ETH zürich

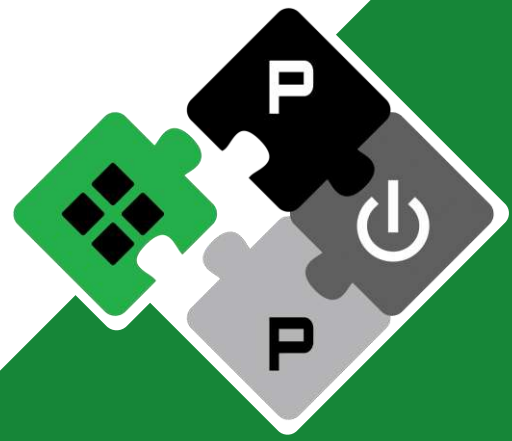
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Diving into MemPool:
Scaling the Shared-Memory Cluster to 256 Cores

Integrated Systems Laboratory (ETH Zürich)

Samuel Riedel sriedel@iis.ee.ethz.ch

and the **PULP team**

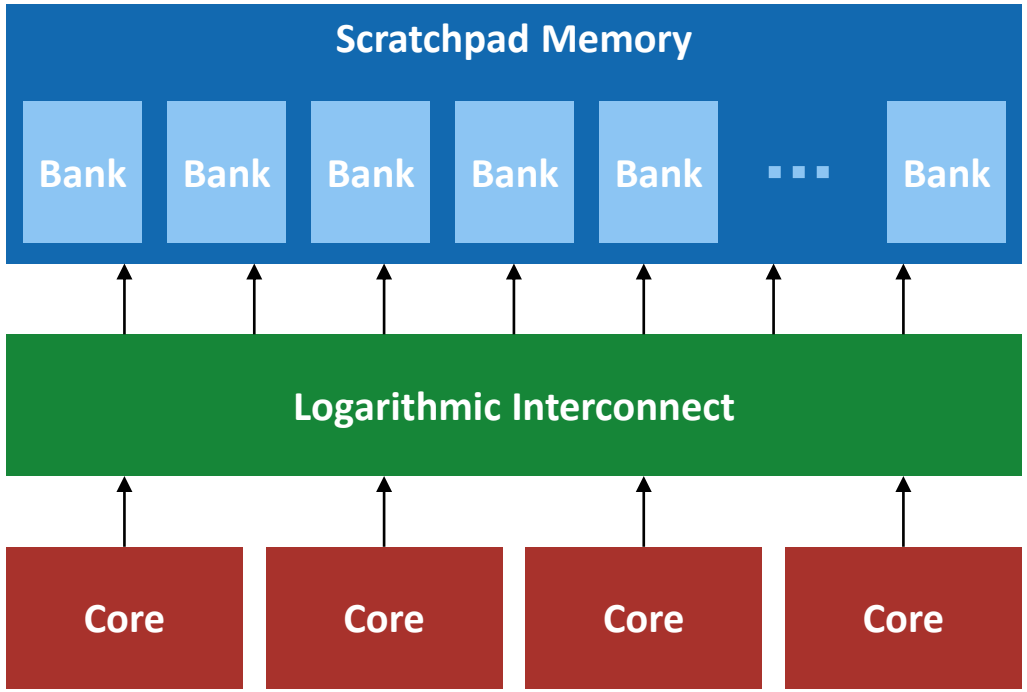


PULP Platform
Open Source Hardware, the way it should be!

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

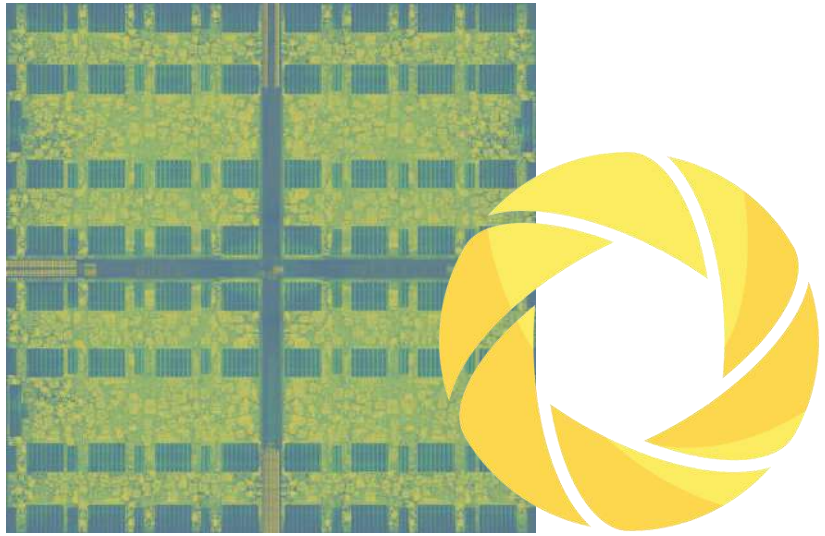
The Compute Cluster

- Widely used building block
  - Parallel and flexible
- Low latency access SPM
  - Multi-banked architecture
  - Fast logarithmic interconnect
- Fast synchronization
  - Atomics
- Efficient cores
  - Hide SPM “residual” latency
  - Expressive, domain-specific ISA extensions



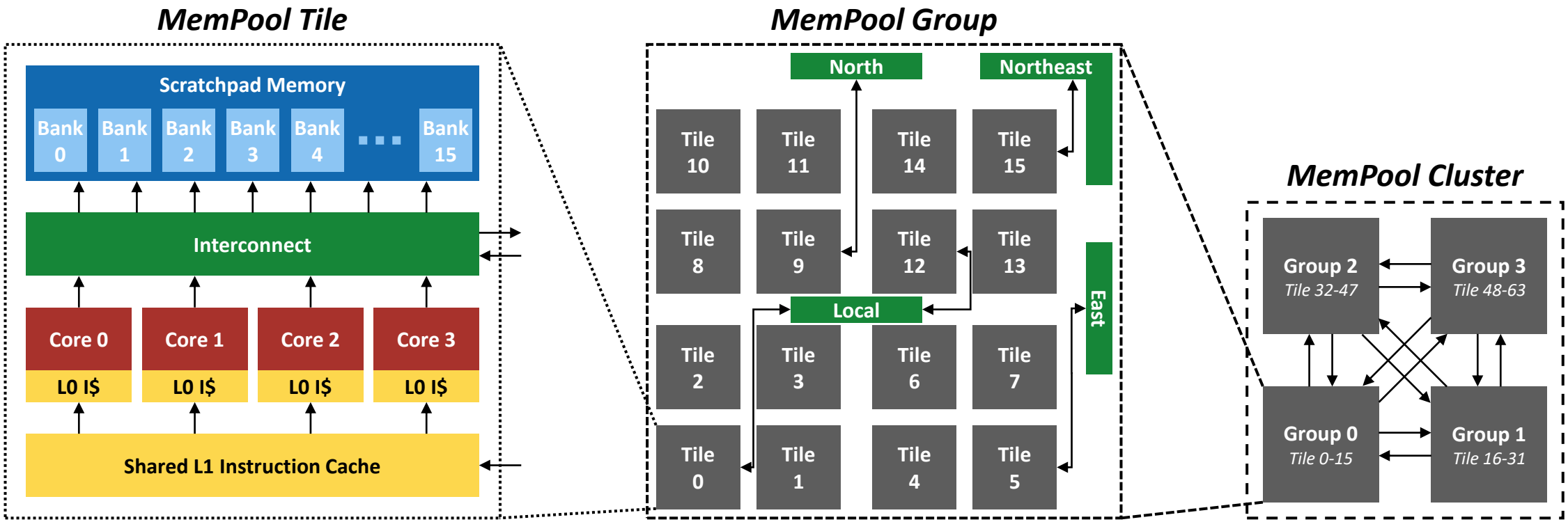
Can we scale the cluster?

