ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Threads or Vectors? Evaluating SPMD and Vector Accelerators for Resource Constrained RISC-V Architectures

Amirhossein Kiamarzi, Samuele Colmi, Yvan Tortorella,
Angelo Garofalo, Davide Rossi, Giuseppe Tagliavini

PULP Platform
Open Source Hardware, the way it should be!

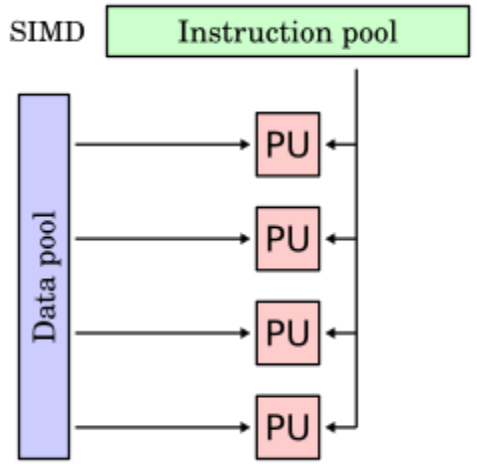
@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

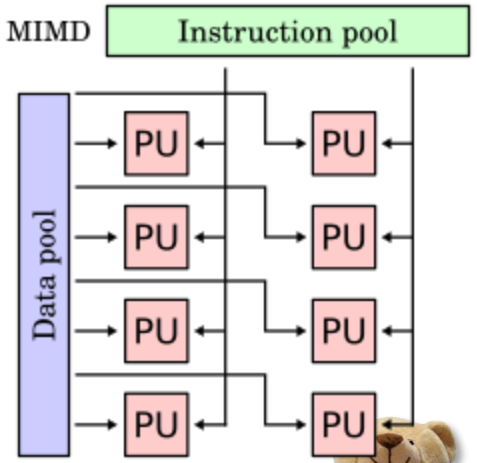
Outline

- **Introduction**
- Architecture Background
- Comparative Analysis of Computational Models
- Design Trade-offs in Performance and Efficiency
- Conclusion

Edge Computing Demands Smarter Architectures

- Edge devices are no longer just sensors. They now run compute-intensive workloads across multiple application domains including Linear Algebra · ML · DSP · Language Models
  - Parallelization patterns are not uniform:
    - Regular workloads → easy to exploit
    - Irregular workloads → hard to exploit
- "The question is no longer can we run this workload, but which execution paradigm runs it best."
  - maximum performance within limited power and memory budget
- Single-core MCUs cannot meet the computational demands of modern edge workloads
- Beyond single-core, two dominant embedded-friendly paradigms emerge:
  - Multi-core SPMD (MIMD) → embodied by PULP Cluster
  - SIMD → embodied by Spatz vector processor



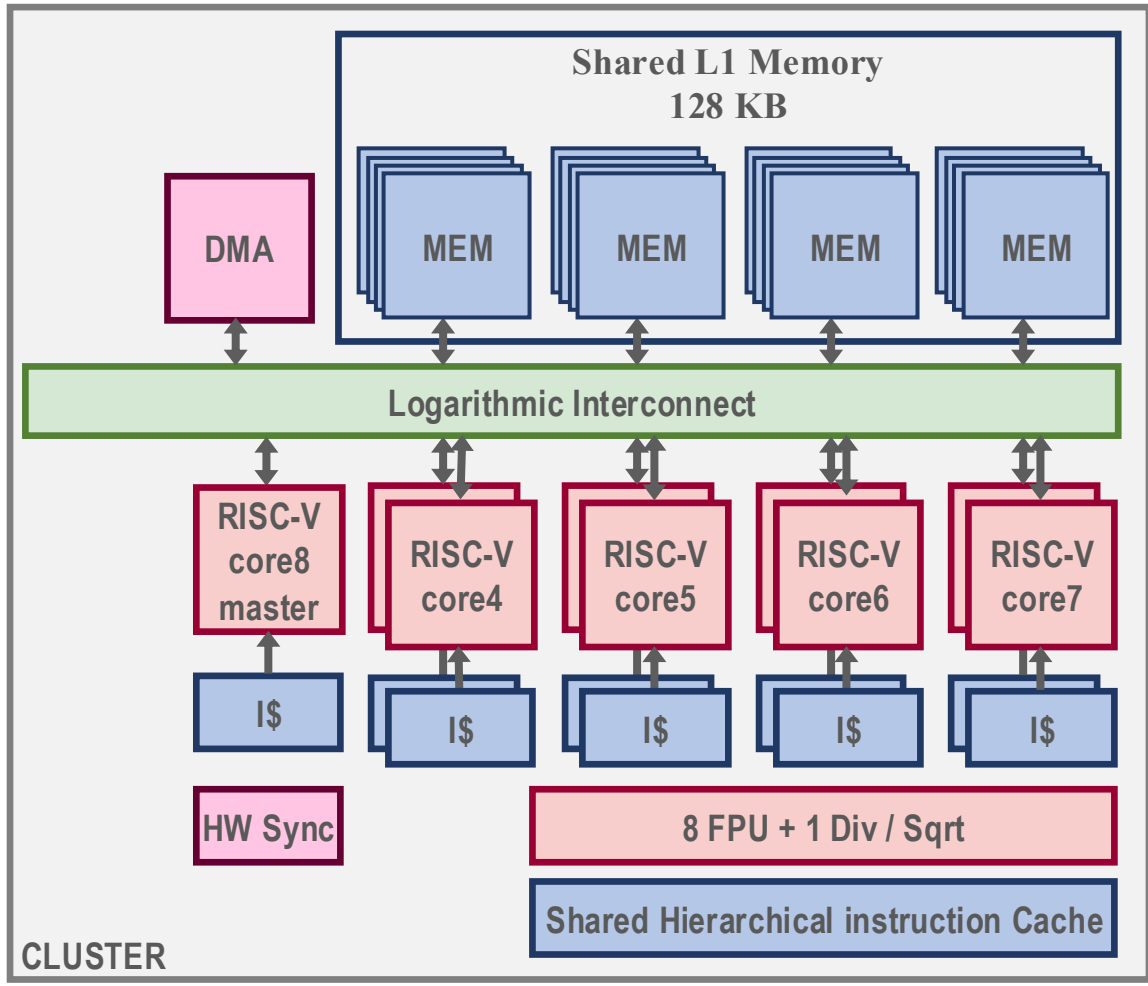


Outline

- Introduction
- **Architecture Background**
- Comparative Analysis of Computational Models
- Design Trade-offs in Performance and Efficiency
- Conclusion

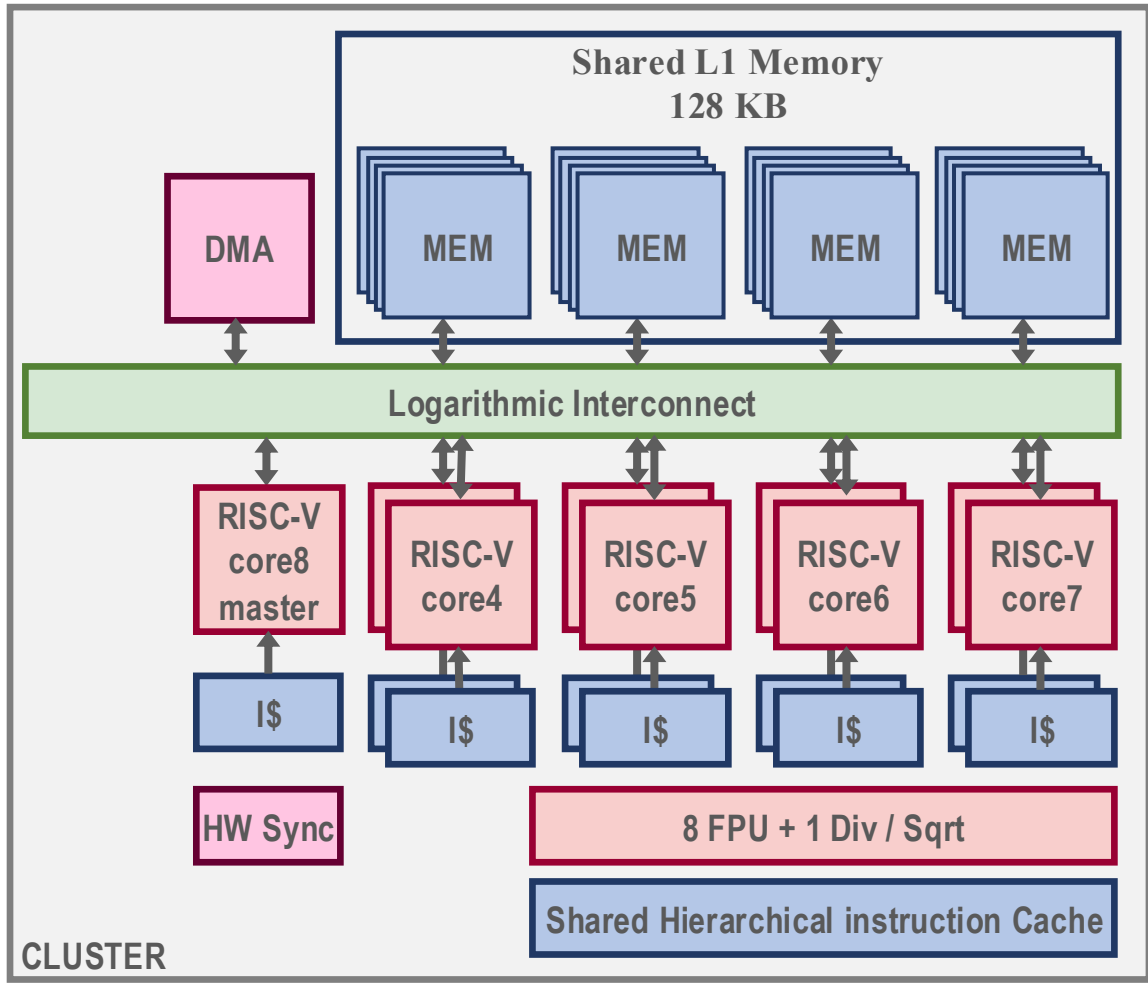
Pulp-cluster Architecture

- **Core Functions:**
  - **Data Processing Cluster**:
    - **8 Data Cores**: Perform high-performance data processing.
    - **1 Control Core**: Manages data transfer and cluster operations.
- **Logarithmic Interconnect:** Connects cores with single-cycle latency for rapid data access.



