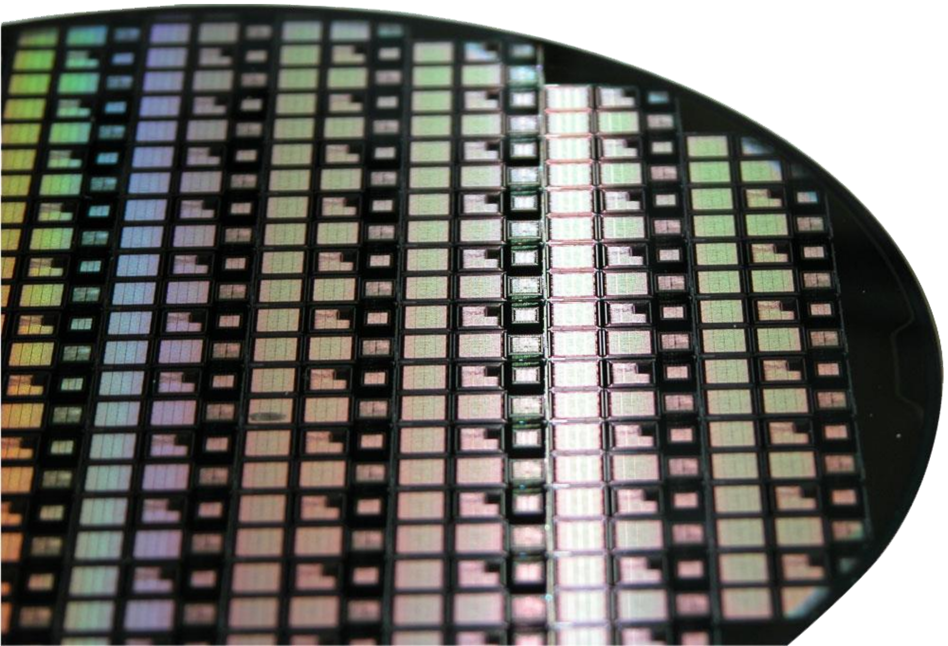
ETH zürich

DSP ISA Extensions for an Open-Source RISC-V Implementation

PULP

Parallel Ultra Low Power



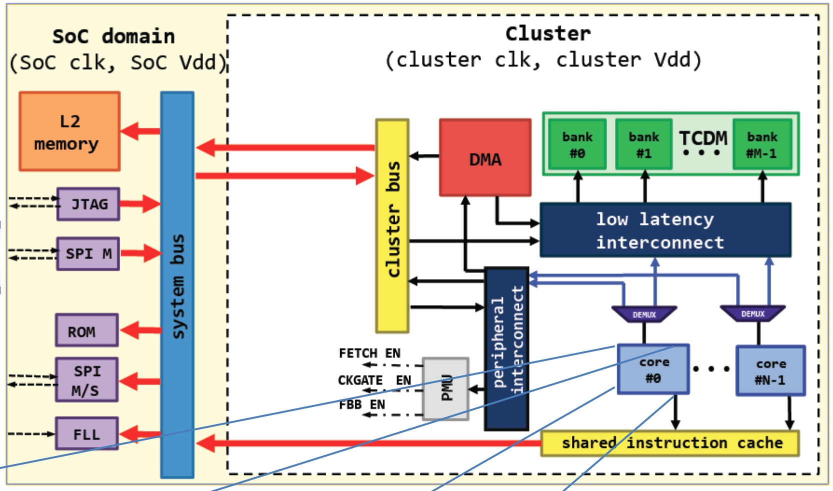
Davide Schiavone
Davide Rossi
Michael Gautschi
Eric Flamand
Andreas Traber
Luca Benini

ETH zürich
Integrated Systems Laboratory

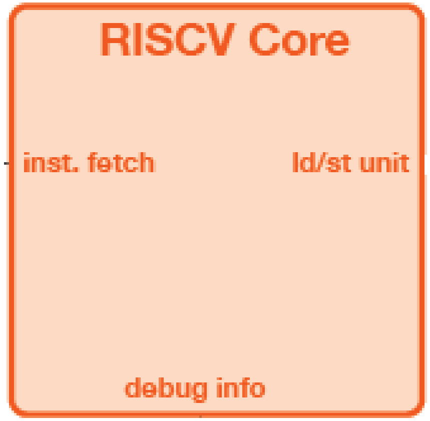
Introduction: a typical embedded system

Sensor

Analyze and Classify



Transmit



PULP

Energy Constrained Environment

RI5CYv2 Core

1. RI5CYv2 is an evolution of our RISC-V implementation RI5CY, in-order 4 pipeline stages
   (presented by Andreas Traber during last RISC-V Workshop)

2. It is a RV32IMC implementation plus some PULP specific ISA extensions to target energy efficiency

3. We support interrupts and exceptions and we provide a debug interface

4. We have GCC supports for all our extensions

5. Profiling and core execution trace

6. Our core targets tightly-coupled multi-core systems

→ **Fully featured microcontroller competitive with SOA microcontrollers**

Coremarks/MHz

Core	Coremark/MHz
RV32IM	2.36
RI5CYv1	2.75
RI5CYv2	2.86
ARM Cortex M4 [1]	2.85

[1] http://infocenter.arm.com/help/topic/com.arm.doc.dai0350a/DAI0350A_coremark_benchmarking.pdf

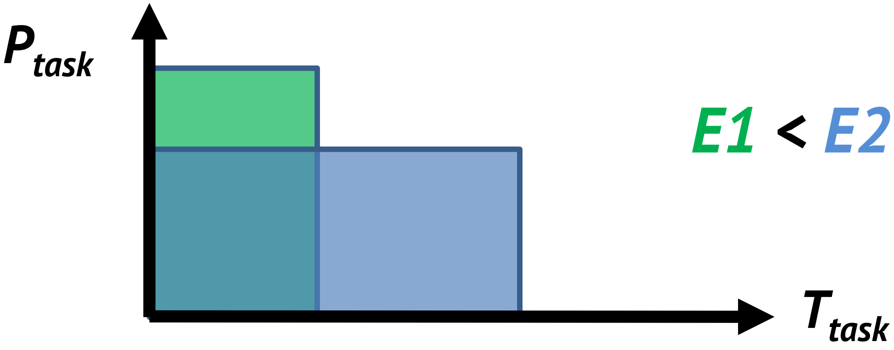
Why DSP extensions?