- **Why?**
  - Easier to program (data-parallel, functional pipeline…)
  - Smaller data partitioning and communication overhead
  - Better global latency tolerance with large L1
- **MemPool:** A Scalable Manycore Architecture with Low-Latency Shared L1 Memory
  - 256+ cores
  - 1+ MiB of shared L1 data memory
  - ≤ 5 cycles latency (without contention)
- Physical-aware design
  - WC Frequency > 500 MHz
  - Targeting iso-frequency with a small cluster

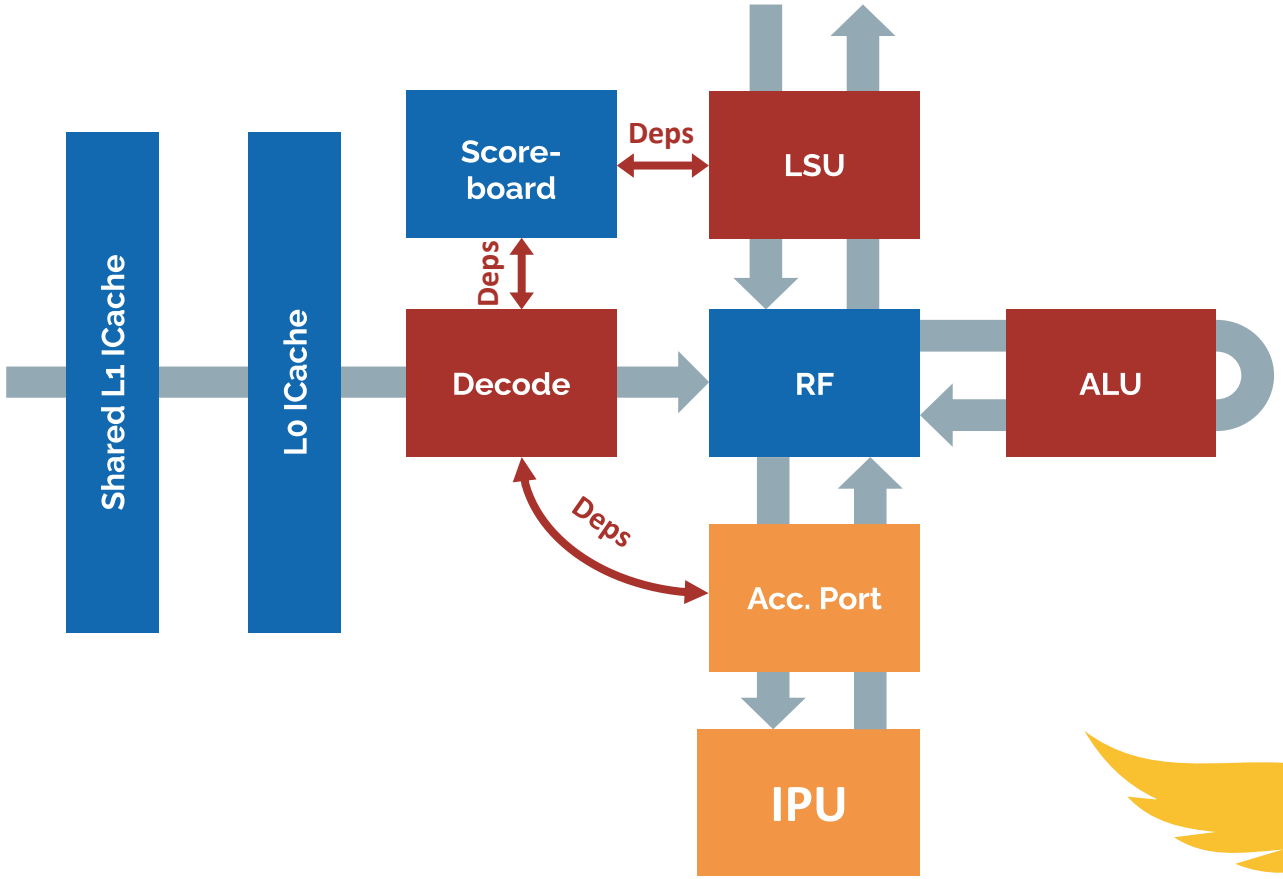


MemPool's Hierarchy

- **Tile**
  - 4 32-bit RISC-V cores
  - 16 SPM banks
  - **Single-cycle** memory access
- **Group**
  - 64 cores
  - 256 SPM banks
  - **3 cycles** latency
- **Cluster**
  - 256 cores
  - 1024 SPM banks → **1 MiB**
  - **5 cycles** latency



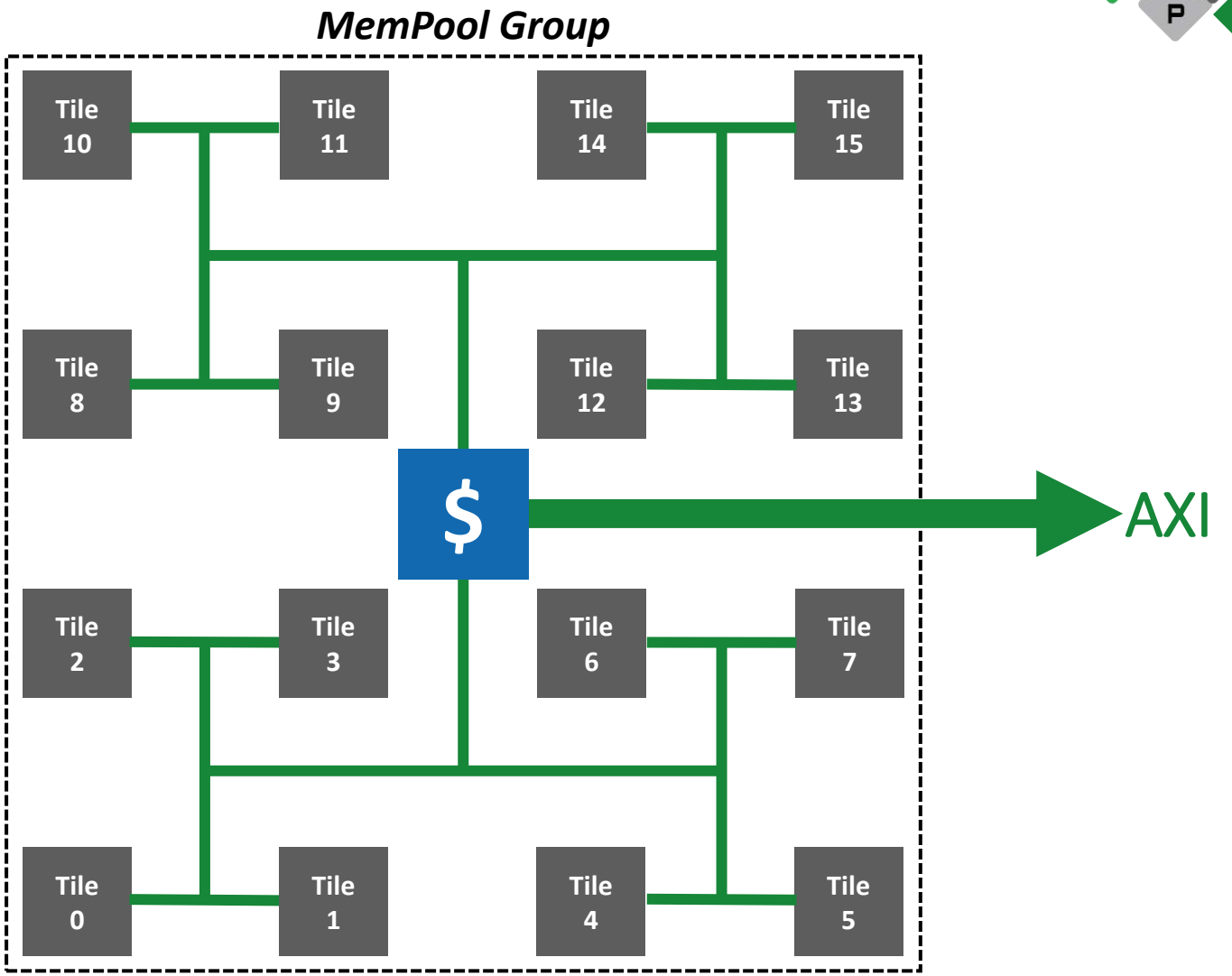
Hiding Latencies with Snitch



- Latency-tolerant → Scoreboard
  - Extended LSU for out-of-order retiring
- Extensible → IPU
  - Supports the *Xpulpimg* extension

How to move data in and out?