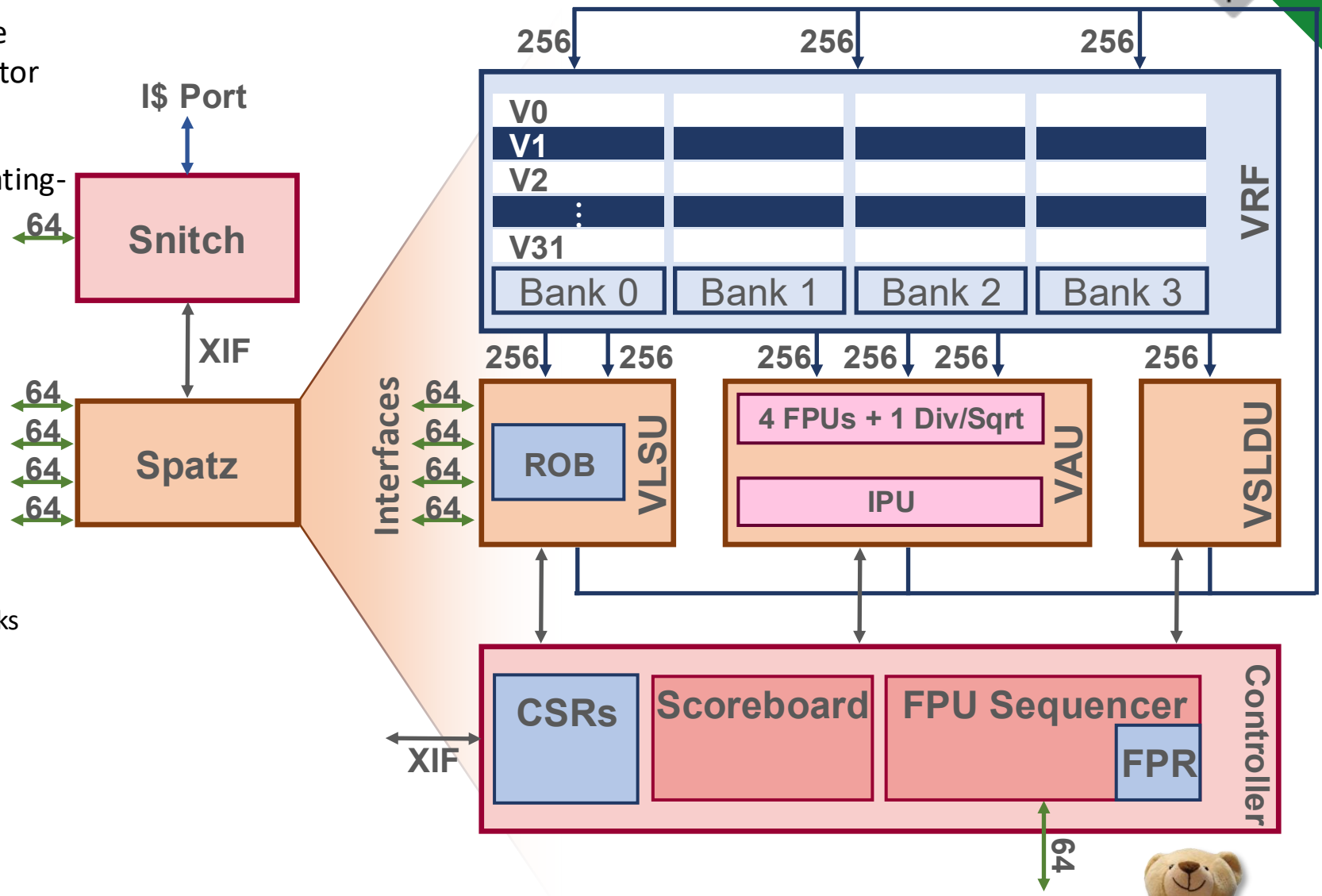
Pulp-cluster Architecture

- **Memory System:**
  - **L1 Scratchpad:** 128 kB, 16 banks, single-cycle access.
  - **Caches:** 512 B private and 4 KB shared instruction caches.
- **Parallelization and Efficiency:**
  - **Event Unit:** Supports threading, barriers, and power-saving by idling/resuming cores quickly.
  - **8 FPUs:** Single-cycle latency.
  - **Division/Sqrt Unit**: 7-14 cycles latency.



Spatz Architecture

- A compact 64-bit floating-point-capable vector processor based on RISC-V's Vector Extension.
- Supports 8, 16, 32, 64-bit integer & floating-point operations
- **Instruction Flow**
  - **Snitch**: pre-decodes vector instructions, dispatches to vector unit
  - **Spatz:** decodes instructions, execute, and acknowledges completion to Snitch
- **Vector Register File (VRF)**
  - VLEN = 512
  - Multi-banked, multi-ported 4× 3R1W banks
- **Functional Units**
  - **VLSU** – Vector Load/Store Unit
  - **VAU** – Vector Arithmetic Unit
  - **VSLDU** – Vector Slide Unit

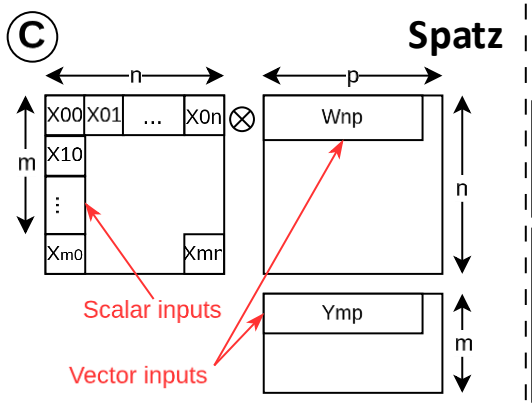


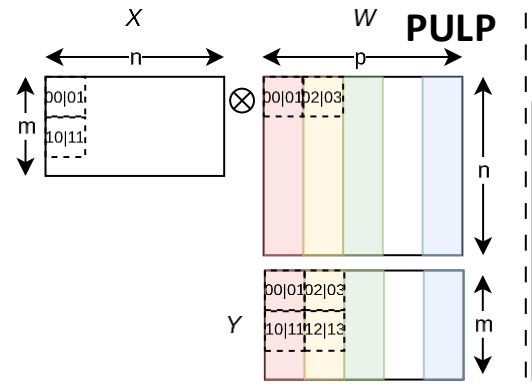
Outline

- Introduction
- Architecture Background
- **Comparative Analysis of Computational Models**
- Design Trade-offs in Performance and Efficiency
- Conclusion

