Working with RISC-V

Part 3 of 5: PULP concepts

Luca Benini <luca.benini@unibo.it>
Davide Rossi <davide.rossi@unibo.it>
Summary

- Part 1 – Introduction to RISC-V ISA
- Part 2 – Advanced RISC-V Architectures
- Part 3 – PULP concepts
  - Cores
  - Clusters
  - Heterogeneous systems
- Part 4 – PULP Extensions and Accelerators
- Part 5 – PULP based chips
AI Workloads from Cloud to Edge (Extreme?)

- High OP/B ratio
- Massive Parallelism
- MAC-dominated
- Low precision OK
- Model redundancy

GOP+
MB+
Energy efficiency @ GOPS is **THE** Challenge

High performance MCUs

Low-Power MCUs

1pJ/OP=1TOPS/W

InceptionV4 @1fps in 10mW

Cool.. But how??

J Pineda, NXP + Updates
RI5CY – Recap from Part 1

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

V1 Baseline RISC-V RV32IMC (not good for ML)
V2 HW loops, Post modified Load/Store, Mac
V3 SIMD 2/4 + DotProduct + Shuffling
Bit manipulation, Lightweight fixed point

XPULP 25 kGE → 40 kGE (1.6x) but 9+ times DSP!

40 kGE
70% RF+DP

Xpulp Extensions Performance

Nice – But what about the GOPS/W? Faster+Superscalar is not efficient!

M7: 5.01 CoreMark/MHz - 58.5 µW/MHz
M4: 3.42 CoreMark/MHz - 12.26 µW/MHz

ACACES 2021 - Sept 2021
ML & Parallel, Near-threshold: a Marriage Made in Heaven

- 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

ML is massively parallel and scales well (P/S ↑ with NN size)

Better to have N× PEs running at lower voltage than one PE at nominal voltage!
Multiple RI5CY Cores (1-16)
Low-Latency Shared TCDM

Tightly Coupled Data Memory

Mem
Mem
Mem
Mem
Mem
Mem
Mem
Mem

Logarithmic Interconnect

RISC-V core
RISC-V core
RISC-V core
RISC-V core

CLUSTER
High speed single clock logarithmic interconnect

Ultra-low latency → short wires + 1 clock cycle latency

World-level bank interleaving «emulates» multiported mem

**TCDM Interconnect**

- **Synchronous low latency crossbar based on binary trees**
  - Single or dual channel (Each with $2^n$ master ports, $n=[0,1,2 ...]$)
  - **Distributed Arbitration** (Round robin)
  - Combinational handshake (single phase)
  - Deterministic access latency (1 cycle) + response in second cycle
  - Test&Set Supported
  - Word-level Interleaving

- **Emulates Multiported RAMs**
  - Slaves are pure memories.
  - Bank conflicts can be alleviated $\rightarrow$ higher Banking Factor (BF=1,2,4)
Peripheral Interconnect

- Synchronous low latency crossbar based on binary trees
  - Single or dual channel Each with $2^n$ master ports, $n=[0,1,2 \ldots]$
  - Distributed Arbitration (Round robin)
  - Combinatorial handshake (single phase)
  - Custom port mapping (Address ranges)
  - Decoupled request and response path
    - Slaves have unpredictable latencies

- Used to build Peripheral systems
  - Slaves are pure generic peripherals like bridges, timers etc (Req/grant)
  - Mostly used to move data in and out from the cluster processors
Fast synchronization and Atomics

Avoid busy waiting!
Minimize sw synchro. overhead
Efficient fine-grain parallelization

Private, per core port
→ single cycle latency
→ no contention
PULP Cluster Event Unit

Cluster synchronization unit

- periph. event FIFO
- HW barrier
- HW mutex
- HW dispatch

Software event matrix

- event logic
- event logic
- ... event logic

DMA

CNN acc.

- Event source
- Single-cycle
- Core accessible

periph.

control MCU

core 0

core 1

... core N
Energy-efficient Event Handling

- Single instruction rules it all:
  - address determines action
  - read datum provides corresponding information

```c
evt_data = evt_read(addr);
```

- event buffer content, program entry point, mutex message, triggering core id, ...
- trigger sw event, trigger barrier, try mutex lock, read entry point, auto buffer clear, ...