- Hierarchical AXI interconnect
  - Tree-like structure
  - Read-only caches at nodes

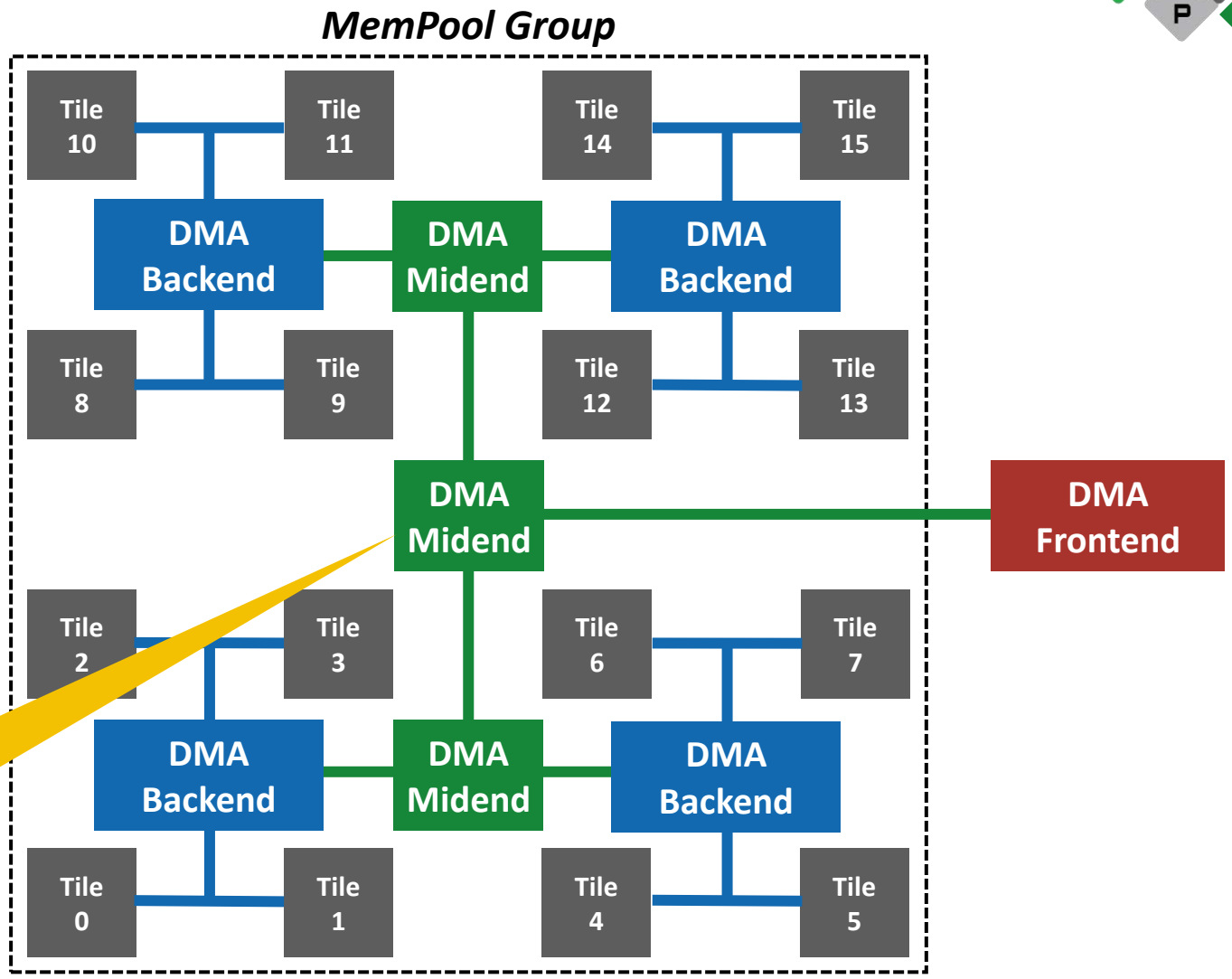


How to move data in and out?

- Hierarchical AXI interconnect
  - Tree-like structure
  - Read-only caches at nodes
- Distributed DMA
  - Single frontend
  - Tree of midends
  - Multiple backends

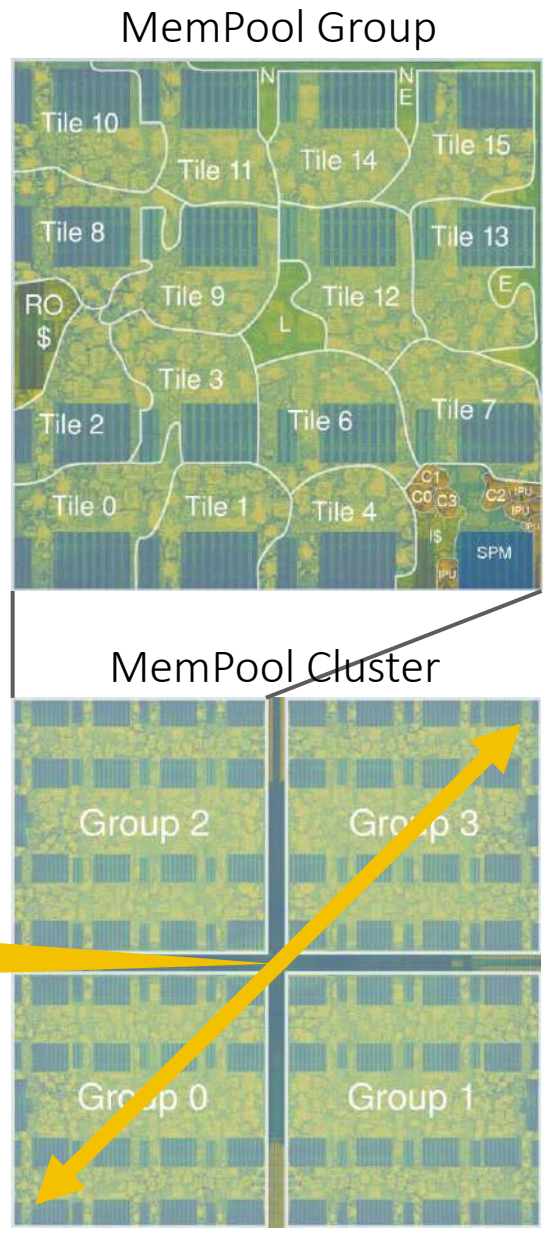
Thomas at 15:30

Parametrizable and scalable



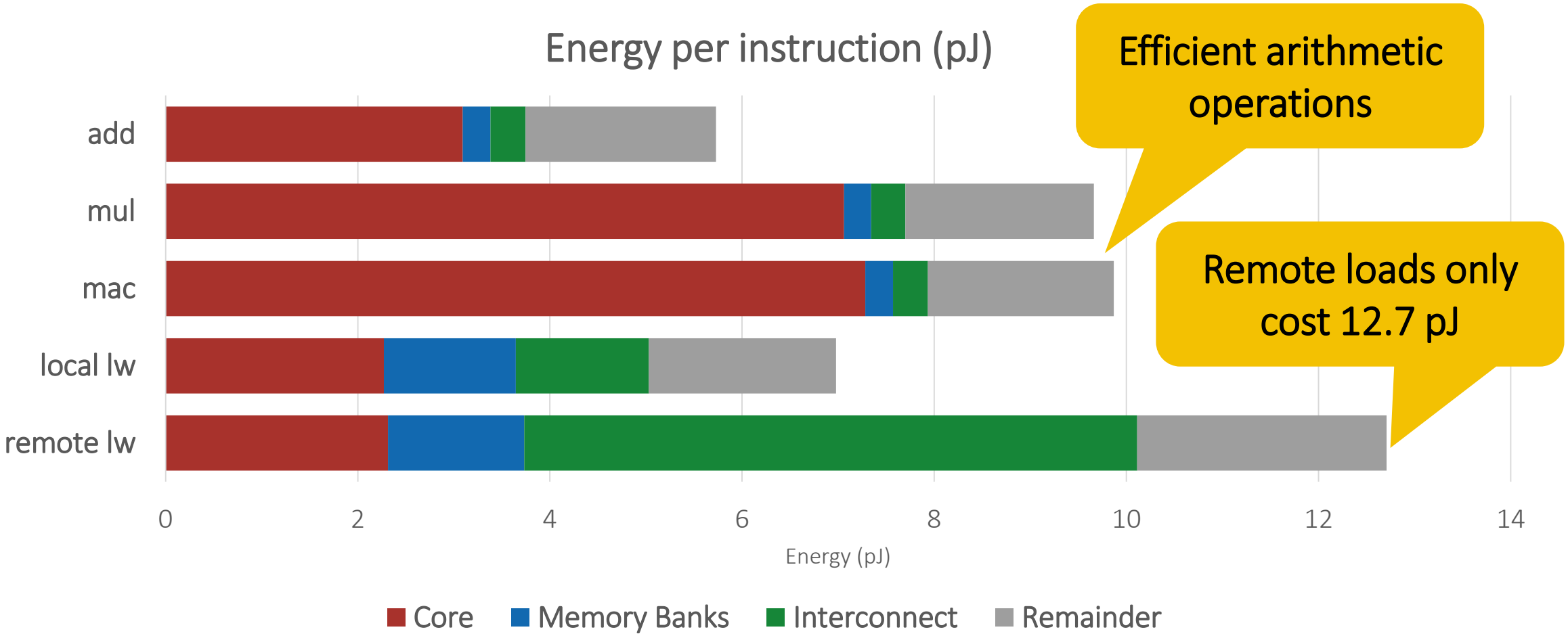
Physical Implementation

- Implemented in GF22
  - Targeting 500 MHz (SS/0.72V/125°C)
  - Reaching 600 MHz (TT/0.80V/25°C)
- Hierarchical design
- Cluster area of 13 mm^2
  - 5 mm diagonal