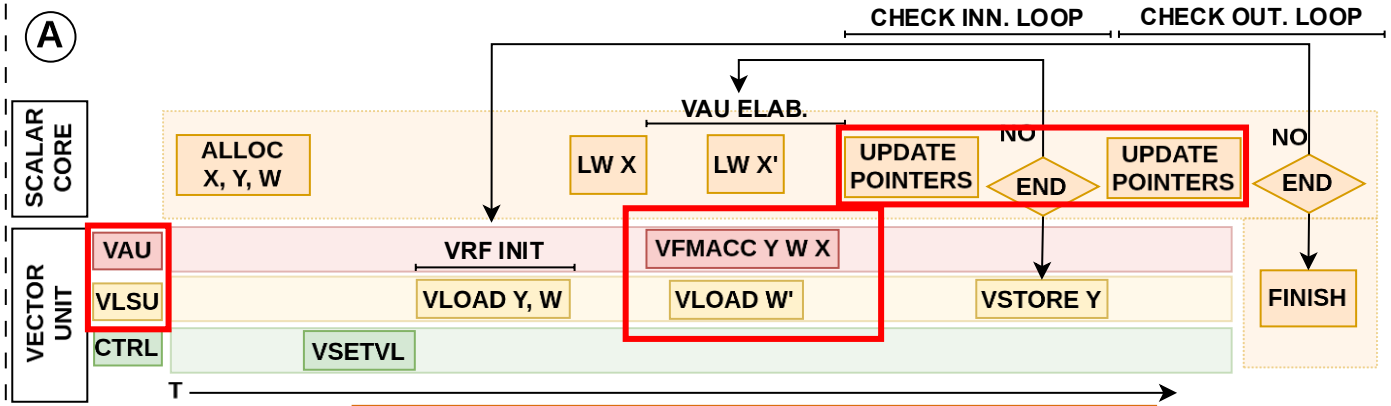
GEMM Execution Flow

Data tile organization

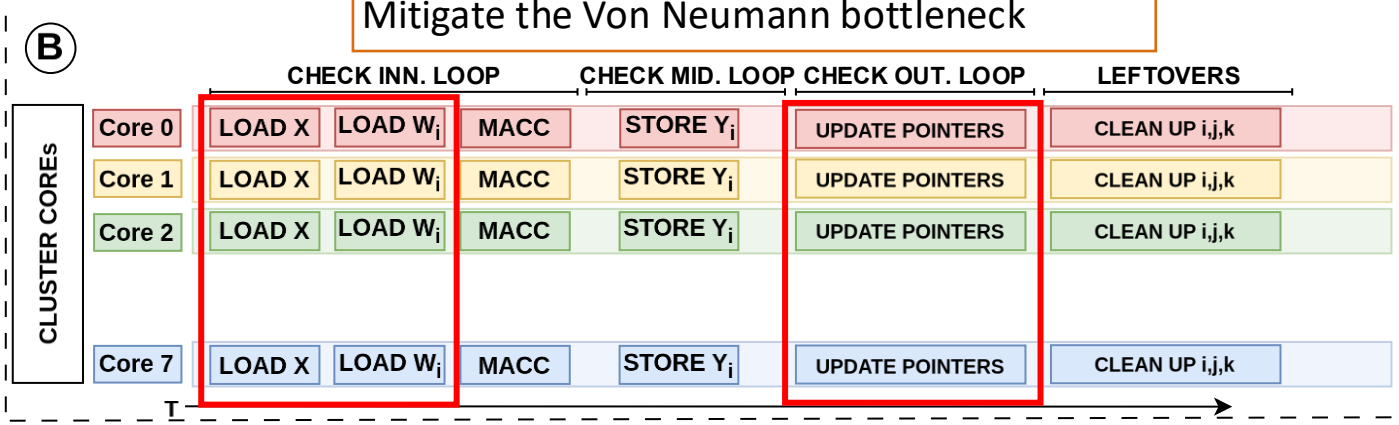




Temporal execution flow



Mitigate the Von Neumann bottleneck



Pseudo code snippets

```
alloc(X,Y,W);
while(p < p_end) {
  vsetvl(p_end - p);
  while(m < m_end) {
    vload(Ymp);
    while(n < n_end) {
      lw(Xnm);
      vload(Wnp);
      vfmacc(Ymp,Xnm,Wnp);
      n++;}
    vstore(Ymp);
    m++;}
p += max(vlmax,(p_end-p));}
```

```
alloc(X,Y,W);
for(k=c_id;k<P/2;k+=#Cores{
 for(i=0;i<M/2;i++){
  for (j = 0; j < N; j++){
      load x0 [i * 2 * N + (j)]
      load x1 [i * 2 * N + N + (j)]
      load w0[j * P + (k * 2)]
      load w1[j * P + (k * 2 + 1)]
      macc(sum0-3,x0-1,w0-1)
  }
  store(sum0-3)
}}
clean up for j,i,k
```

Von Neumann bottleneck

Performance analysis of different kernels

- Code pattern is the naïve behavior of the algorithm not the implementation.
- Bold values show the minimum per kernel.
- Kernels with FPU utilization below 10% and above 40% for PULP Cluster, and above 80% for Spatz, are highlighted.
- IMA = irregular memory access
- LP = low parallelization
- CD = control-dominant
- DNL = dependent nested loop
- RED = reduction
- CI = compute intensive
- MI = memory-intensive
- DCF = divergent control flow
- NPI = non-pipelined instruction (e.g., div/sqrt)

		PULP CLUSTER		Spatz			
Domain	Method	Tot. cycles[#]	FPU Util[%]	Tot. cycles[#]	FPU Util[%]	Code Pattern	Speed-up
LA	CHOLESKY	**38026**	15.77	166039	3.61	IMA+LP	4.37×
	LU DECOMP	**52784**	7.32	193161	2.00	IMA+LP+CD	3.66×
	LU SOLVE	**15050**	3.61	16503	3.29	DNL+CD+RED	1.1×
	GEMV	**6542**	31.80	16616	12.52	CI+RED	2.54×
	GEMVT	7140	29.13	**2537**	82.00	CI+RED	2.81×
	GEMM	70066	46.77	**33605**	97.51	CI	2.08×
	SVD	**507413**	31.74	2197742	7.33	RED	4.33×
DSP	FIR	34025	22.60	**8272**	92.94	CI+RED	4.11×
	FFT	6956	41.40	**3428**	84.01	CI+IMA	2.03×
	DWT	35451	27.60	**11343**	86.26	CI+RED	3.13×
ML	KMEANS	**114216**	29.70	478862	7.08	LP+NPI+DCF	4.19×
	CONV3	6302	16.07	**2136**	47.40	CD+RED	2.95×
	SVM EXP BILL	70250	32.78	**31208**	73.78	CI+CD+RED	2.25×
	SVM EXP CANC	381814	34.58	**237274**	55.64	CI+CD+RED	1.61×
	RANDOM FOREST	**13608**	0	80398	0	CD	5.91×
LLM	ENCODER	209152	11.75	**104960**	23.41	MI	1.99×
	GELU	3038592	48.53	**2927424**	50.37	CI+NPI	1.04×
	RESIDUAL	177984	13.81	**108432**	22.66	MI	1.64×
	SOFTMAX	**58724352**	16.43	123355136	7.82	NPI+RED	2.1×
	MATMUL	125127936	45.25	**63840960**	88.69	CI	1.96×
	ATTENTION	**19309920**	13.20	40682400	6.27	DNL	2.11×
	LAYER NORM	**933888**	21.07	1994496	9.86	RED	2.14×