1. We target signal processing algorithms

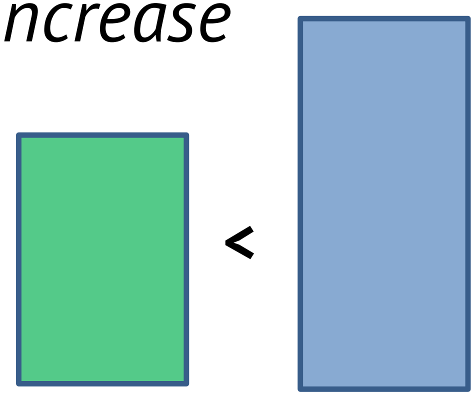
2. We need to speed up our applications to put the system in low power mode as soon as possible

How to be energy efficient?

Performance increase higher than Power increase



E1 < E2



RI5CYv1 Core previous ISA Extensions from OR10N

```
for (i = 0; i < 101; i ++)
   d[i] = a[i] + b[i];
```

Baseline

```
mv   x5, 0
mv   x4, 101
Lstart:
  lw      x2, 0(x10)
  lw      x3, 0(x11)
  addi  x10, x10, 4
  addi  x11, x11, 4
  add   x2, x3, x2
  sw      x2, 0(x12)
  addi  x4, x4, -1
  addi  x12, x12, 4
bne      x4, x5, Lstart
```

Auto-incr load/store

```
mv   x5, 0
mv   x4, 101
Lstart:
  lw   x2, 0(x10!)
  lw   x3, 0(x11!)
  add x2, x3, x2
  addi  x4, x4, -1
  sw x2, 0(x12!)
bne      x4, x5, Lstart
```

HW Loop

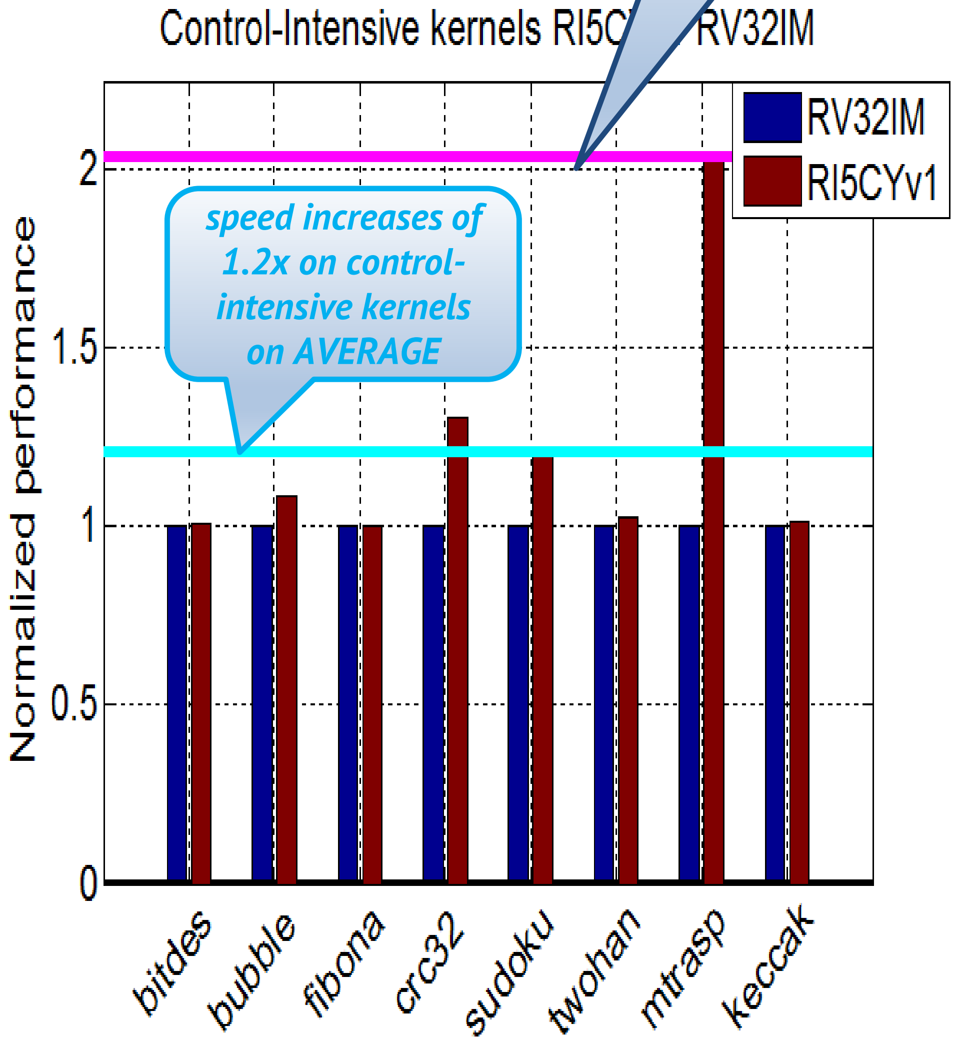
```
lp.setupi 101, Lend
  lw   x2, 0(x10!)
  lw   x3, 0(x11!)
  add x2, x3, x2
Lend: sw x2, 0(x12!)
```