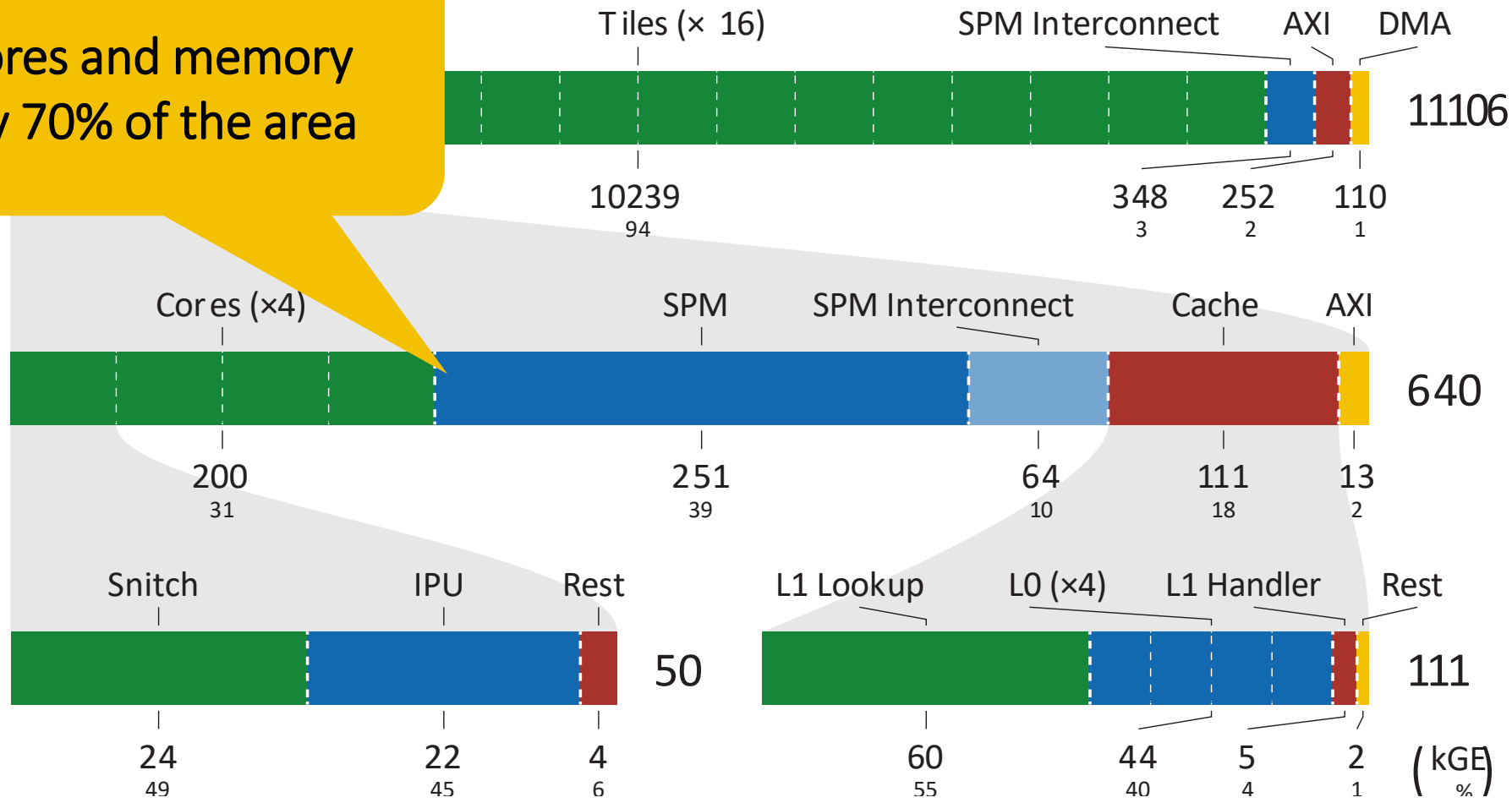
What is the cost of crossing the whole cluster?

What is the cost of traversing MemPool?



Area Breakdown of a Group

The cores and memory occupy 70% of the area

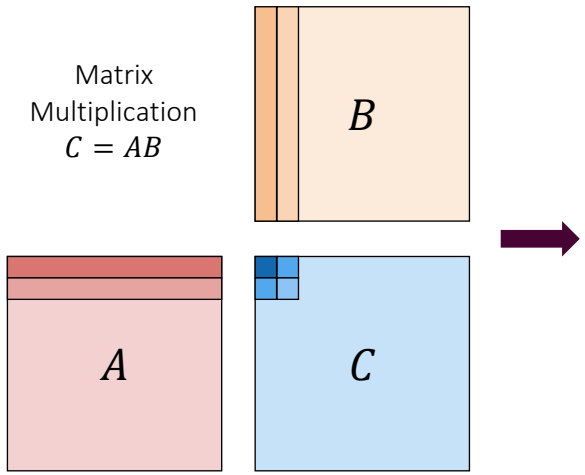


We implemented a manycore system in a single cluster.

How do we program it?

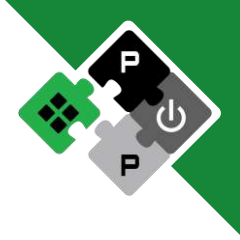
MemPool's Compilers

- Support both LLVM and GCC
  - Custom extensions fully supported
- Instruction scheduling
  - Hide load latency
  - Hide latency of accelerator




```
lw      a1,0(t2)
lw      t0,0(s0)
lw      a3,0(s1)
lw      a5,4(s1)
lw      t3,4(t2)
lw      a2,4(s0)
lw      a0,0(s2)
lw      a7,4(s2)
mul     t6,a1,a3
mul     a1,a1,a5
mul     a3,t0,a3
mul     a5,t0,a5
add     t4,t6,t4
add     t5,a1,t5
mul     t6,t3,a0
add     t1,a3,t1
mul     a0,a2,a0
mul     t3,t3,a7
add     a6,a5,a6
mul     a2,a2,a7
addi    s3,s3,2
add     t4,t6,t4
add     t5,t3,t5
add     t1,a0,t1
add     a6,a2,a6
...
```

Programming Models



- **Bare-metal C runtime**
  - All cores execute the same code (SPMD)
  - Option to go to Assembly

  \+ Full control
  \+ High performance
  – High effort

- **OpenMP**
  - Single-threaded master core
  - Fork-join behavior

  \+ Medium effort
  \+ Well-known framework
  – Runtime overhead
  – Less room for optimization

- **Halide**
  - DSL extending C++
  - Functional traits
    - Decouple function & execution

  \+ Low effort
  \+ Can optimize across pipeline stages
  – Hard to control
  – Runtime overhead
  – More challenging to support than OpenMP

How to run software on MemPool?

- **QuestaSim, VCS, Verilator**
  - RTL Simulation
  - \+ Cycle accurate
  - \+ Detailed traces
  - \- Slow simulation

↳ Hardware development and benchmarking

- **Banshee**
  - Binary translation-based functional simulator
  - https://github.com/pulp-platform/banshee
  - \+ Extremely fast simulation (×1000-10’000)
  - \+ Tracing and debugging capabilities
  - \+ Full architectural model
  - \+ Instruction accurate
  - \- Limited time model