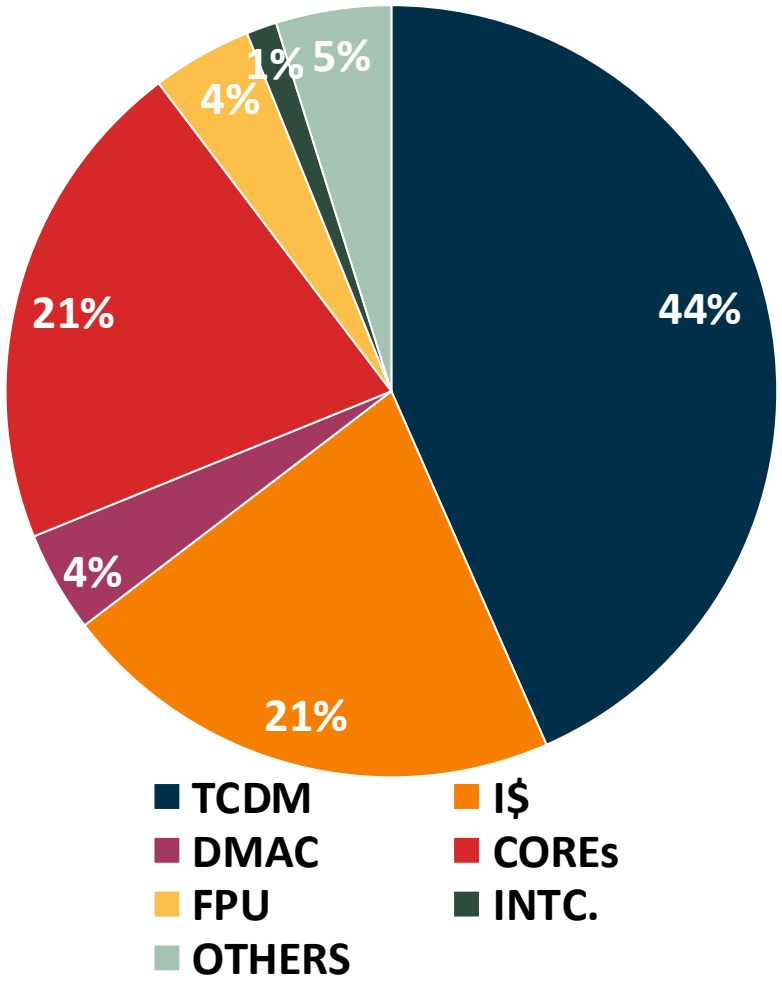
Outline

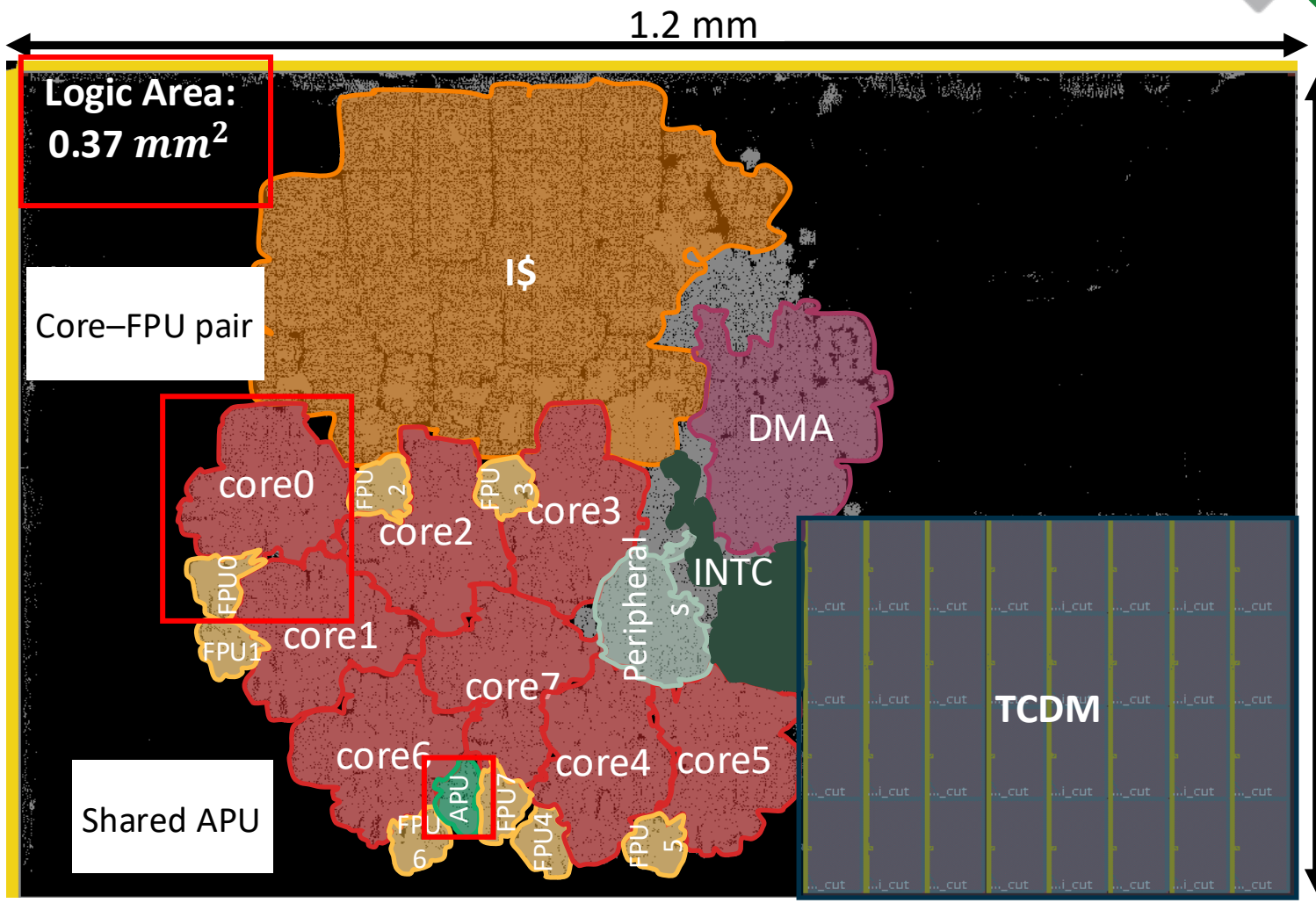
- Introduction
- Architecture Background
- Comparative Analysis of Computational Models
- **Design Trade-offs in Performance and Efficiency**
- Conclusion

Experimental Setup

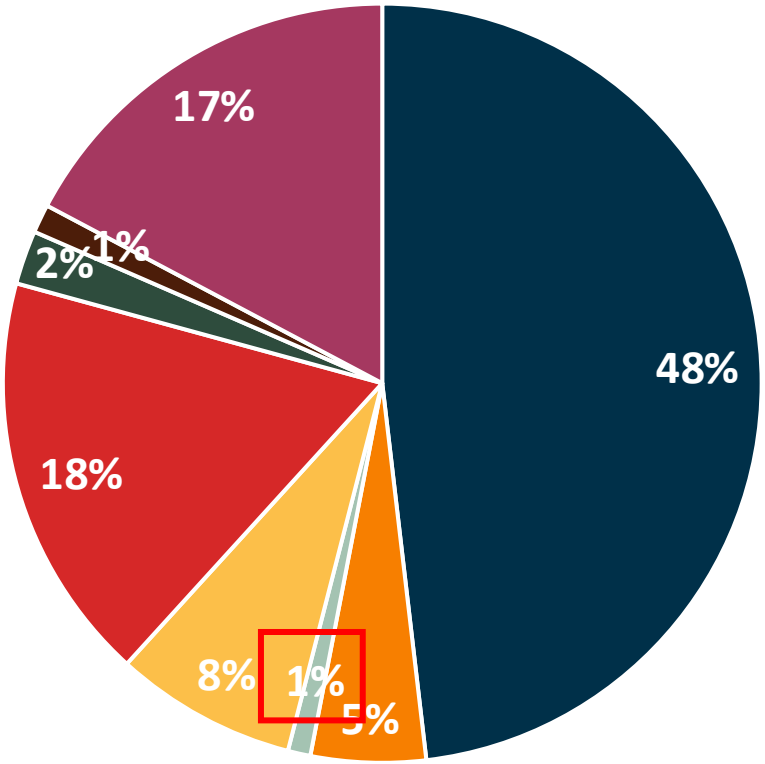
- We combine functional performance with physical implementation metrics, analyzing execution time, utilization, and power to derive insights into performance, energy efficiency, and area efficiency (PPA).
- Functional Validation:
  RTL and post-layout simulations using Siemens QuestaSim v2023.4
- **Physical Implementation:**
  - **Technology:** GlobalFoundries 12nm (GF12)
  - **Logic Synthesis:** Synopsys Design Compiler v2020.09
  - **Place & Route:** Cadence Innovus v21.17
  - **Power Estimation:** Synopsys PrimeTime v2022.03