RI5CYv1 vs RV32IM

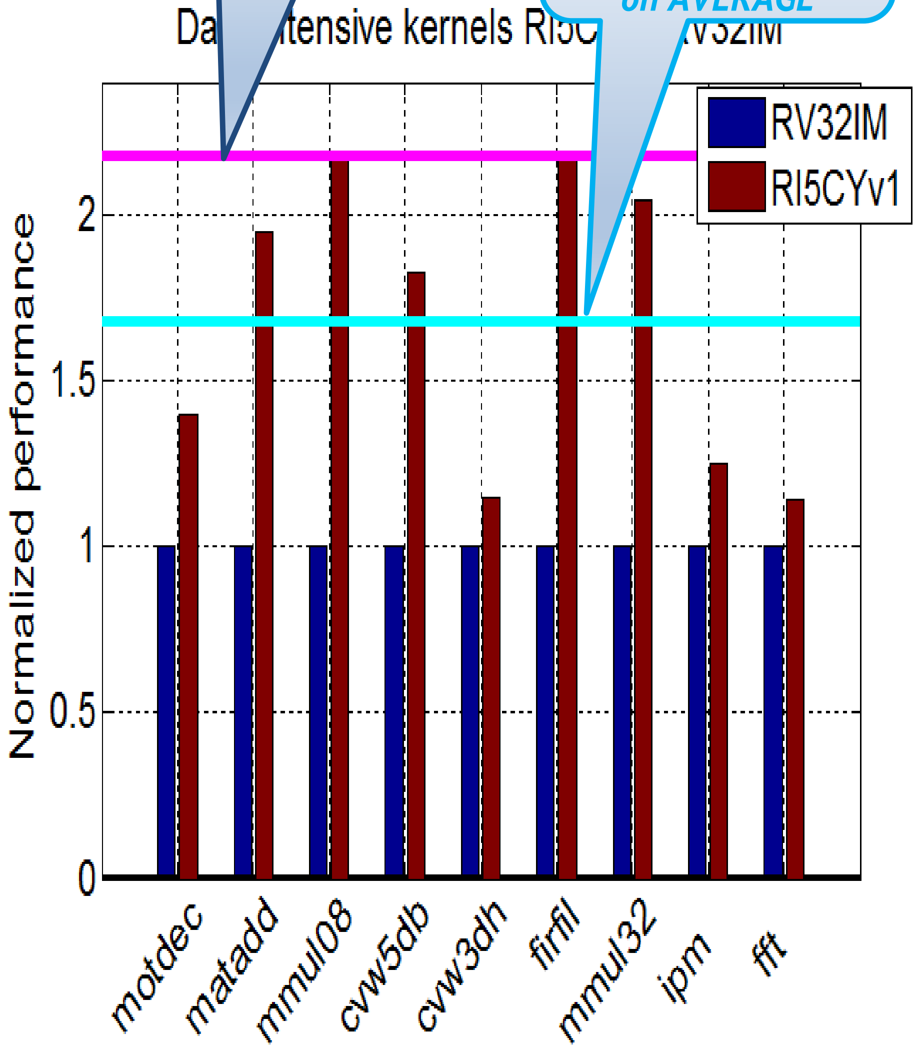
speed increases UP TO 2x on control-intensive kernels

speed increases UP TO 2.2x on data-intensive kernels

speed increases of 1.7x on data-intensive kernels on AVERAGE



speed increases of 1.2x on control-intensive kernels on AVERAGE

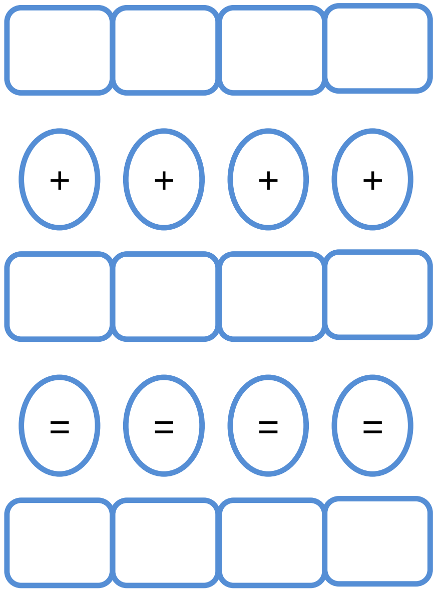


RI5CYv2 Core ISA Extensions

RI5CYv2 provides a set of extensions for small vectors manipulation typical of several DSP applications:

1. *Dot product between vectors*
2. *Saturation between [-2^n or 0 ; 2^n-1]*
3. *Mul/Add/Sub plus round and normalization (eg: fixed point)*
4. *Shuffle operations for vectors*
5. *Packed-SIMD ALU operations (ported from our previous OpenRisc implementations)*
6. *Bit manipulations*

Packed-SIMD Architecture



GCC support

RI5CYv2 new instructions are supported by our GCC.
We use attributes and built-in functions in our C code

```
typedef char charVec
__attribute__ ((vector_size (4)));

charVec a = {1, 2, 3, 4};

charVec res = a + a;
//res = {2,4,6,8}
```