Diagram:
- core clock
- read req.
- read ack.
- event
- read data

Instruction turns off core clock → NO pipeline flush!
Results: Barrier

- Fully parallel access to SCU: Barrier cost constant
- Primitive energy cost: Down by up to 30x
- Minimum parallel section for 10% overhead in terms of …
  - … cycles: ~100 instead of > 1000 cycles
  - … energy: ~70 instead of > 2000 cycles
Results: Mutex

- Sequential execution: Cycle overhead always large
- TAS-variable inherently well-suited for mutex; lower cycle savings compared to barrier
- SCU still avoids L1 accesses: Energy of TAS mutex up to 1.6x higher
- Smallest parallel section for 10% energy overhead: ~1000 instead of 1600 cycles
Shared instruction cache with private “loop buffer”
The Memory Bottleneck

PULPv1 POWER BREAKDOWN (Back in 2015!) @ BEST ENERGY POINT:

- SRAM limits voltage scalability (very well known problem…)
- SRAM forms a huge bottleneck for energy efficiency (>60% of total power)
ULP (NT) Bottleneck: Memory

- “Standard” 6T SRAMs:
  - High VDDMIN
  - Bottleneck for energy efficiency
    - >50% of energy can go here!!!

- Near-Threshold SRAMs (8T)
  - Lower VDDMIN
  - Area/timing overhead (25%-50%)
  - High active energy
  - Low technology portability

- Standard Cell Memories:
  - Wide supply voltage range
  - Lower read/write energy (2x - 4x)
  - High technology portability
  - Major area overhead 4x → 2.7x with controlled placement

---

$I$: a Look Into ‘Real Life’ Applications

SCM-BASED $I$ IMPROVES EFFICIENCY BY $\sim$2X ON SMALL BENCHMARKS, BUT...

Survey of State of The Art

<table>
<thead>
<tr>
<th>Existing ULP processors</th>
<th>Latch based $I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>REISC (ESSCIRC2011)</td>
<td>64b</td>
</tr>
<tr>
<td>Sleepwalker (ISSCC 2012)</td>
<td>128b</td>
</tr>
<tr>
<td>Bellevue (ISCAS 2014)</td>
<td>128b</td>
</tr>
</tbody>
</table>

Applications on PULP

Issues:
1) Area Overhead of SCMs (4Kb/core not affordable….)
2) Capacity miss (with small caches)
3) Jumps due to runtime (e.g. OpenMP, OpenCL) and other function calls
Shared I$:

- **Shared instruction cache**
  - OK for data parallel execution model
  - Not OK for task parallel execution model, or very divergent parallel threads

- **Architectures**
  - **SP**: single-port banks connected through a read-only interconnect
    - Pros: Low area overhead
    - Cons: Timing pressure, contention
  - **MP**: Multi-ported banks
    - Pros: High efficiency
    - Cons: Area overhead (several ports)

- **Results**
  - Up to 40% better performance than private I$
  - Up to 30% better energy efficiency
  - Up to 20% better energy*area efficiency

---

Fixed-Point Parallel Speed-Up

Amdahl’s Limit

Speed-Up

Number of cores

SVM
MatMul (8-bit)
5x5 Conv (32-bit)
FFT
MatMul (32-bit)

CNN Layer (32-bit)
5x5 Conv (16-bit)
5x5 Conv (8-bit)
MatMul (16-bit)
DWT

PCA
HD
CNN Layer (16-bit)
BNN
CNN Layer (8-bit)
FIR
FP & Transprecision supported also in PULP

- Main FP operation groups
  - MUL/ADD: Add/Subtract, Multiply, FMA
  - CMP/SGNJ: Comparisons, Min/Max etc.
  - CAST: FP-FP casts, Int-FP / FP-Int casts

- Parametrizable
  - Number & Encoding of Formats
  - Packed-SIMD Vectors
  - # Pipeline Stages (per Op and Format)
  - Implementation (per Op and Format)
    - PARALLEL for best Speed
    - MERGED (or Iterative) for best Area

- Special Functions for Transprecision
  - Cast-and-Pack 2 FP Values to Vector
  - Casts amongst FP Vectors + Repacking
  - Expanding FMA (e.g. FP32 += FP16*FP16)

FPU Interconnect (1/2)