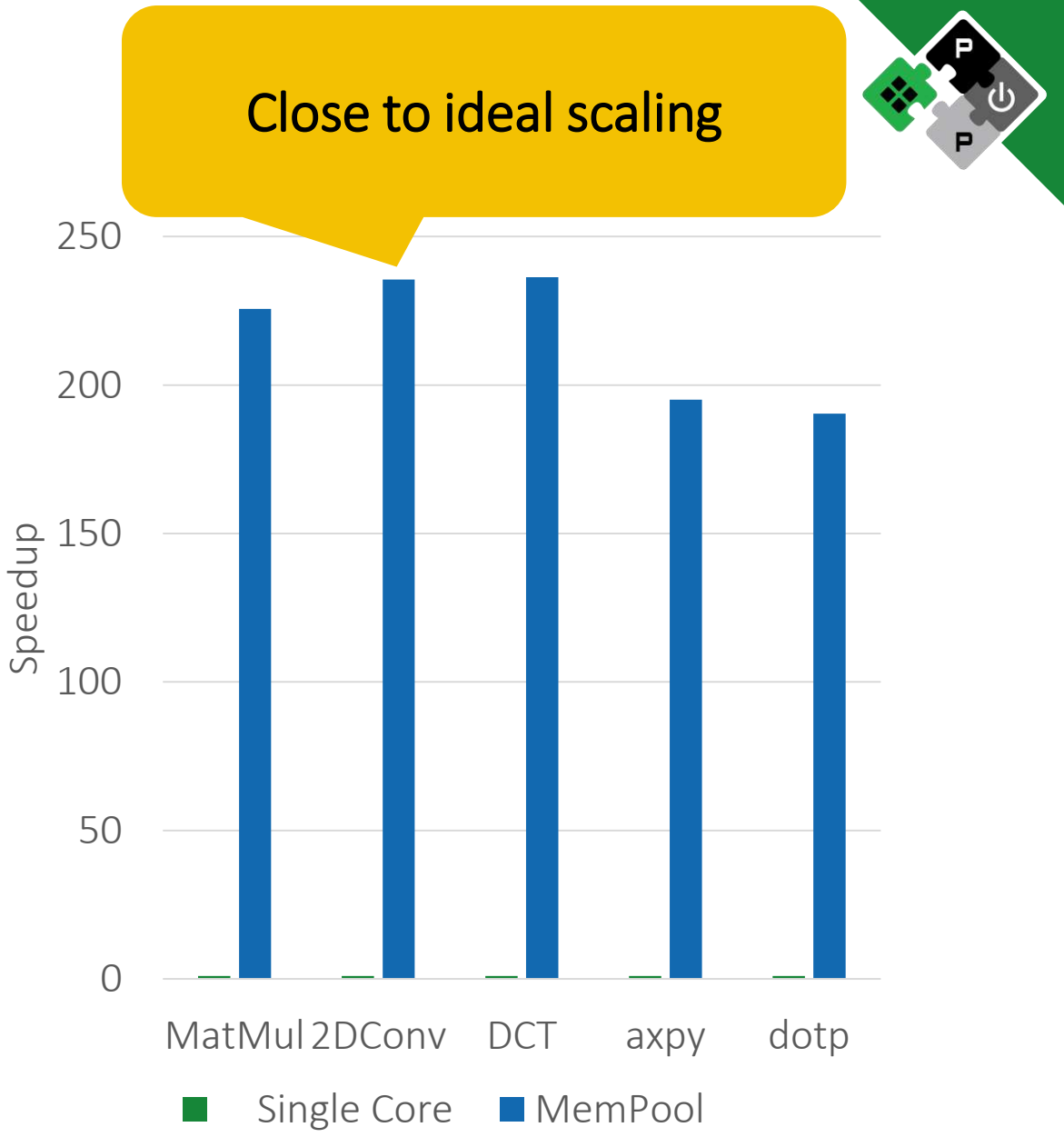
↳ Software development and prototyping

We can run applications on MemPool.

How does it perform?

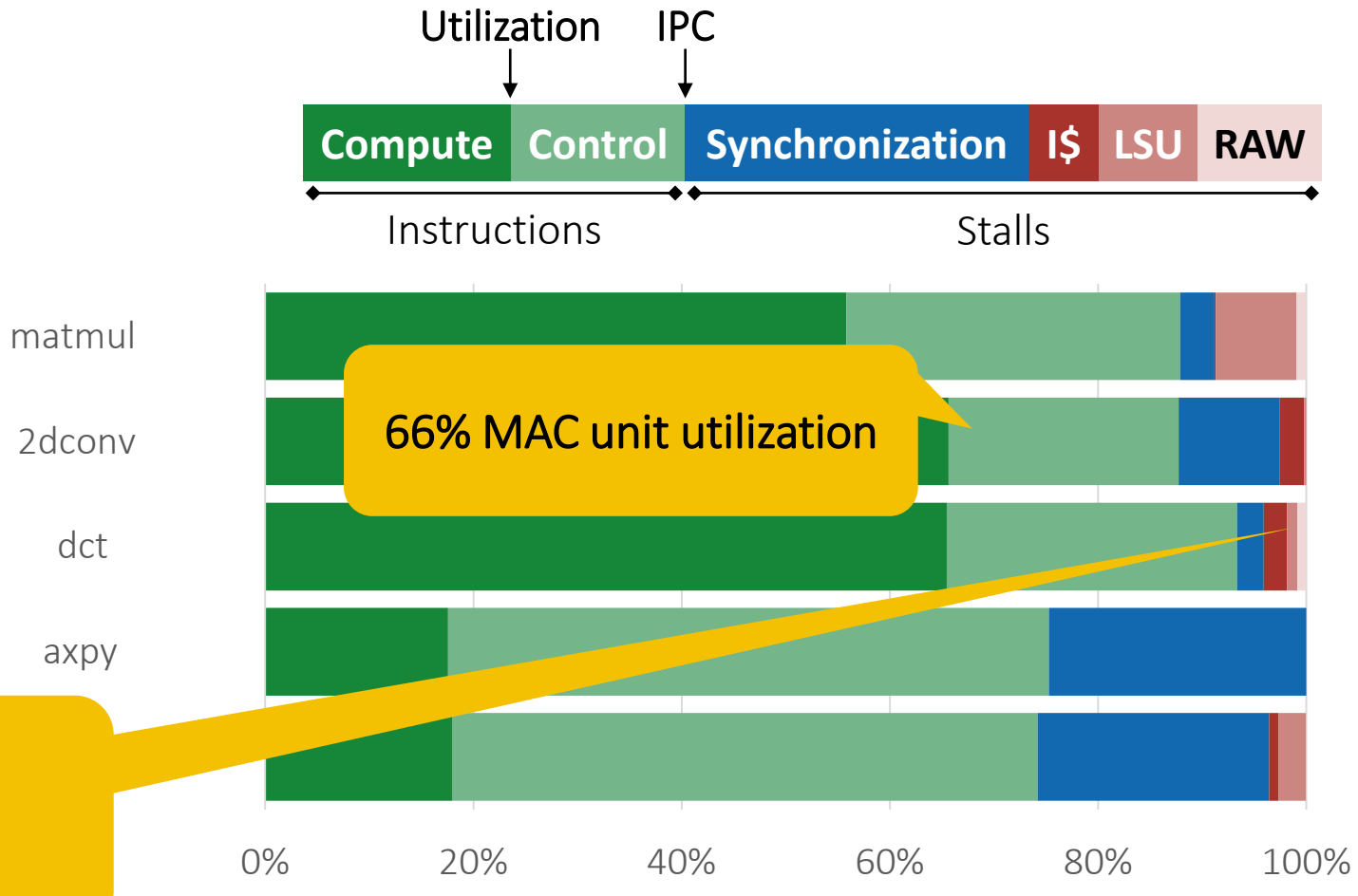
How well do we scale?

- Evaluate highly optimized kernels
- Baseline: single core system
  - No synchronization overhead or contention
- Compute-heavy kernels achieve more than **88%** of the ideal speedup
- Memory-bound kernels still achieve **75%** of the ideal speedup



Where do we lose performance?

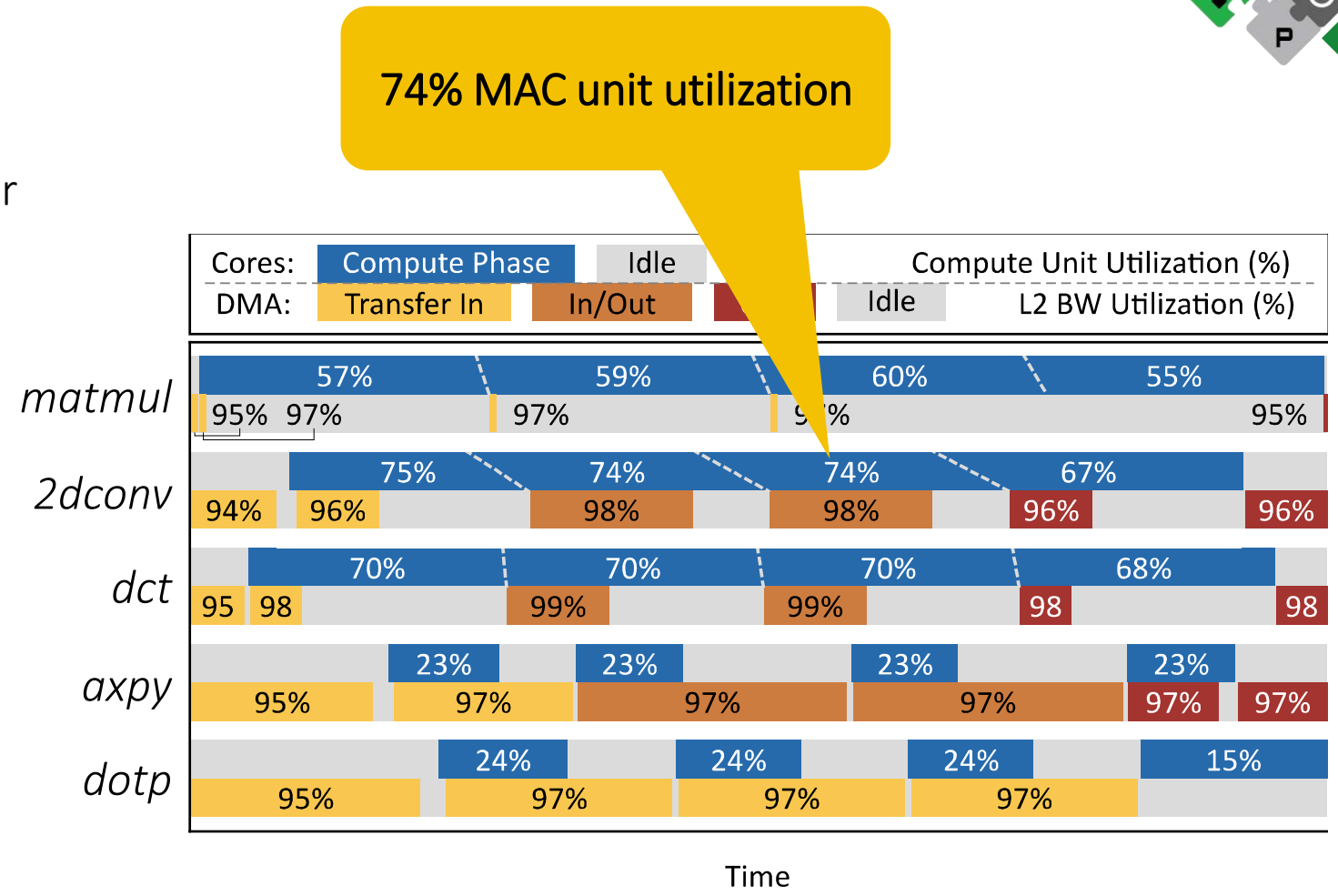
- Compute-heavy kernels:
  - Mainly synchronization overhead
  - Interconnect congestion
  - Achieve up to **66% MAC unit utilization**
- Memory-bound kernels:
  - Short execution time
  - Synchronization overhead
  - Still achieve **IPC of 75%**