→

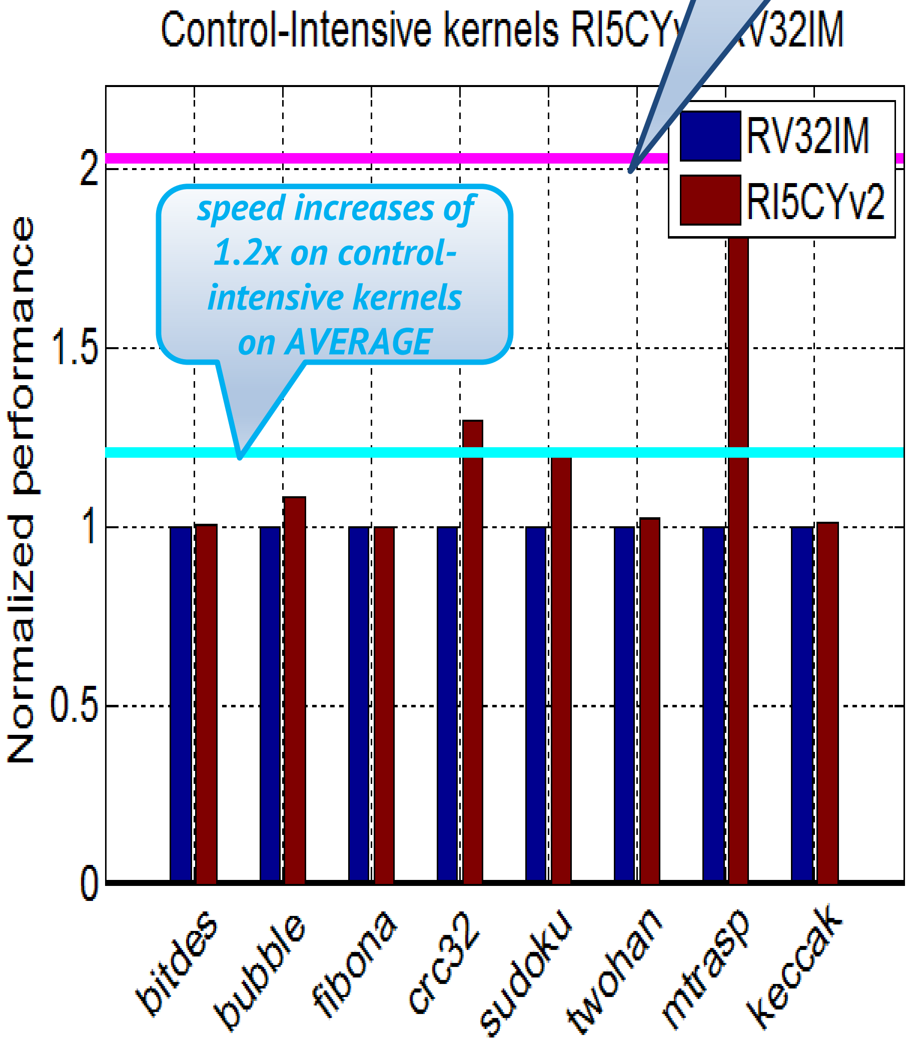
```
pv.add.b x5, x4, x3
sw x5, 0(x10)
```

```
typedef short shortVec
__attribute__ ((vector_size (2)));

shortVec a = {5, 6};
shortVec b = {-1, -2};

int res = __builtin__pulp_dotsp2(a,b);
//res = -1*5 -2*6 = -17
```

→

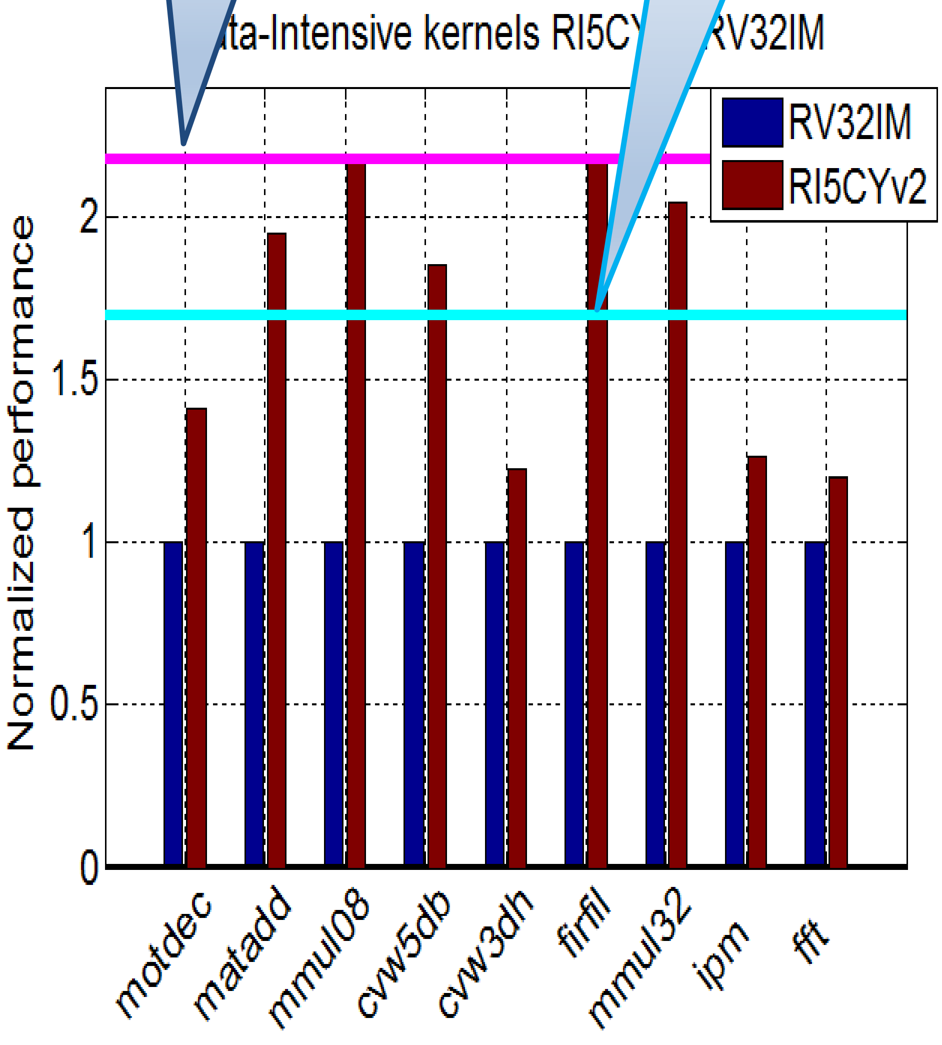
```
pv.dotsp.h x5, x4, x3
sw x5, 0(x10)
```

