estimation)</th>
<th>Synaptic OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>16kB</td>
<td>250MHz</td>
<td>1.2 TOP/s</td>
<td>~250 GSOP/s</td>
</tr>
</tbody>
</table>
More Efficiency (4): Focal Plane Processing

Enable the extraction of low-level features in a parallel and efficient way by **integrating pixel-wise mixed-signal processing circuits** on the sensor die to reduce the imager energy costs.

Ultra-Low Power Imaging (GrainCam)

Imager performing spatial filtering and binarization on the sensor die through mixed-signal sensing!

This process naturally reflects the operation of a binarized pixel-wise convolution and can be seen as embedding the first convolutional layer within the image sensor die.

Combinational “Fully Spatial” BNN

Binarized Neural Network for early detection as combinational logic (with registers)

Mixed-Signal Sensing
Focal Plane Binarization

Mixed-Signal Sensing

Combinational Digital Logic

Apply XNOR and binarization over input data to produce any of output pixels
Synthesis Results

Synthesis of both models with hard-wired or reconfigurable weights

GF 22nm SOI with LVT cells (typical corner case 0.65V, 25°C)

Massive area reduction when hard-wiring the weights:

- XNOR operations reduce to wires or inverter, which can be also shared among different receptive fields
- popcounts also exploits sharing mechanisms

Advanced Synthesis Tools become central to exploit weights and intermediate results sharing to reduce the area occupation

2018 IEEE International Symposium on Circuits and Systems (ISCAS), 1-5
Conclusion

- Near-sensor processing → Energy efficiency pJ/OP and below
  - Ultra-low power architecture and circuits are needed
  - Memory is THE challenge
- TinyML: Inference can be squeezed into mW envelope
  - Non-von-Neumann acceleration → remove Imem bottleneck
  - Very robust at low precision → memory footprint reduction
  - fJ/OP is in sight! (100+TFLOPs/W) → mW inference engines!
- Pushing on the memory+IO bottleneck
  - TCDM + SCM memory hierarchy optimization and logic+physical optimization
  - In-place, in memory, staged, event-based, non-DNN inference
  - Better memories and memory interfaces (NVM, HBM…)
  - Multi-chip Systolic (2.5D chiplets)
  - Feature map compression

Open-Source Innovation ecosystem!
The fun is just beginning

http://pulp-platform.org
### Closing the Bin/Ternarization accuracy Gap

Soa results (Sept19)
1. First and last layer FP
2. ResNets have type-B bypasses (with 1x1 conv. in the non residual paths)
3. Modified network 2.25x more weights

#### Table 1: Experimental Results on ImageNet

<table>
<thead>
<tr>
<th>Model</th>
<th>Method*</th>
<th>Levels†</th>
<th>Accuracy [%] (top-1/top-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>baseline</td>
<td>torchvision v0.4.0</td>
<td>full-prec.</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>QN</td>
<td>(Yang et al., 2019)</td>
<td>5: ${o_i}_i$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>5: ${0} \cup {\pm 2^i}_i$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>LQ-Nets</td>
<td>(Zhang et al., 2018)</td>
<td>4: ${\pm o_i}_i$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>QN</td>
<td>(Yang et al., 2019)</td>
<td>3: ${o_1, o_2, o_3}$</td>
</tr>
<tr>
<td>ResNet-18+‡</td>
<td>TTQ</td>
<td>(Zhu et al., 2017)</td>
<td>3: ${o_1, 0, o_3}$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>INQ</td>
<td>(Zhou et al., 2017)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-18+‡</td>
<td>TWN</td>
<td>(Li et al., 2016)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>TWN</td>
<td>(Li et al., 2016)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td><strong>RPR (ours)</strong></td>
<td></td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>baseline</td>
<td>torchvision v0.4.0</td>
<td>full-prec.</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>TWN</td>
<td>(Li et al., 2016)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td><strong>RPR (ours)</strong></td>
<td></td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>XNOR-net BWN</td>
<td>(Rastegari et al., 2016)</td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>ResNet-50</td>
<td><strong>RPR (ours)</strong></td>
<td></td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>baseline</td>
<td>torchvision v0.4.0</td>
<td>full-prec.</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>TWN</td>
<td>(Li et al., 2016)</td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td><strong>RPR (ours)</strong></td>
<td></td>
<td>3: ${-1, 0, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>ADMM</td>
<td>(Leng et al., 2018)</td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>XNOR-net BWN</td>
<td>(Rastegari et al., 2016)</td>
<td>2: ${-1, 1}$</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td><strong>RPR (ours)</strong></td>
<td></td>
<td>2: ${-1, 1}$</td>
</tr>
</tbody>
</table>