Minimal architectural stalls

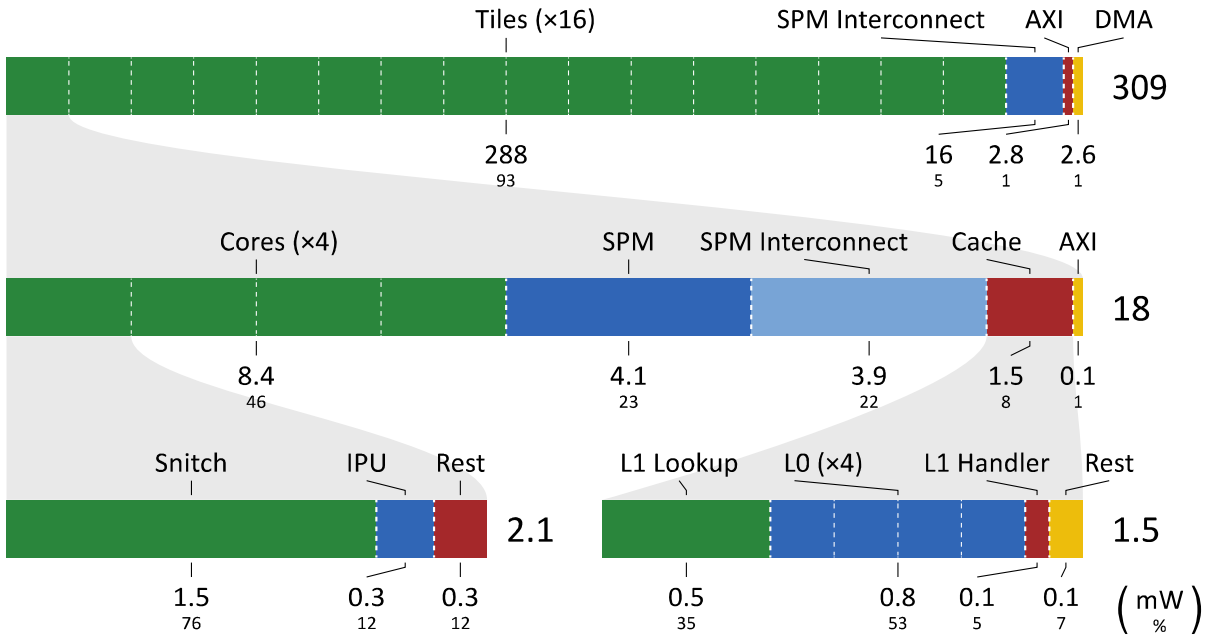
System Performance

- Double-buffer the kernels
  - Overlap compute and data transfer phase
- Compute-bound kernels:
  - Eliminate strict barriers
  - Achieve up to **74% MAC unit utilization**
- Memory-bound kernels:
  - Performance restricted by the L2 bandwidth
  - Almost at the roofline with **97% DMA utilization**



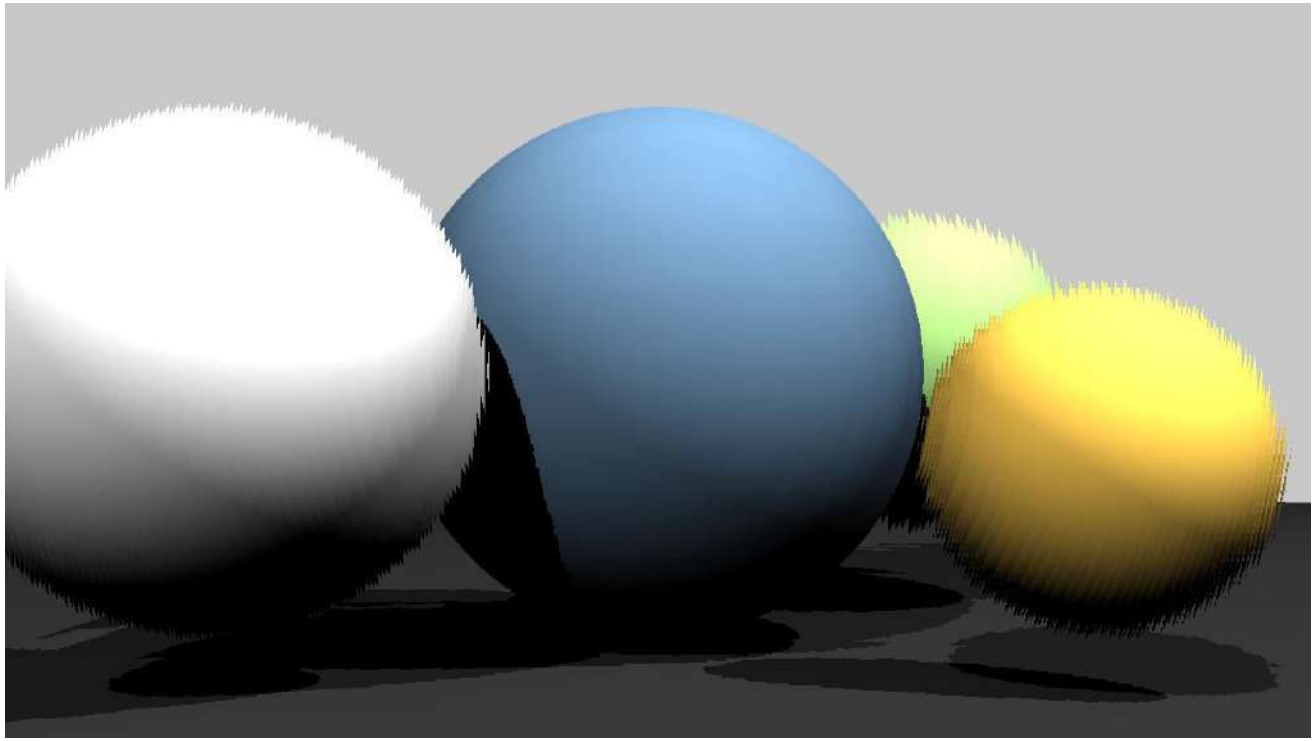
How efficient is MemPool?