RI5CYv2 vs RV32IM



speed increases UP TO 2x on control-intensive kernels

speed increases of 1.2x on control-intensive kernels on AVERAGE

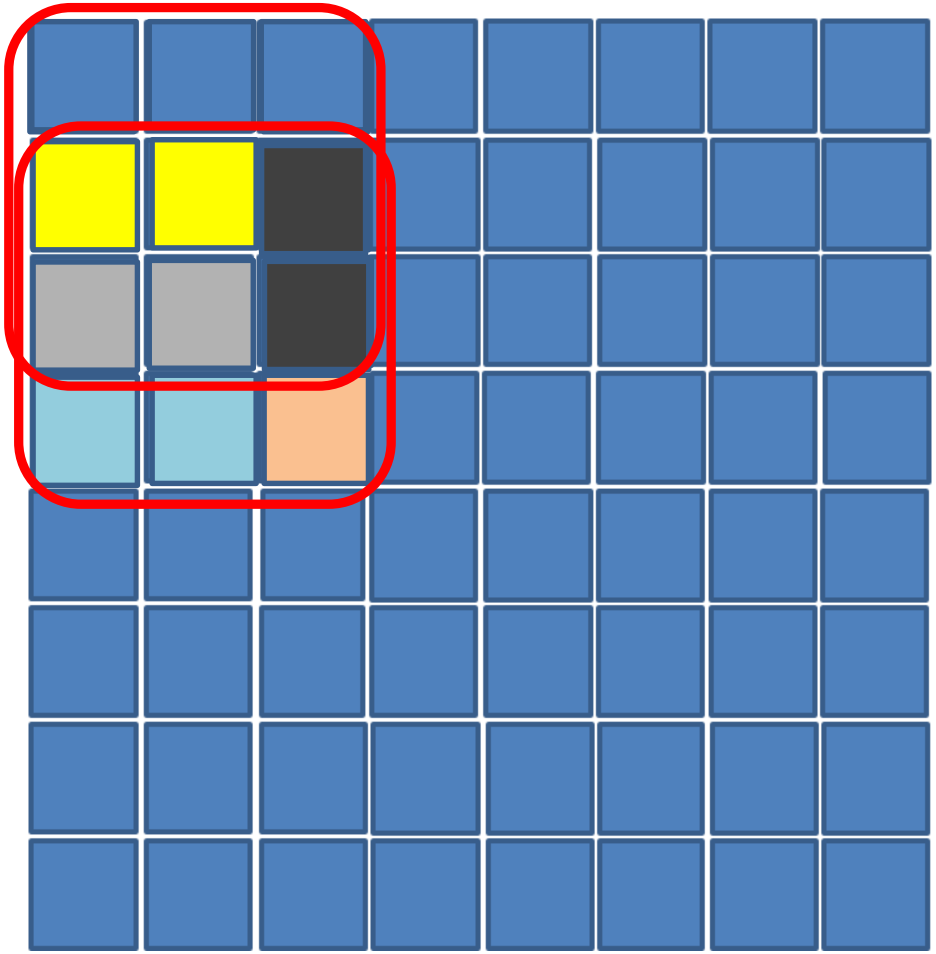


speed increases UP TO 2.2x on data-intensive kernels

speed increases of 1.7x on data-intensive kernels on AVERAGE

Optimized C code

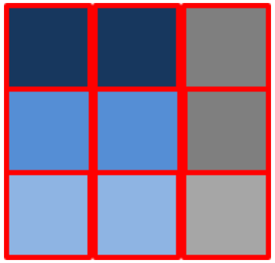
eg: 2D-Convolution $\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} h(k_1,k_2)x(n_1-k_1,n_2-k_2)$



Kernel

Load 5 vectors from the Kernel

K1, K2, K3, K4, K5



Load 5 vectors from the Image

V1, V2, V3, V4, V5

Compute the dot products

S = V1*K1
S += V2*K2
S += V3*K3
S += V4*K4
S += V5*K5

Doing it for each output pixel

2D-Convolution with our extensions: Pseudo C Code

Scalar

```
For each column c
    For each row r
        S = 0;
        for (i = -1; i <= 1; i++) {
            for (j = -1; j <= 1; j++) {
                compute index k, w
                data = InImg[k];
                coeff = Kernel[w];
                S = S + coeff*data;
            }
        }
       S = S + ROUNDBIT;
       S = S >> (DATAWIDTH-1);
       S = S > SAT ? SAT : S;
       S = S <      0 ?       0 : S;
      OutImg[t] = S;
```

Vector

```
load the 5 vectors from Filter
For each column c
    load 5 vectors from InImg[c]
    For each row R
        S = __builtin_pulp_dotsp2(V1,K1);
        S = __builtin_pulp_sdotsp2(V2,K2, S);
        S = __builtin_pulp_sdotsp2(V3,K3, S);
        S = __builtin_pulp_sdotsp2(V4,K4, S);
        S = __builtin_pulp_sdotsp2(V5,K5, S);
        S =  __builtin_pulp_addRN(S,0,DATAWIDTH-1,ROUNDBIT);
        //clip inferred automatically
        S = S > SAT ? SAT : S;
        S = S <      0 ?       0 : S;
        OutImg[t] = S;
        load the next 2 new vectors V6, V7 from InImg
        V1 = V2;
        V2 = V3;
        V3 = V6;
        V4 = __builtin_pulp_shuffle2h(V4,V5,mask);
        V5 = V7;
```