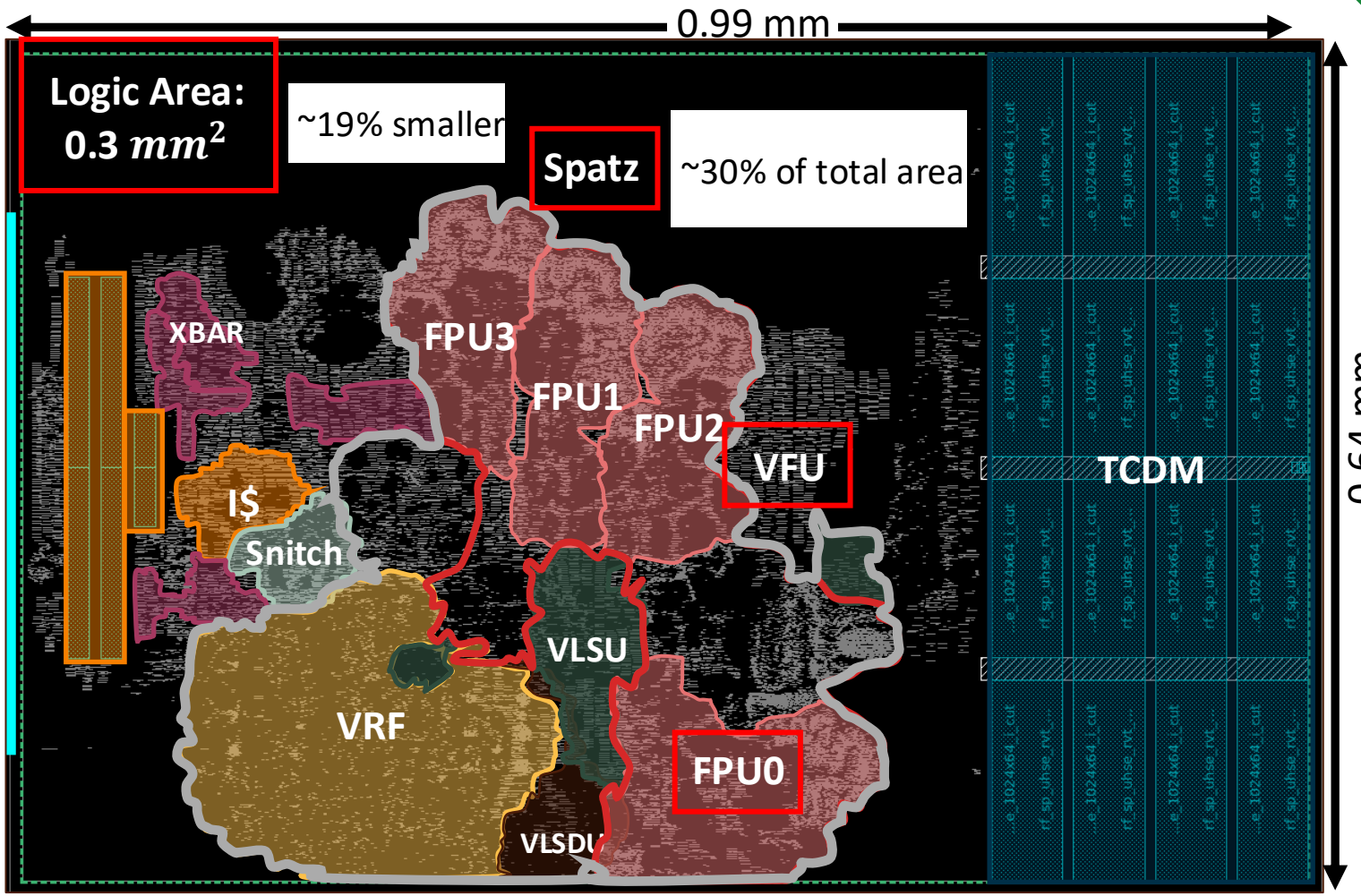
Pulp Cluster Chip Layout



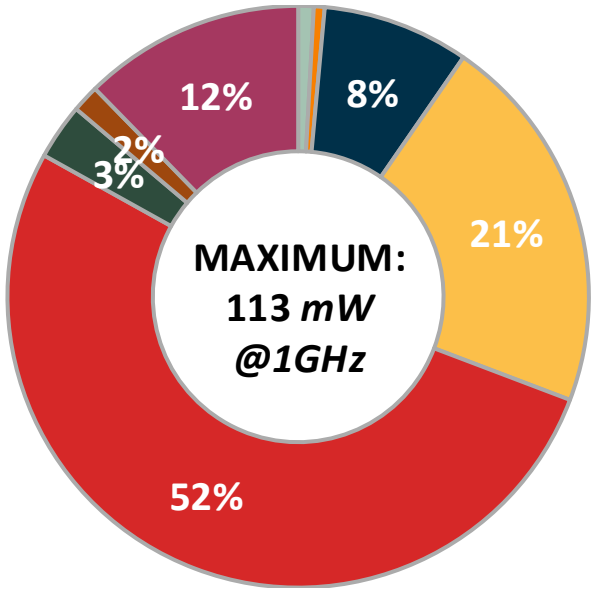


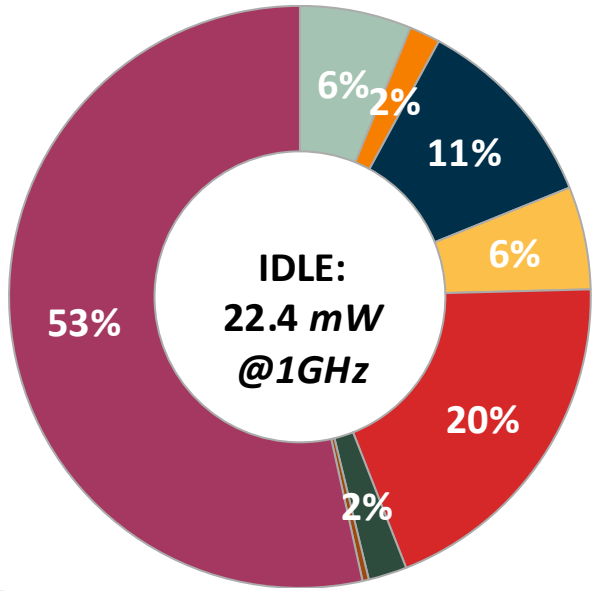
Spatz Chip Layout





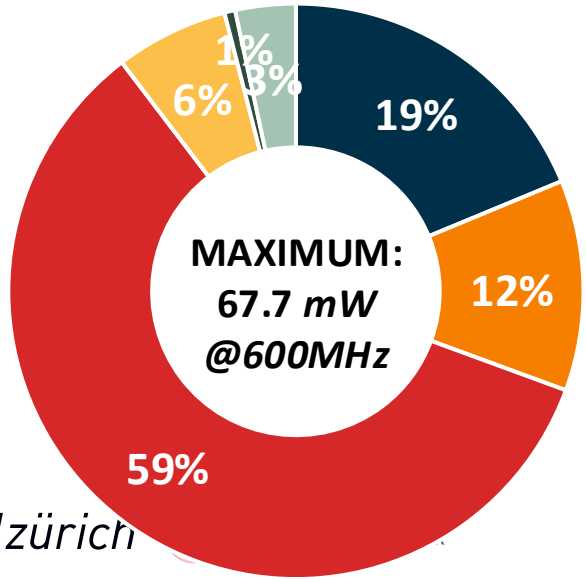
Post-layout power breakdown

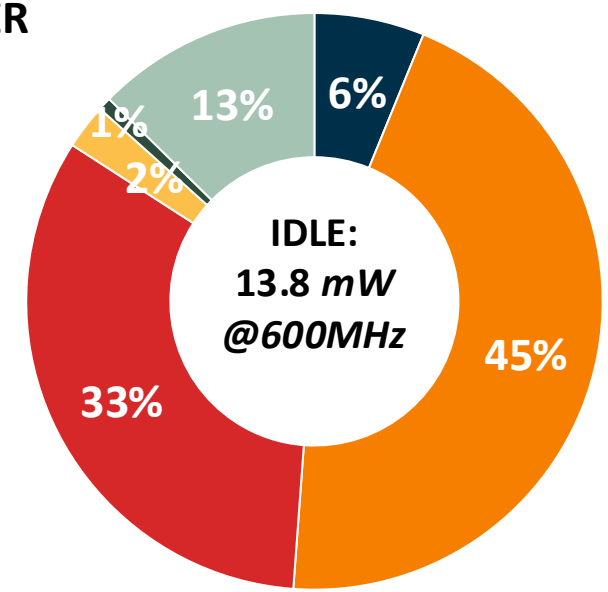




SPATZ

- Snitch
- I$
- TCDM
- VRF
- VFU
- VLSU
- VSLDU
- OTHERS

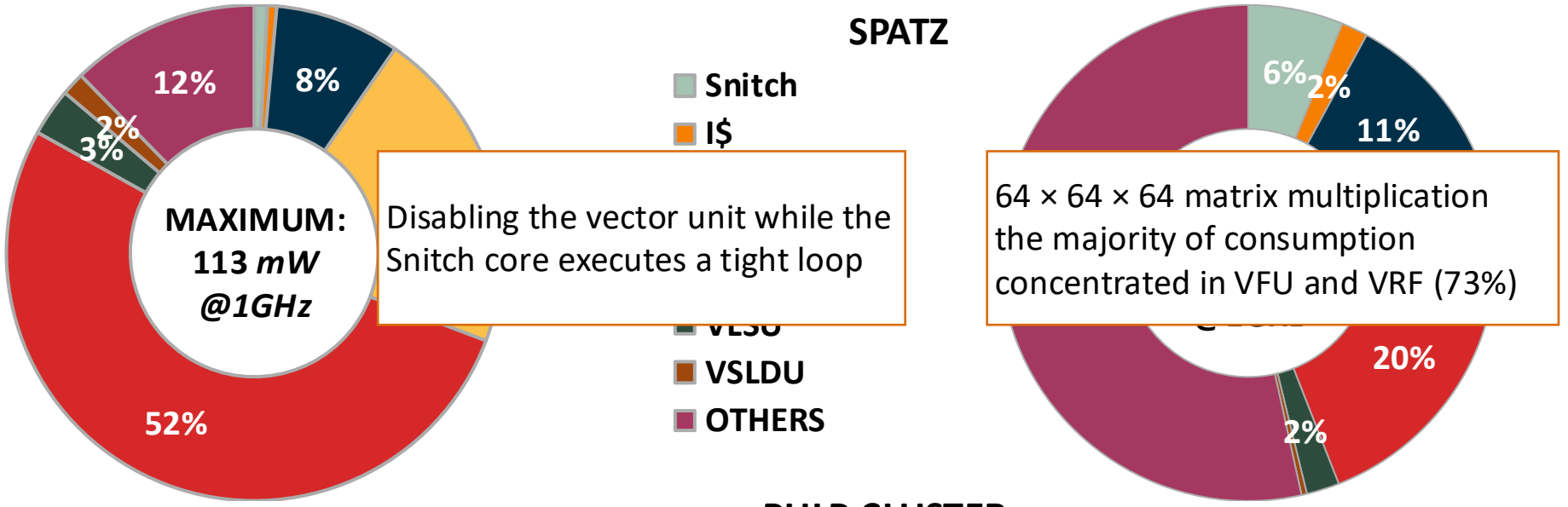


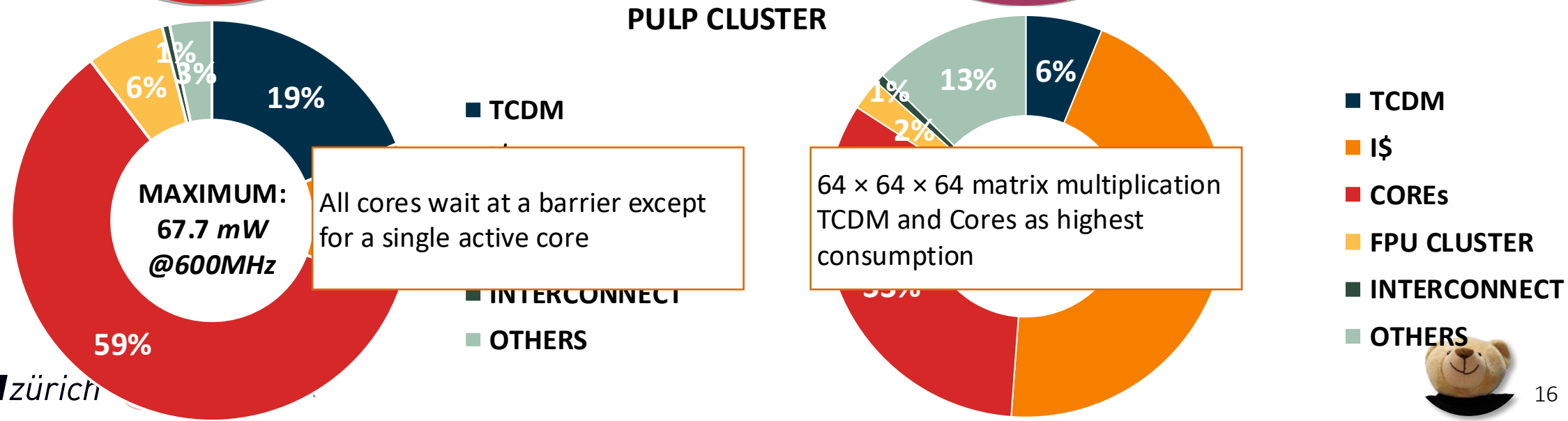


PULP CLUSTER

- TCDM
- I$
- CORES
- FPU CLUSTER
- INTERCONNECT
- OTHERS