- GF22 @ 600MHz (TT/0.80V/25°C)
- MatMul achieves **285 OPs/cycle**
  - MemPool consumes 1.25 W → **136 GOPS/W**
  - Snitch consumes **43%**
  - SPM and interconnect consume **47%**
- 2D Convolution achieves **336 OPs/cycle**
  - Peak energy efficiency of **170 GOPS/W**



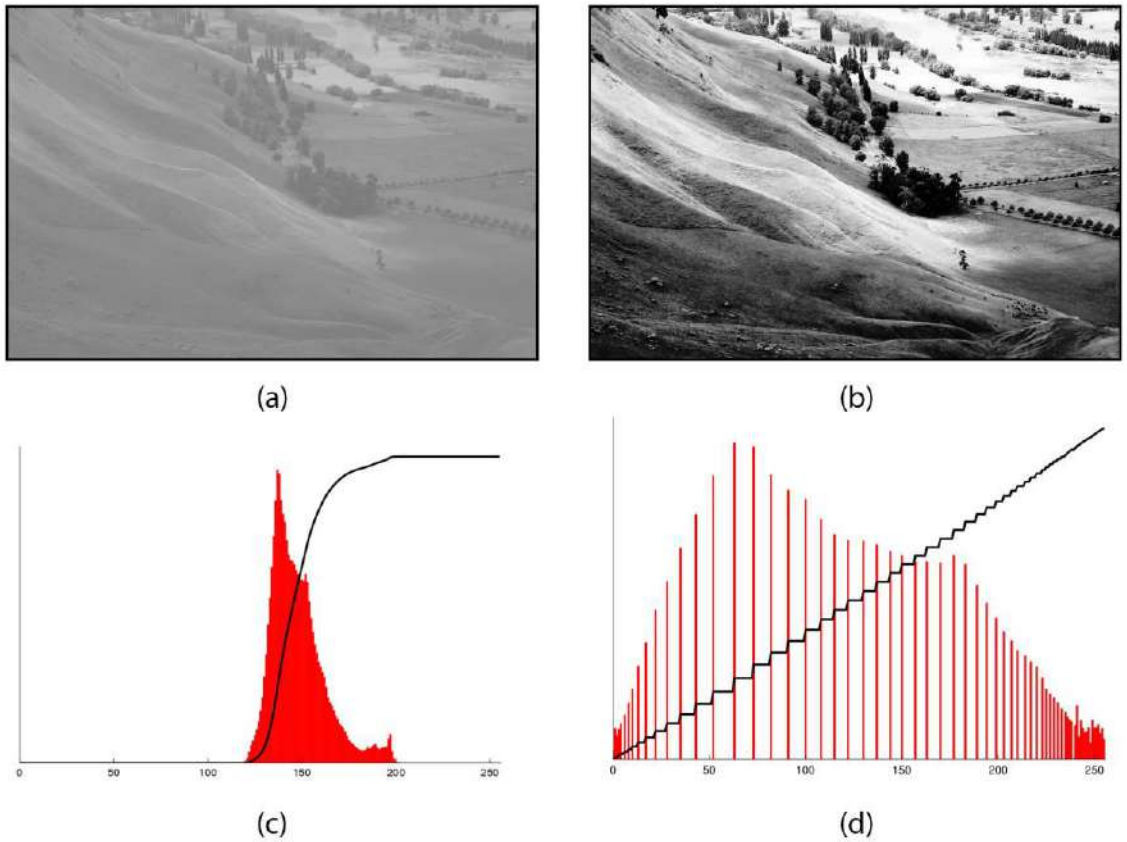
Ray Tracing on MemPool

- Non-data-oblivious
- Parallelized with **OpenMP**
- Quantized for integers
  - Proof of concept
- Static schedule:
  - **86%** of ideal speedup
- Dynamic schedule:
  - **91%** of ideal speedup



Histogram Equalization

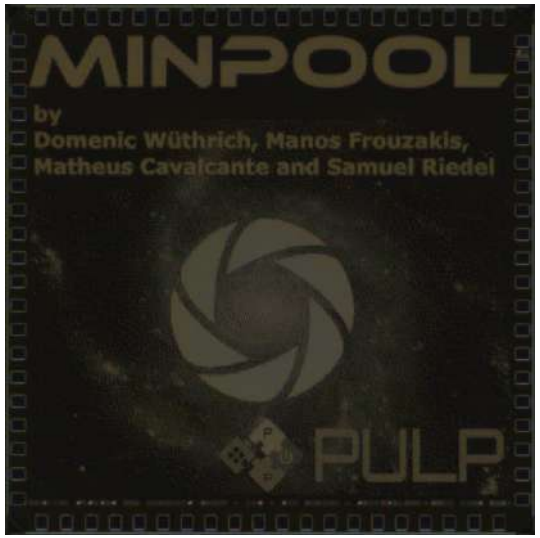
- Implemented in **Halide**
- Three key phases
  - Compute histogram
  - Compute transformation function
  - Transform image
- Achieve **40%** of the ideal speedup
  - Halide runtime is holding us back
  - 12% overhead due to reductions

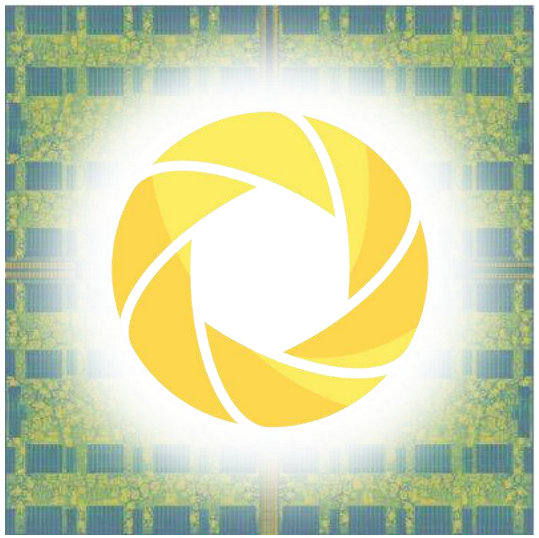


https://commons.wikimedia.org/wiki/File:Histogram_equalization.png

Is there more?

The MemPool Family

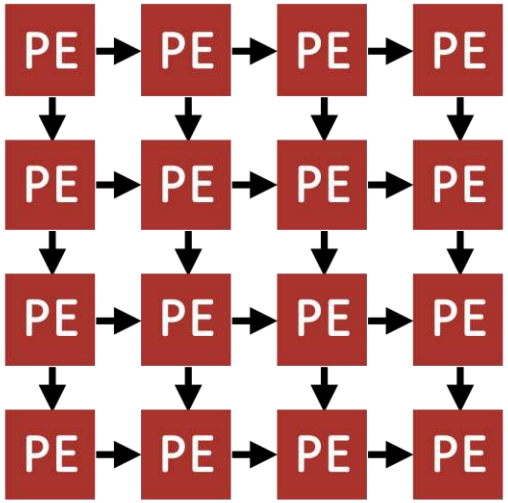




- **MinPool**: first tape-out
  - **16** cores, 64 KiB
  - TSMC 65
- **MemPool:** main driver
  - **256** cores, 1 MiB
  - GF 22FDX
  - MemPool-3D
- **TeraPool:** going even bigger
  - **1024** cores, 4 MiB

Yichao & Marco at 15:00

Extensions on MemPool



- **MemPool meets Systolic**
  - Accelerate systolic computation on MemPool
  - Hybrid architecture for flexibility and speed
- **ITA:** Transformer
  - Integrate a transformer accelerator

Gamze at 10:00

- **Spatz:** Vector processing
  - Extend MemPool's cores with small vector units

Matteo at 14:00

Try MemPool Yourself

- Fully open-source:
  - Compiler, runtimes, applications, hardware
  - Verilator support
- Available on GitHub
  - https://github.com/pulp-platform/mempool

A quick summary

- **MemPool:** A Scalable Manycore Architecture with Low-Latency Shared L1 Memory
  - 256+ cores
  - 1+ MiB of shared L1 data memory
  - ≤ 5 cycles latency (without contention)
- Physical-aware design
  - GF22 @WC: 500 MHz
- Flexible and easy to program
  - Independent cores
  - Shared memory
  - No communication overhead