5 DOT PRODUCTs

ROUND AND SHIFT

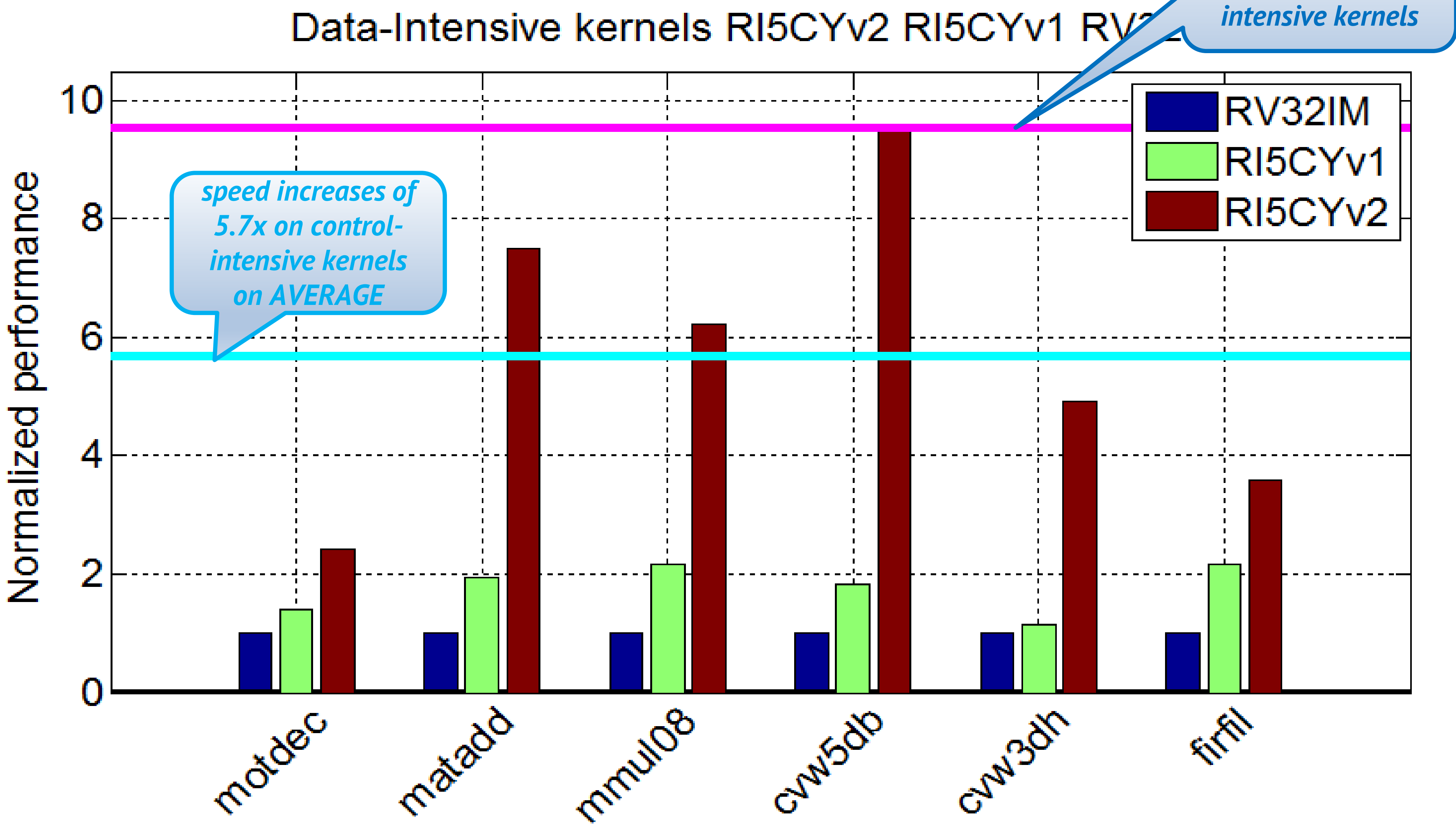
CLIP

STORE PIXEL

SHIFT BACK THE WINDOW

RI5CYv2 optC vs RI5CYv1 vs RV32IM on data-intensive kernels

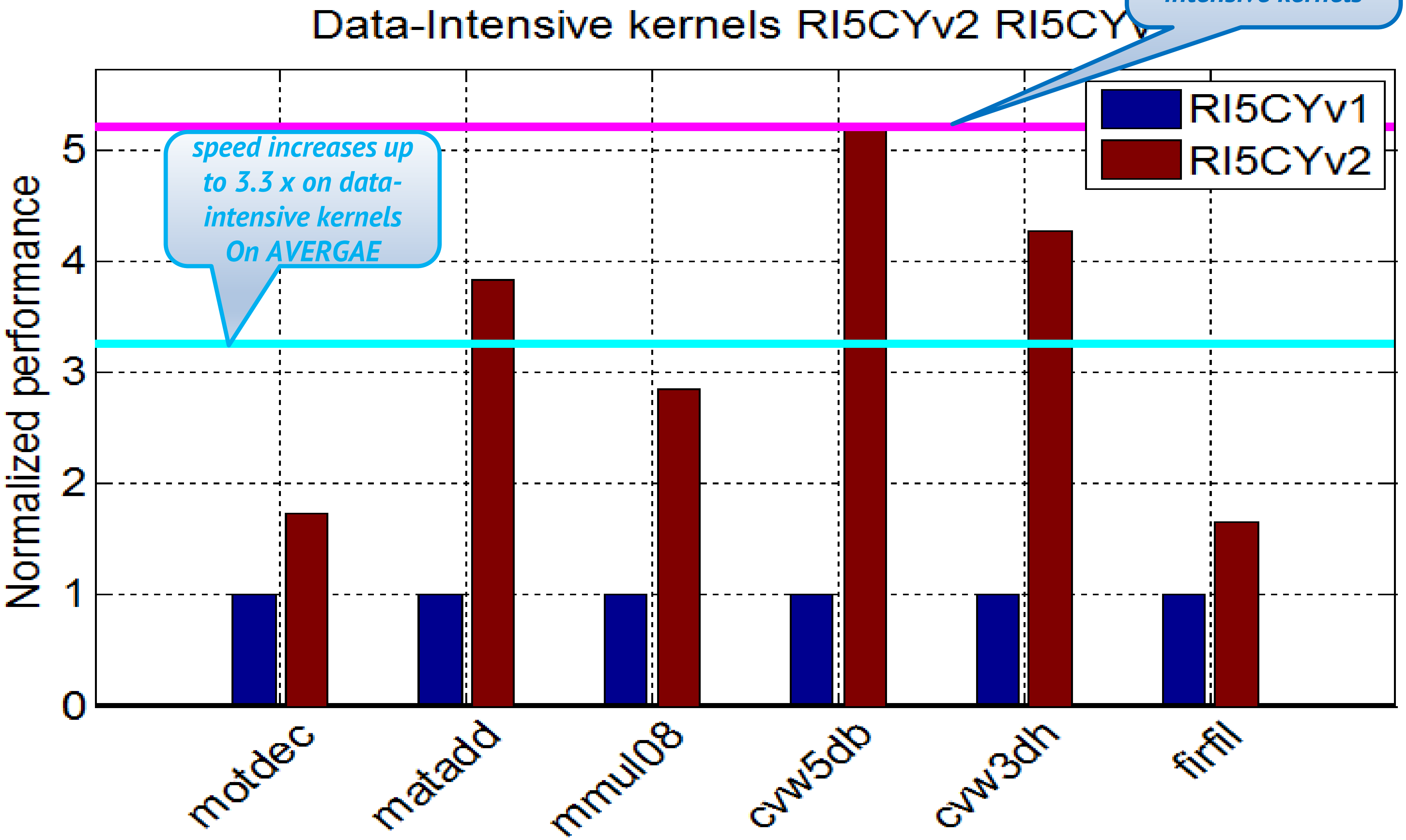
Using the optimized C code performance increases up to 9.5x!!!