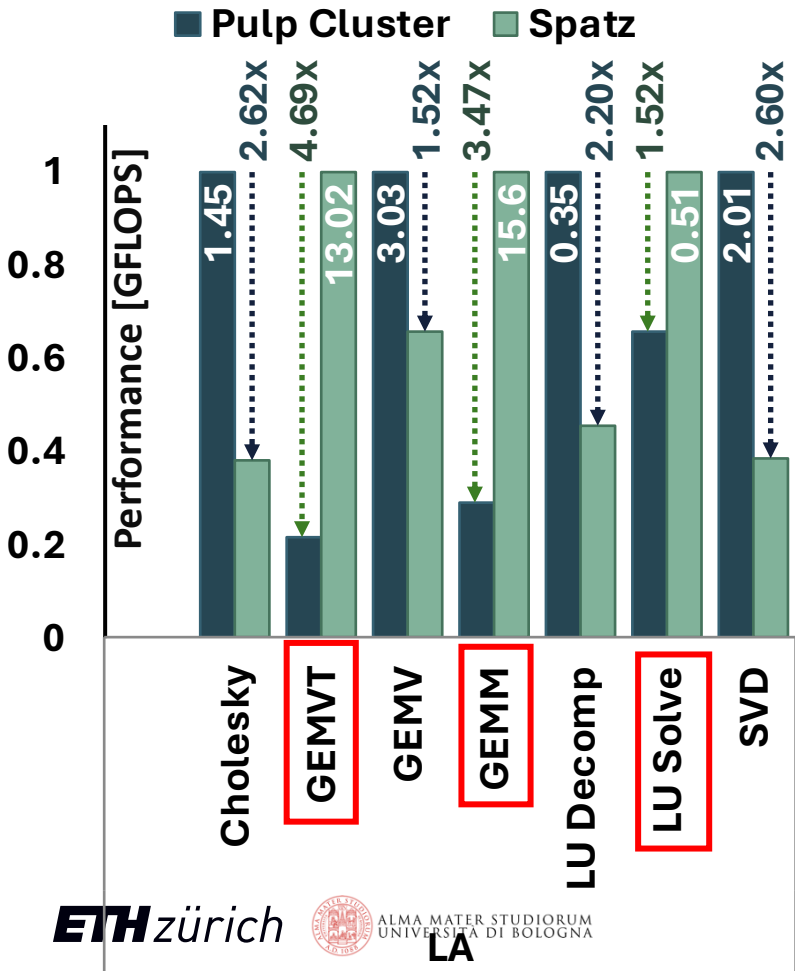
Post-layout power breakdown

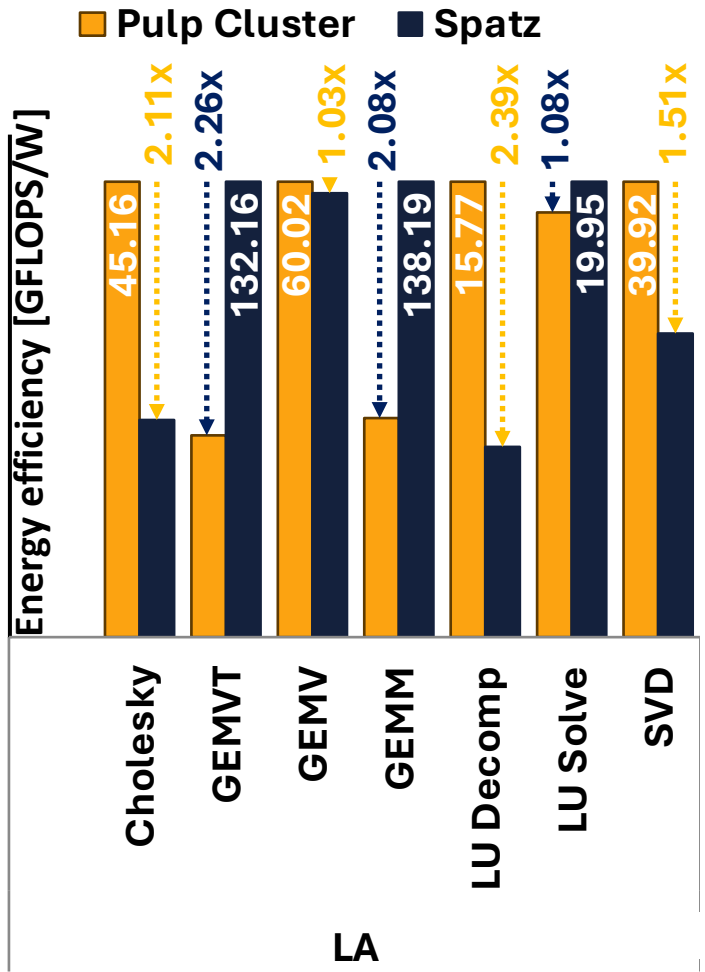


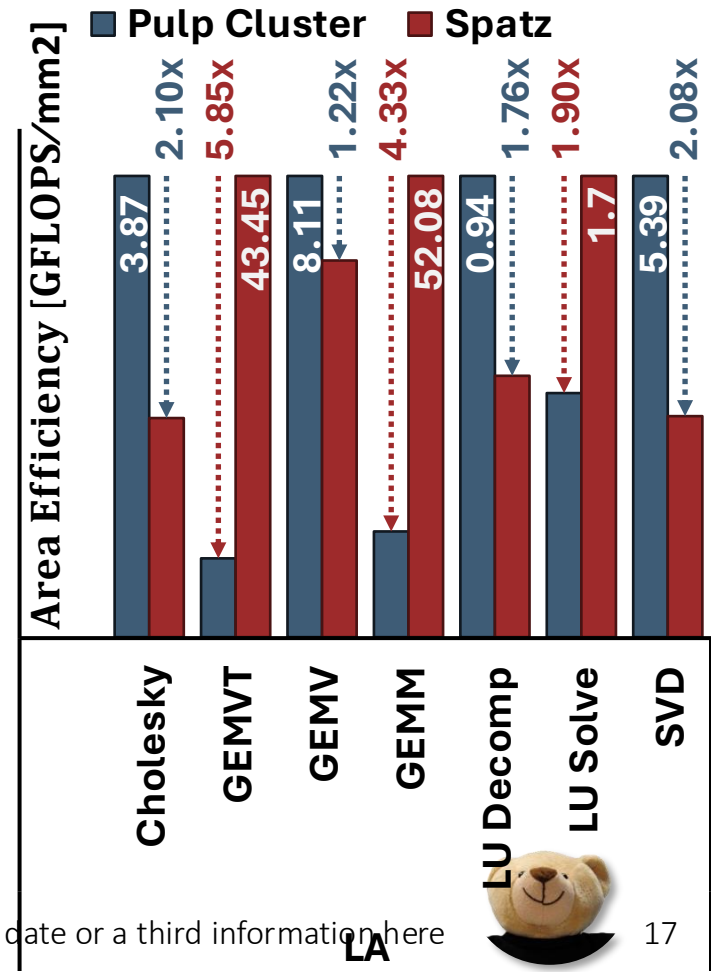


Post-layout PPA comparison (LA)

- Irregular memory access (e.g. Cholesky) and reduction-heavy computation (e.g. SVD) kernels perform better on the PULP Cluster.
- Compute-intensive kernels with regular memory access perform better on Spatz.
- GEMV and GEMVT are both compute-intensive and reduction-heavy, but different memory access layouts make each one favor a different platform.