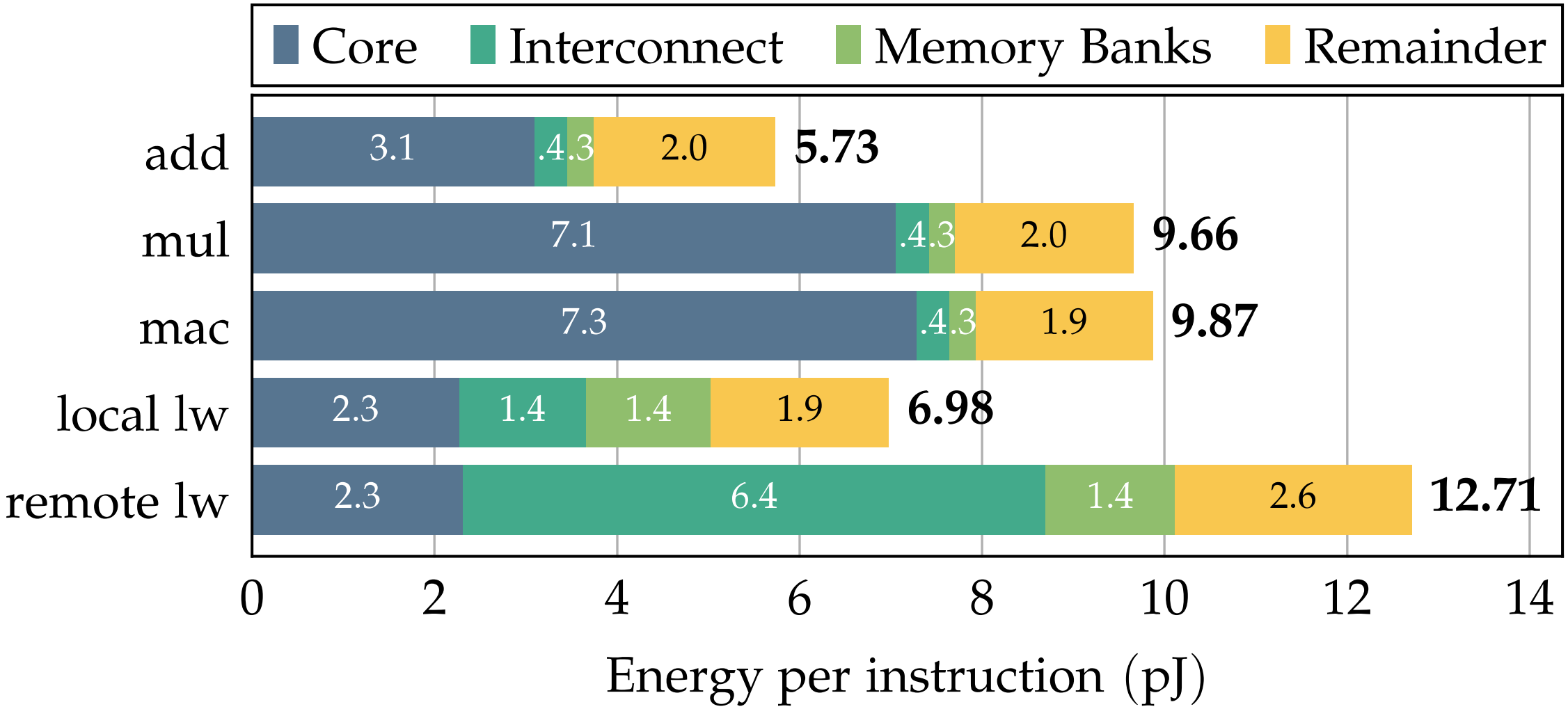
Close to ideal scaling with **88%** of the ideal speedup

Performant architecture with **>90% IPC** and **74% MAC unit utilization**

High energy efficiency **170 GOPS/W**

Halide and OpenMP runtime allow implementing complex algorithms like ray tracing efficiently.

Energy per Instruction



Performance

TABLE 1
Benchmark and power results obtained from post-layout simulations. An operation corresponds to a 32-bit addition or multiplication.

Kernel	Size	Util (IPC)	Power (W)	Performance (OP/cycle)	(GOPS/W)
matmul	256×256	0.88	1.25	285	136
2dconv	96×1024	0.87	1.19	336	170
dct	192×1024	0.93	1.02	168	99
axpy	$98'304$	0.76	1.17	90	46
dotp	$98'304$	0.74	1.21	92	46

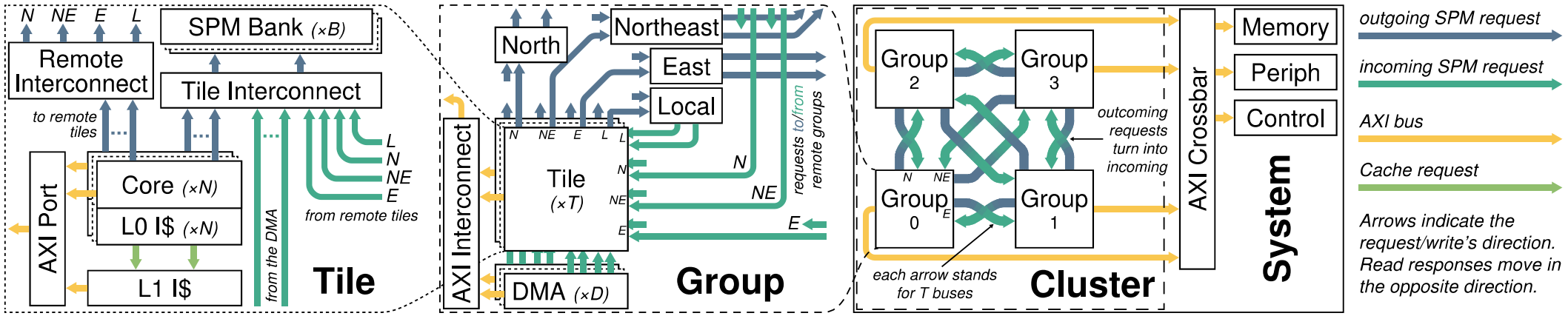
Related Work

TABLE 2
Comparison of MemPool with related, cluster-based architectures.

	Architecture	ISA	Cluster PEs	Total PEs	Independent PEs	Low latency interconnect	Open source
mesh-based	RAW [24]	32-bit MIPS-style	-	16	✓	✗	✗
	Celerity [25]	32-bit RISC-V	-	496*	✓	✗	✓
	KiloCore [26]	40-bit RISC	-	1000	✓	✗	✗
	Piton [27]	64-bit SPARC V9	-	25	✓	✗	✓
	TILE64 [28]	64-bit VLIW	-	64	✓	✗	✗
	Epiphany-V [29]	64-bit RISC	-	1024	✓	✗	✗
	Pixel Visual Core [6]	16-bit VLIW	256	2048	✗	✗	✗
crossbar-based	GAP9 [11]	32-bit RISC-V	9	9	✓	✓	≈
	RC64 [13]	32-bit VLIW	64	64	✓	✓	✗
	Manticore [14]	32-bit RISC-V	8	4096	✓	✗	✓
	MPPA3 [12]	64-bit VLIW	16	80	✓	✗	✗
	ET-SoC-1 [15]	64-bit RISC-V	32	1088	✓	✗	✗
	H1000 [4]	32/64-bit PTX	128	18432	✗	✗	✗
	This Work	32-bit RISC-V	256	256	✓	✓	✓

≈ = Closed source based on open source. *Contains additional five Linux-capable 64-bit cores and ten ultra-low-power cores.

Overview Figure



Publications

- Under review – Transactions on Computers:
  MemPool: A Scalable Manycore Architecture with a Low-Latency Shared L1 Memory
- 2022 - DATE:
  MemPool-3D: boosting performance and efficiency of shared-l1 memory many-core clusters with 3D integration
- 2021 - DATE:
  MemPool: A shared-L1 memory many-core cluster with a low-latency Interconnect

Xpulpimg: what's in the pot

Generic arithmetical operations

- Set less equal
- Absolute value
- Minimum, maximum
- Sign/zero-extension
- Clip operations
- Immediate branching

Extended load/store addressing modes

- Post-increment load/store
- Register-register load/store

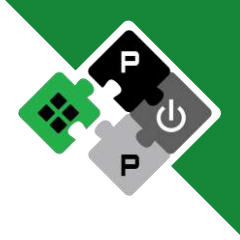
Multiply-accumulate

- Multiply and accumulate
- Multiply and subtract

Packed-SIMD

- 8-bit and 16-bit
- Vectorial and scalar replication mode
- Dot-product
- Addition, subtraction
- Average
- Abs, min, max
- Shift operations
- Logical operations
- Extract, insert, shuffle

The *Xpulp* extension



```
for (i = 0; i < 100; i++)
  c[i] = a[i] + b[i];
```

RV32I

```
mv x5, 0
mv x4, 100
Lstart:
  lb x2, 0(x10)
  lb x3, 0(x11)
  addi x10,x10, 1
  addi x11,x11, 1
  add x2, x3, x2
  sb x2, 0(x12)
  addi x4, x4, -1
  addi x12,x12, 1
  bne x4, x5, Lstart
```

11 cycles/output

Post-incr. L/S

```
mv x5, 0
mv x4, 100
Lstart:
  lb x2, 0(x10!)
  lb x3, 0(x11!)
  addi x4, x4, -1
  add x2, x3, x2
  sb x2, 0(x12!)
  bne x4, x5, Lstart
```

8 cycles/output

Hardware loops

```
lp.setupi 100, Lend
  lb x2, 0(x10!)
  lb x3, 0(x11!)
  add x2, x3, x2
Lend: sb x2, 0(x12!)
```

5 cycles/output

Packed-SIMD

```
lp.setupi 25, Lend
  lw x2, 0(x10!)
  lw x3, 0(x11!)
  pv.add.b x2, x3, x2
Lend: sw x2, 0(x12!)
```

1.25 cycles/output

Colors