speed increases UP TO 9.5 x on data-intensive kernels

speed increases of 5.7x on control-intensive kernels on AVERAGE

RI5CYv2 optC vs RI5CYv1 on data-intensive kernels



speed increases UP TO 5.2 x on data-intensive kernels

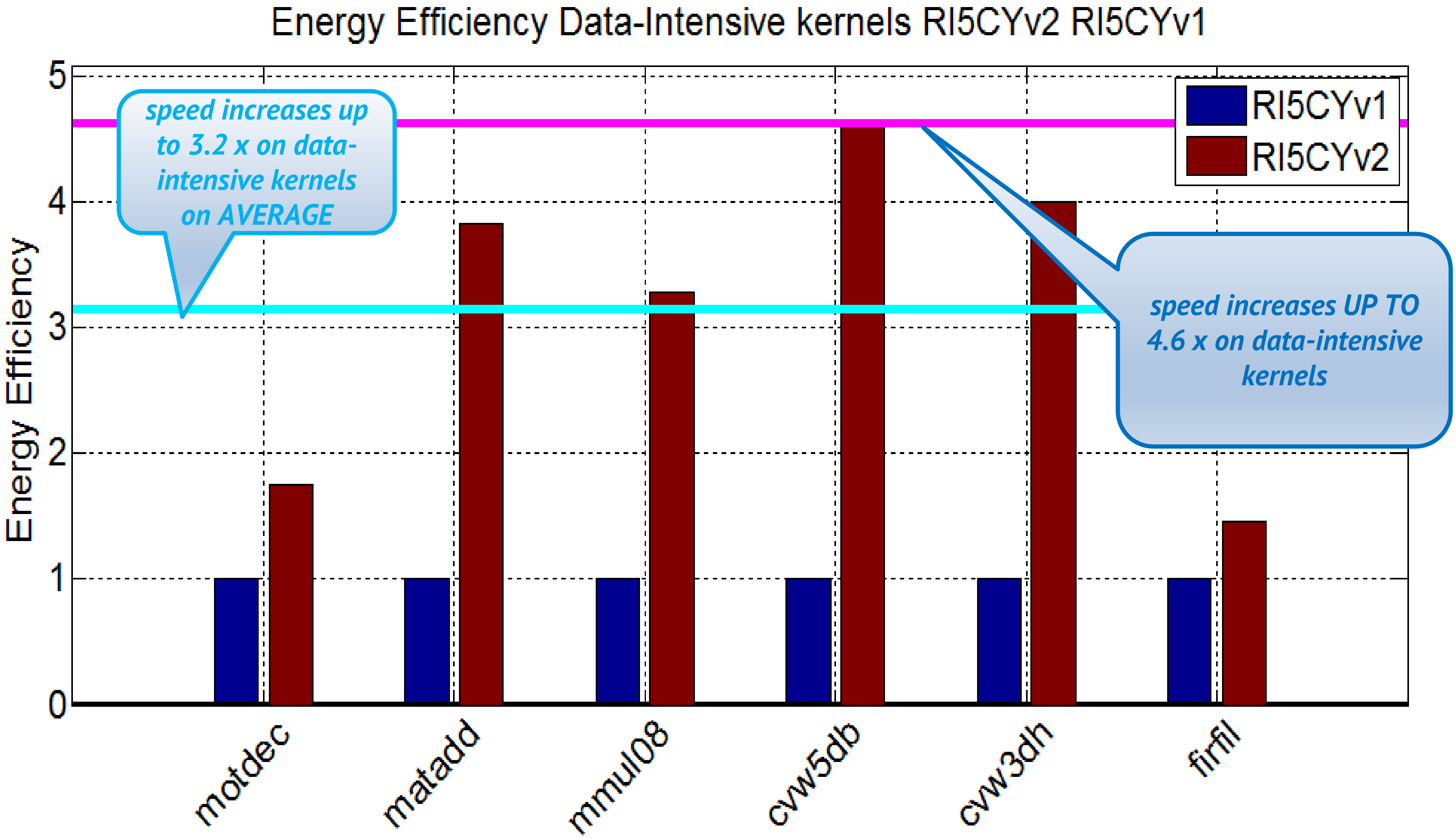
speed increases up to 3.3 x on data-intensive kernels On AVERGAE

RI5CYv2: Timing, Area and Power

	RI5CYv2
Technology	65 nm
Conditions	25°C, 1.2V
Dynamic Power $[\mu W/MHz]$	30.6
Area $[mm^2]$	0.074
Max Frequency	650 MHz

Running dsp optimized kernels increases power up to +14%, ***but…***

Energy: RI5CYv2 optC vs RI5CYv1 on data-intensive kernels [*]



speed increases up to 3.2 x on data-intensive kernels on AVERAGE

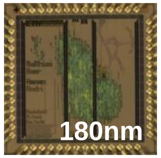
speed increases UP TO 4.6 x on data-intensive kernels

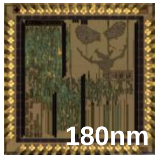
[*] Power estimated using the same core with different compiler options