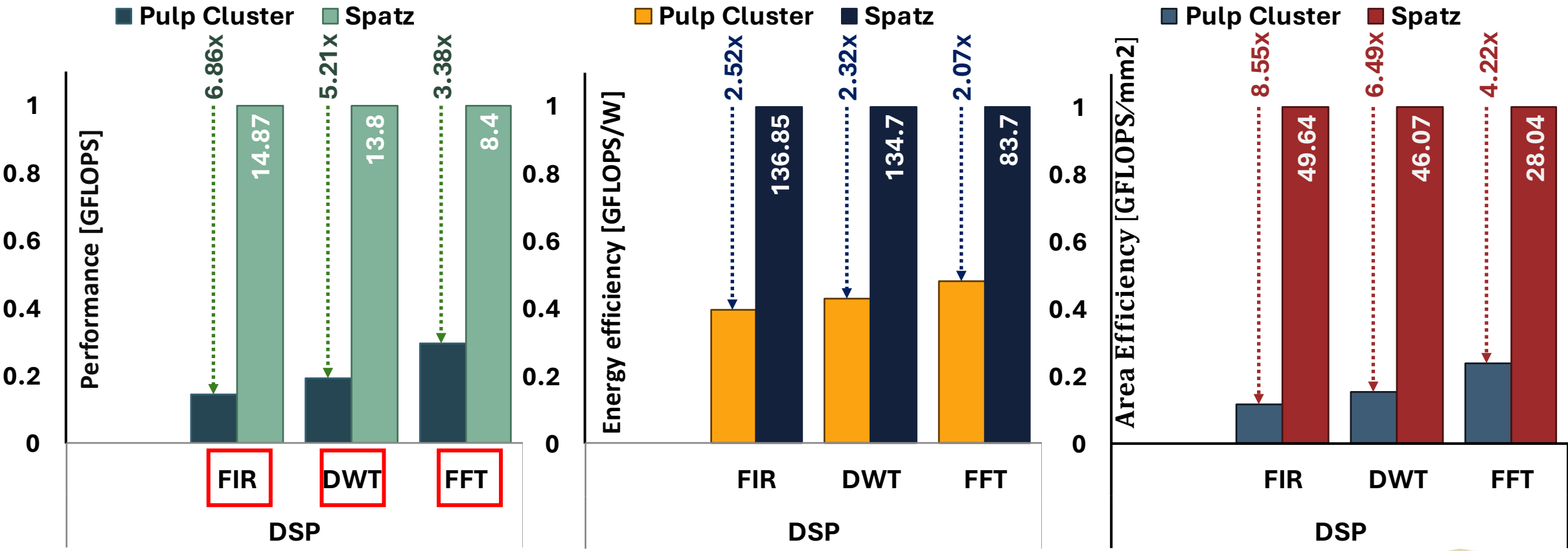






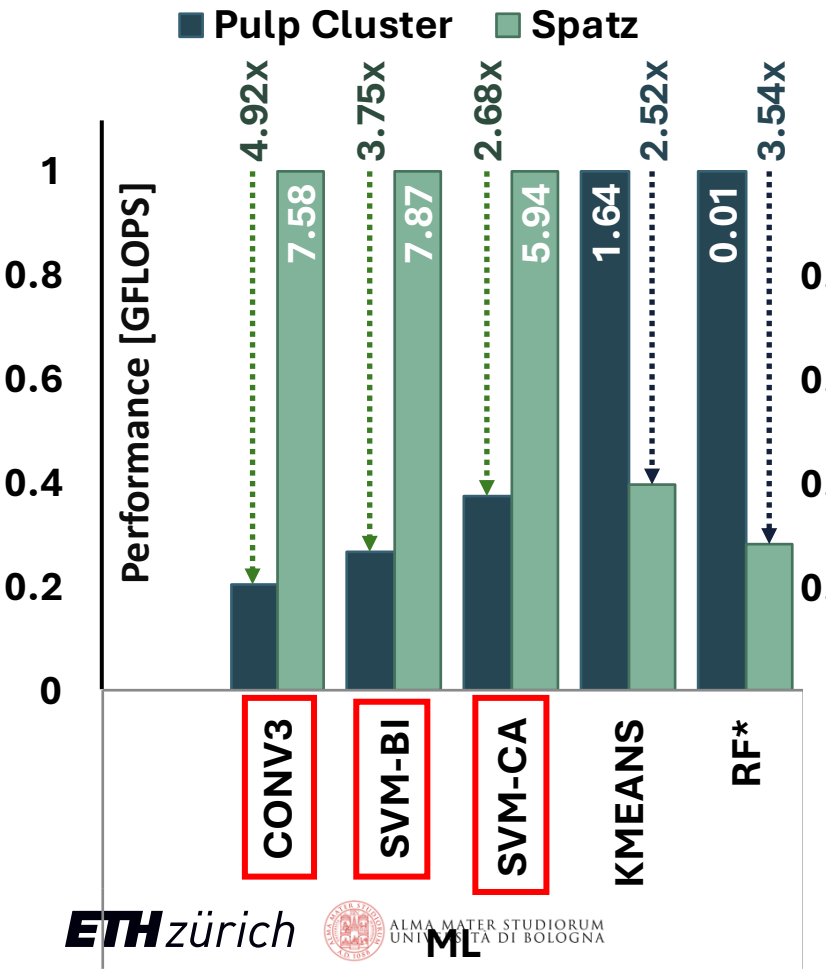
Post-layout PPA comparison (DSP)

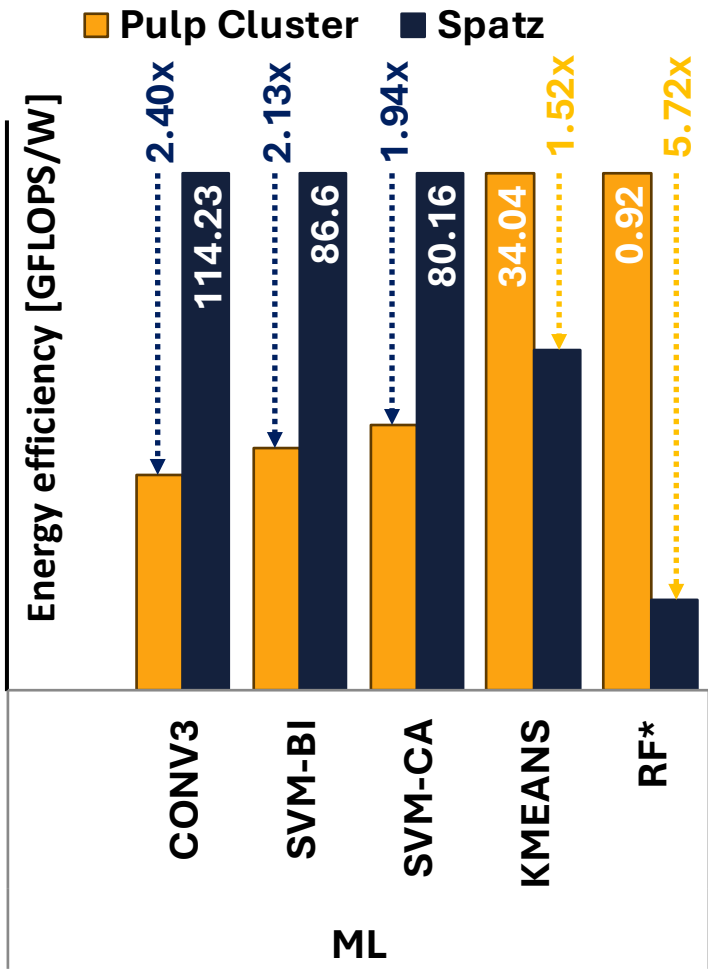
- Irregular memory access latency can be hidden when followed by large computation and high data reuse, as in FFT, making vector execution still competitive in such cases.

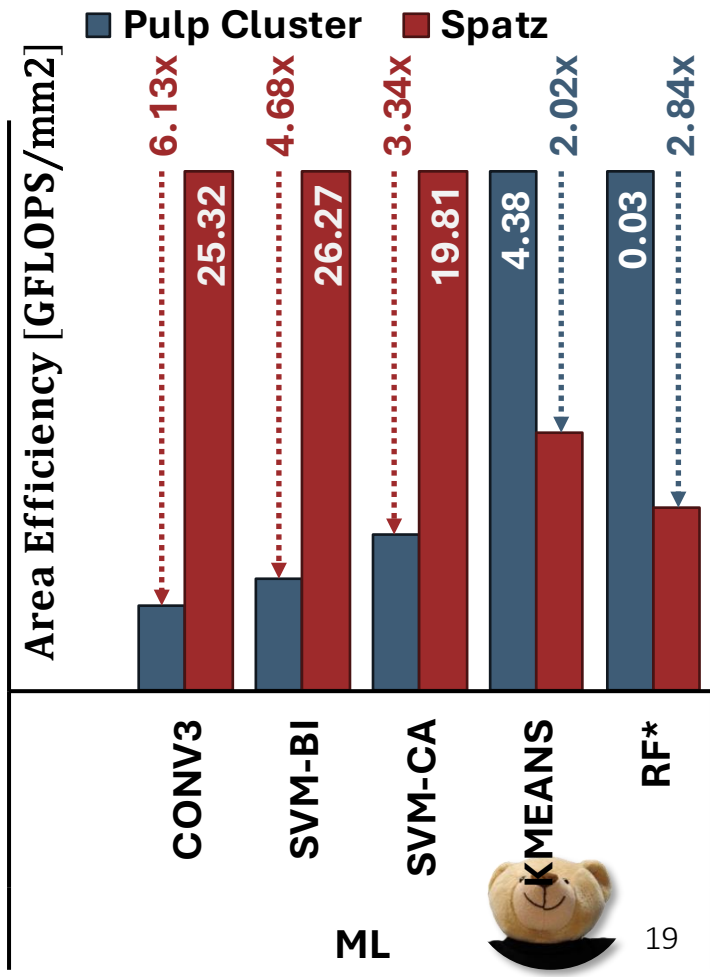


Post-layout PPA comparison (ML)

- All values are in GFLOPS, except RF, which is reported in GOPS.
- Kernels with divergent control flow requiring element-wise decisions perform better on the PULP Cluster.