- Synchronous low-power/low latency crossbar used to share several FPUs in a multicore system:
  - $2^n$ master ports ($n=[0,1,2 ...]$)
  - $2^m$ slave ports ($m=[0,1,2 ...]$) → General purpose FPUs (ADD/SUB, MUL etc)
  - Combinatorial handshake (single phase)
- Allocator
  - Random: given a request, a random fpu is chosen
  - Optimal: Maxime the utilization of FPUs
FPU Interconnect (2/2)

- **Features:**
  - Independent response paths for each sub-FPU block
    - fully pipelined FPU sub-blocks with different latencies
  - Two Operators (A,B), one command (OP) and the ID are carried to the FPU.
  - No need of Flow control on the FPU side.
  - Flexible and parametrizable

![Example of FPU attached to the FPU interconnect](image)

![Graph showing performance and power comparison](image)

Private FPU
Results: Floating-Point NSAA Performance

- Almost linear parallelization speed-up
- Performance close to Amdahl limit when applications can be efficiently vectorized

DMA for data transfers from/to L2
PULP MCHAN DMA Engine

- A DMA engine optimized for integration in tightly coupled processor clusters
  - Dedicated, per-core non blocking programming channels
  - Ultra low latency programming (~10 cycles)
  - Small footprint (30K gates): avoid usage of large local FIFOs by forwarding data directly to TCDM (no store and forward)
  - Support for multiple outstanding transactions
  - Parallel RX/TX channels allow achieving full bandwidth for concurrent load and store operations

Configurable parameters:
- # of core channels
- Size of command queues
- Size of RX/TX buffer
- # of outstanding transactions

Integration in a PULP cluster

An additional I/O controller is used for IO
Efficient use of system resources
HW support for double buffering allows continuous data transfers
Multiple data streams can be time multiplexed
PULP interrupts controller (INTC)

- It generates interrupt requests from 0 to 31
- Mapped to the APB bus
- Receives events in a FIFO from the SoC Event Generator (i.e. from peripherals)
  - Unique interrupt ID (26) but different event ID
- Mask, pending interrupts, acknowledged interrupts, event id registers
- Set, Clear, Read and Write operations by means of load and store instructions (memory mapped operations)
- Interrupts come from:
  - Timers
  - GPIO (rise, fall events)
  - HWCE
  - Events i.e. uDMA
How do we work: Initiate a DMA transfer
Data copied from L2 into TCDM
Once data is transferred, event unit notifies cores
Cores can work on the data transferred

Tightly Coupled Data Memory

Ext. Mem

Mem Cont

L2 Mem

RISC-V core

I/O

PULPissimo

Interconnect

DMA

Event Unit

RISC-V core

I$
Once our work is done, DMA copies data back
During normal operation all of these occur concurrently.
Explicit Memory Management: MobileNet Example

- ~4 Mparameters → need to store weights in off-chip memory (L3)
  - L1 Bandwidth: 256 Gbit/s @ 250 MHz
  - L2 Bandwidth: 32 Gbit/s @ 250 MHz
  - L3 Bandwidth: 1.6 Gbit/s @ 250 MHz

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

L3 / L2 tiling
64 MB / 512 kB

small memory

big memory
Tensor tiling

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

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

- **small memory**
- **big memory**
Tile Data Movement

CONVOLUTIONAL PIPELINE

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
- Input feature map I
- Filters weights W
- Output feature map O

L1 memory

\[ t_0 \quad t_1 \quad t_2 \quad t_3 \quad \ldots \quad t_n \]

DMA ch. 0-1
- Cluster computation

DMA ch. 2

Working with RISC-V
Tile Data Movement

CONVOLUTIONAL PIPELINE

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

t₀  t₁  t₂  t₃  ...  tₙ

In. copy  Convol. kernel

DMA ch. 0-1

DMA ch. 2

Cluster computation
Tile Data Movement

CONVOLUTIONAL PIPELINE

\[
\begin{align*}
&t_0 \\
&\text{In. copy} \\
&\text{Convol. kernel} \\
&t_1 \\
&t_2 \\
&t_3 \\
&\quad \ldots \\
&t_n
\end{align*}
\]