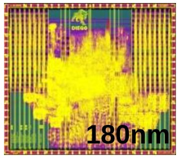
Building PULP: silicon roadmap

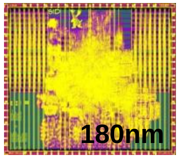
- ***Main PULP chips*** *(ST 28nm FDSOI)*
  - *PULPv1*
  - *PULPv2*
  - *PULPv3 (under test)*
  - ***PULPv4 (RISC-V based systems) (in progress)***
- ***PULP development*** *(UMC 65nm)*
  - *Artemis - IEEE 754 FPU*
  - *Hecate - Shared FPU*
  - *Selene - Logarithic Number System FPU*
  - *Diana - Approximate FPU*
  - *Mia Wallace – full system*
  - ***Imperio - PULPino chip (RISC-V based systems)***
  - *Fulmine – Secure HW accelerator cluster*
- ***PULP development*** *(GF 28nm)*
  - ***Honey Bunny (RISC-V based systems)***

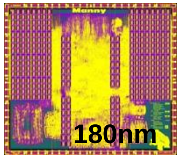
- **Early building blocks** (UMC180)
  - Sir10us
  - Or10n
- **Mixed-signal systems** (SMIC 130nm)
  - VivoSoC
  - VivoSoc2
- **IcySoC chips approx. computing platforms** (ALP 180nm)
  - Diego
  - Manny
  - Sid

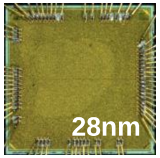


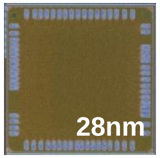


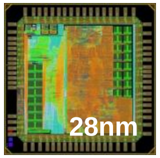


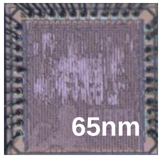


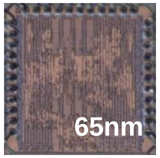


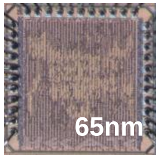


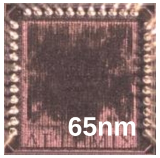






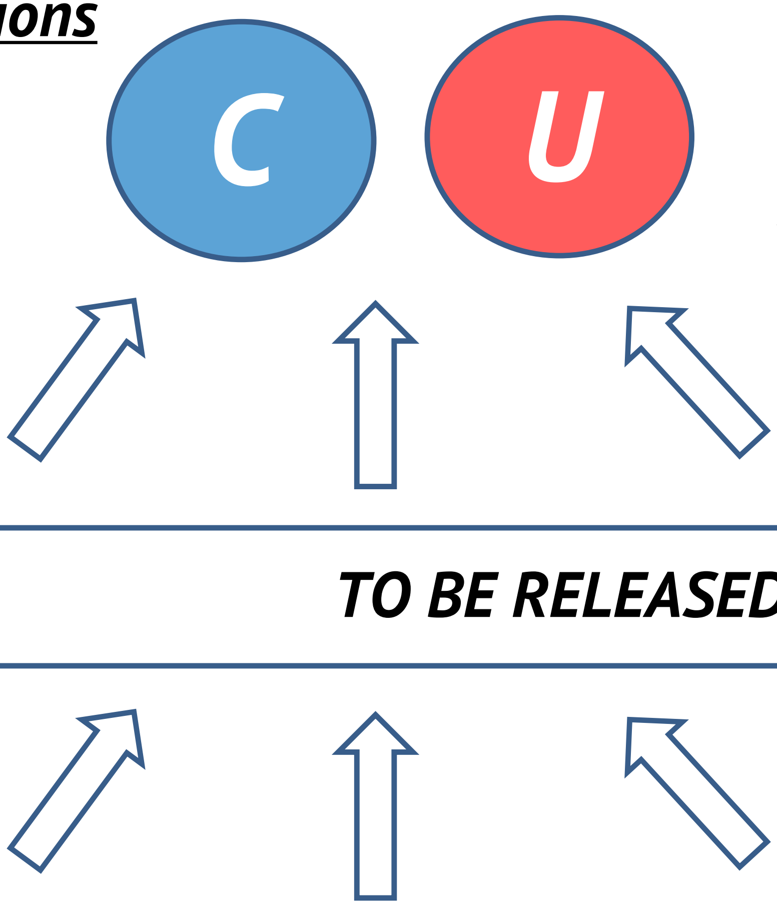






Users and collaborations

7 COMPANIES



7 UNIVERSITIES

...they are growing
…and growing

PULP
Parallel Ultra Low Power

RELESED | TO BE RELEASED

GREENWAVES TECHNOLOGIES

cea leti

POLITECNICO MILANO

EPFL
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

UNIVERSITY OF ILLINOIS
LEARNING & LABOR
CHARTERED 1867

lowRISC

ST
life.augmented

Target Platforms and SDK

1. Our PULP systems target mainly ASIC and FPGA designs

2. Virtual Platform

- C++ for fast execution and Python for configurations
- ISS to emulate our RISC-V
- Timing model: event based
- DMA, Memories, Peripherals models
- Time accuracy ~10%

3. OpenMP runtime support

released soon

Conclusions

1. *Thanks to all our extensions,* ***typical signal processing applications*** *like convolutions, gain up to 9.5x with optimized C code in speed. All of our extensions are supported by our released compiler.*

2. *On average the extended core has a small* ***increase in power and a big increase in performance*** *which leads to a* ***higher energy efficiency***

→ ***Energy efficiency increased*** *up to 4.6x with optimized C code with respect the general purpose implementation*

1. *You can download the HDL of our* ***RISC-V*** *core here http://www.pulp-platform.org/*

2. *We are going to release* ***PULP*** *in December, meanwhile you can use our core on PULPino!*
   See our release plans:
   http://www.pulp-platform.org/release-plan/

We are open to collaborate with you

We would like to receive your feedbacks

- *Help us to improve our work*
- *Please report bugs and fixes → We are looking for verification support*

Thank you!!!

QUESTIONS?

http://www.pulp-platform.org