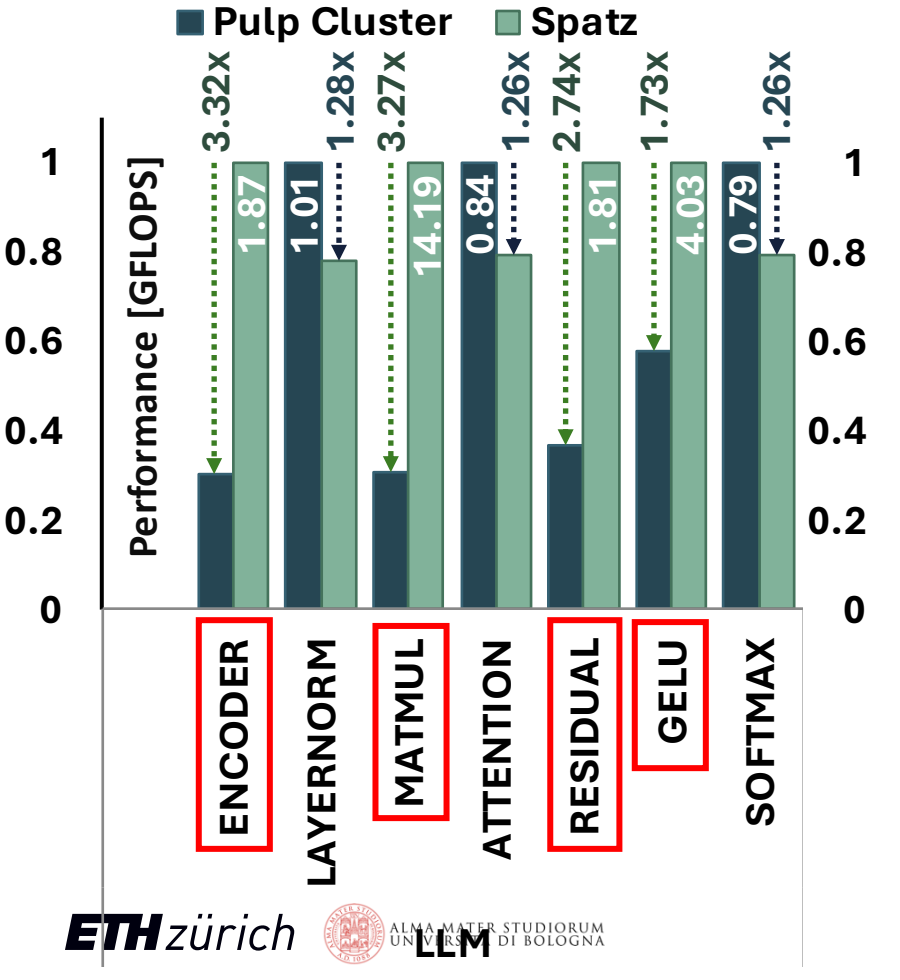


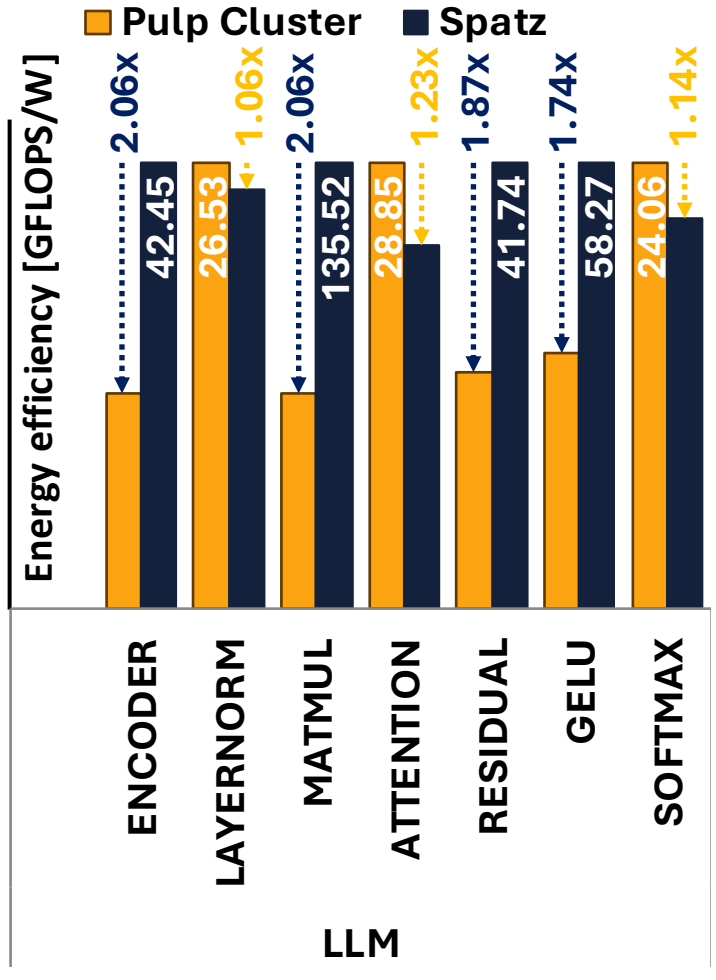


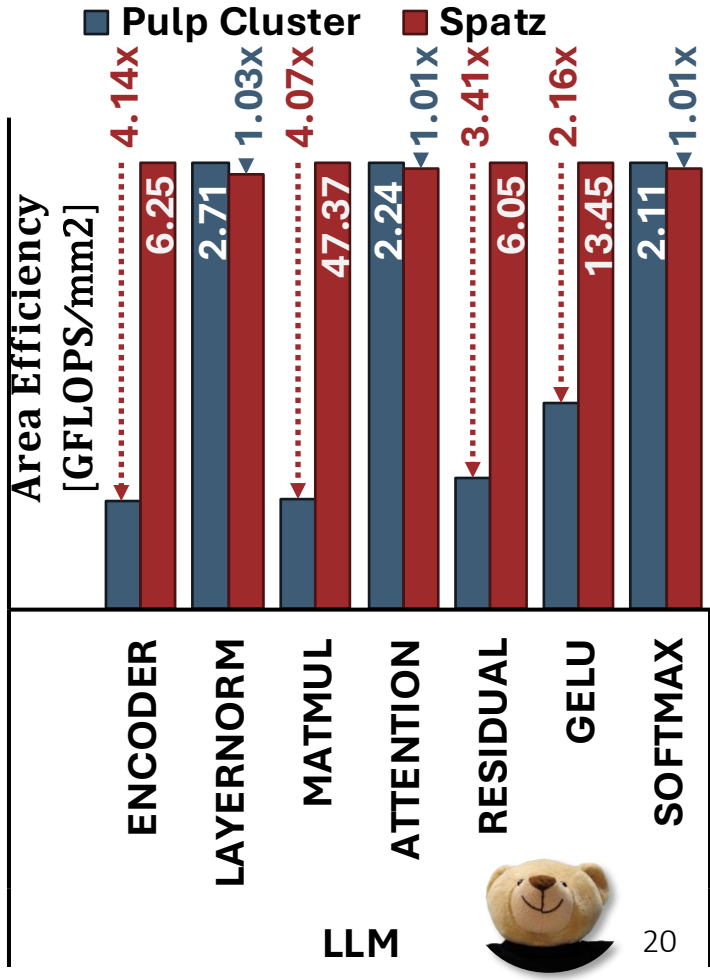


Post-layout PPA comparison (LLM)

- Since memory access and computation are overlapped, memory-intensive kernels such as Encoder and Residual perform better on Spatz.
- Kernels with dependent nested loop patterns perform better on the PULP Cluster, as the number of operations reduces at each iteration leaving the FPU underutilized on Spatz.

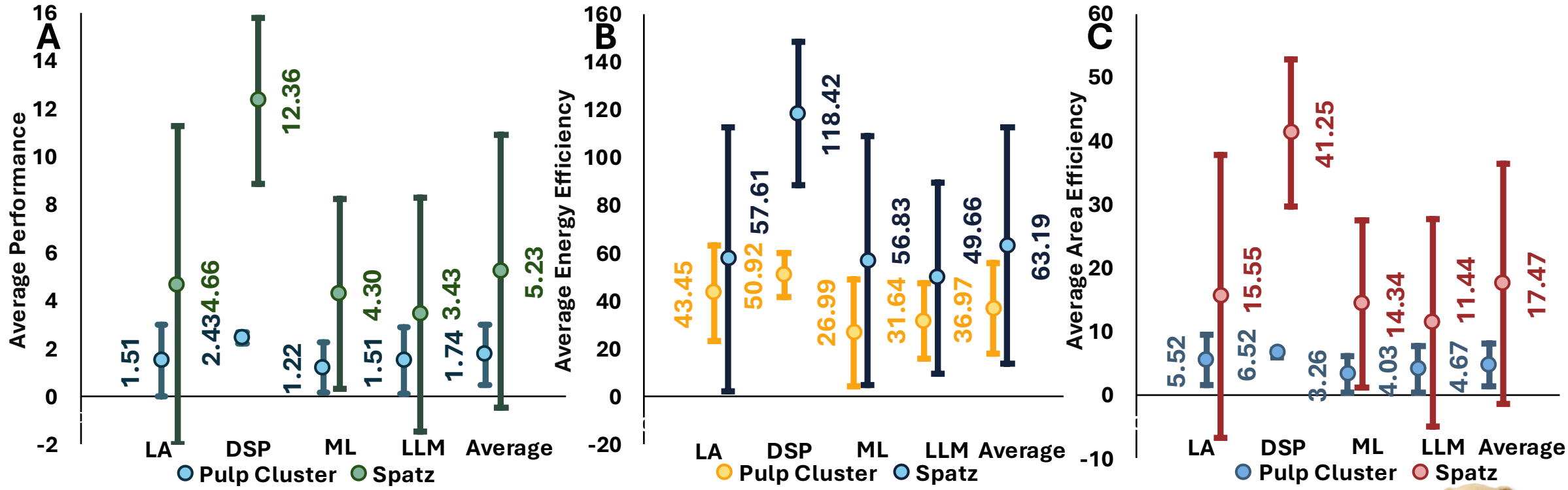






Category-level aggregation of PPA

- PULP cluster demonstrates more uniform behavior.
- Spatz shows greater variability than the PULP cluster (stronger dependence on application characteristics).
- On average, Spatz achieves **3.6×** higher performance, **4.86×** higher energy efficiency, and **2.97×** higher area efficiency.



Outline

- Introduction
- Architecture Background
- Comparative Analysis of Computational Models
- Design Trade-offs in Performance and Efficiency
- **Conclusion**

Threads or Vectors? It Depends, And That's the Point

- Spatz (Vector SIMD Processor)
  - Best for **regular**, **compute-intensive** kernels with predictable memory access
  - Achieves high utilization
  - Up to **8.22×** (performance), **4.95×** (energy), **10.26×** (area) improvement
- PULP (Multi-core SPMD)
  - Best for **irregular**, **control-heavy** workloads
  - Handles **irregular memory access**, **reductions**, and **dependent nested loop** well
  - Up to **2.95×** (performance), **3.84×** (energy), **2.37×** (area) improvement

- PULP is more flexible
  - Scales with core count and adapts to workload structure
  - Effective for **small** GEMMs, reductions, non-linear operations
- Spatz has higher peak efficiency but more sensitive to workload characteristics
  - This sensitivity make it a **high-reward, high-risk** choice as a standalone System on-Chip accelerator

- **Spatz provides higher peak efficiency, while PULP offers better flexibility and robustness across diverse workloads.**

Thank you!