\[
\begin{align*}
\text{L2 memory} & \quad \text{L1 memory} \\
\text{Input feature map } I & \quad \text{L1 buffer 1} \\
\text{Filters weights } W & \quad \text{x TILE 1} \\
\text{Output feature map } O & \quad \text{y TILE 1} \\
\text{L1 buffer 2} & \quad \text{W TILE 1} \\
\text{x TILE 2} & \quad \text{y TILE 2} \\
\text{W TILE 2} & \quad \text{In. copy} \\
\end{align*}
\]

\[
\begin{align*}
&h_M \\
&i_M \\
&w_M \\
&h_M \\
&i_M \\
&w_M
\end{align*}
\]

\[
\begin{align*}
&\text{DMA ch. 0-1} \\
&\text{DMA ch. 2} \\
&\text{Cluster computation}
\end{align*}
\]
Tile Data Movement

**CONVOLUTIONAL PIPELINE**

- **t₀**: In. copy
- **t₁**: Conv. kernel
- **t₂**: Out. copy
- **t₃**...

- **DMA ch. 0-1**
- **DMA ch. 2**
- **Cluster computation**
Tile Data Movement

CONVOLUTIONAL PIPELINE

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

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

In. copy  Conv. kernel  Out. copy  In. copy  Conv. kernel

In. copy  Conv. kernel  Out. copy  In. copy

In. copy  Conv. kernel  Conv. kernel  Out. copy

\[ t_0 \]  \[ t_1 \]  \[ t_2 \]  \[ t_3 \]  \ldots  \[ t_n \]

L2 memory

L1 memory

Input feature map I

Output feature map O

Filters weights W

h_M  \quad w_M  \quad i_M  \quad o_M

ACACES 2021 - Sept 2021

DMA ch. 0-1

DMA ch. 2

Cluster computation

Working with RISC-V
PULP includes Cores+Interco+IO+HWCE → Open Platform

RISC-V Cores
- RI5CY 32b
- Ibex 32b
- Snitch 32b
- Ariane + Ara 64b

Platforms
- Single Core
  - PULPino
  - PULPissimo
- Multi-core
  - Fulmine
  - Mr. Wolf
- Multi-cluster
  - Hero
  - Open Piton

Accelerators
- HWCE (convolution)
- Neurostream (ML)
- HWCrypt (crypto)
- PULPO (1st ord. opt.)

Interconnect
- Logarithmic interconnect
- APB – Peripheral Bus
- AXI4 – Interconnect
Nice, but what exactly is “open” in Open Source HW?

A modern, open, free ISA, extensible by construction
Endorsed and Supported by 300+ Companies

But… an open ISA is not Open HW (it’s a prerequisite!)
Nice, but what exactly is “open” in Open Source HW?

- Only the first stage of the silicon production pipeline can be open HW → **RTL source code** (in an HDL such as SystemVerilog)
- Later stages contain closed IP of various actors + tool licensing issues

Permissive, Copyright (e.g. APACHE) License is Key for industrial adoption
PULP is silicon proven

37 SoCs & counting
Luca Benini, Davide Rossi, Andrea Borghesi, Michele Magno, Simone Benatti, Francesco Conti, Francesco Beneventi, Daniele Palossi, Giuseppe Tagliavini, Antonio Pullini, Germain Haougou, Manuele Rusci, Florian Glaser, Fabio Montagna, Bjoern Forsberg, Pasquale Davide Schiavone, Alfio Di Mauro, Victor Javier Kartsch Morinigo, Tommaso Polonelli, Fabian Schuiki, Stefan Mach, Andreas Kurth, Florian Zaruba, Manuel Eggimann, Philipp Mayer, Marco Guermandi, Xiaying Wang, Michael Hersche, Robert Balas, Antonio Mastrandrea, Matheus Cavalcante, Angelo Garofalo, Alessio Burrello, Gianna Paulin, Georg Rutishauser, Andrea Cossettini, Luca Bertaccini, Maxim Mattheeuws, Samuel Riedel, Sergei Vostrikov, Vlad Niculescu, Hanna Mueller, Matteo Perotti, Nils Wistoff, Luca Bertaccini, Thorir Ingulfsson, Thomas Benz, Paul Scheffler, Alessio Burello, Moritz Scherer, Matteo Spallanzani, Andrea Bartolini, Frank K. Gurkaynak, and many more that we forgot to mention

http://pulp-platform.org  @pulp_platform