



MemPool: A Shared-L1 Memory Many-Core Cluster with a Low-Latency Interconnect

Matheus Cavalcante

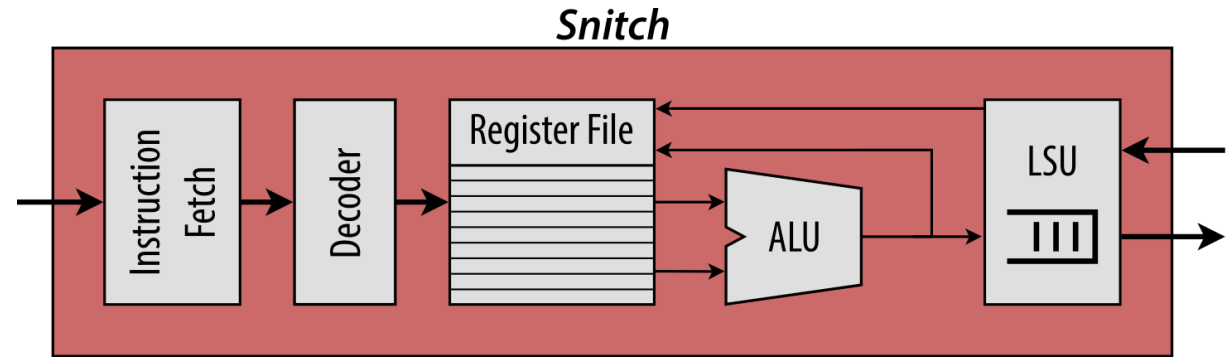
Samuel Riedel

MemPool: scaled-up shared-L1 many-core cluster

- Challenge
 - Scaling-up a shared-L1 cluster to hundreds of cores
- MemPool
 - Many-core cluster with 256 32-bit RISC-V cores
 - Shared view of 1 MiB of L1 memory, accessible within at most 5 cycles
- Physical-aware GF 22FDX design
 - 700 MHz at typical conditions
 - 480 MHz at worst-case conditions

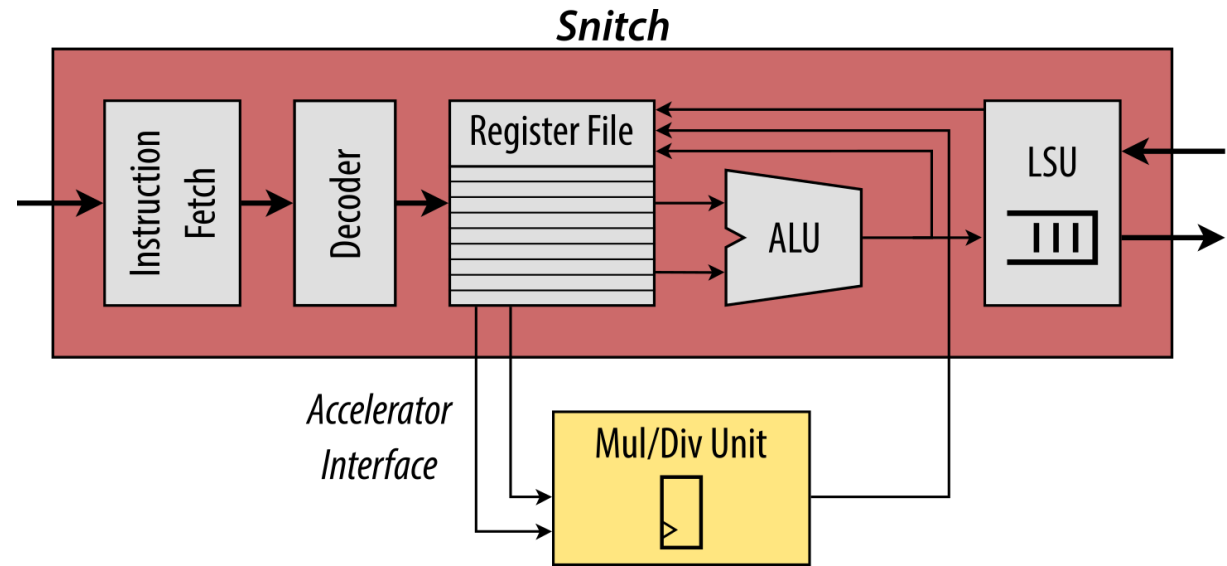
Processor: Snitch

- RISC-V 32-bit IMA
- Single-stage processor
 - Small area
- Accelerator interface
 - Pipelined
 - Complex instructions
- Tolerates multiple outstanding transactions
 - Reorder Buffer
- Snitch + Accelerator: 44 kGE



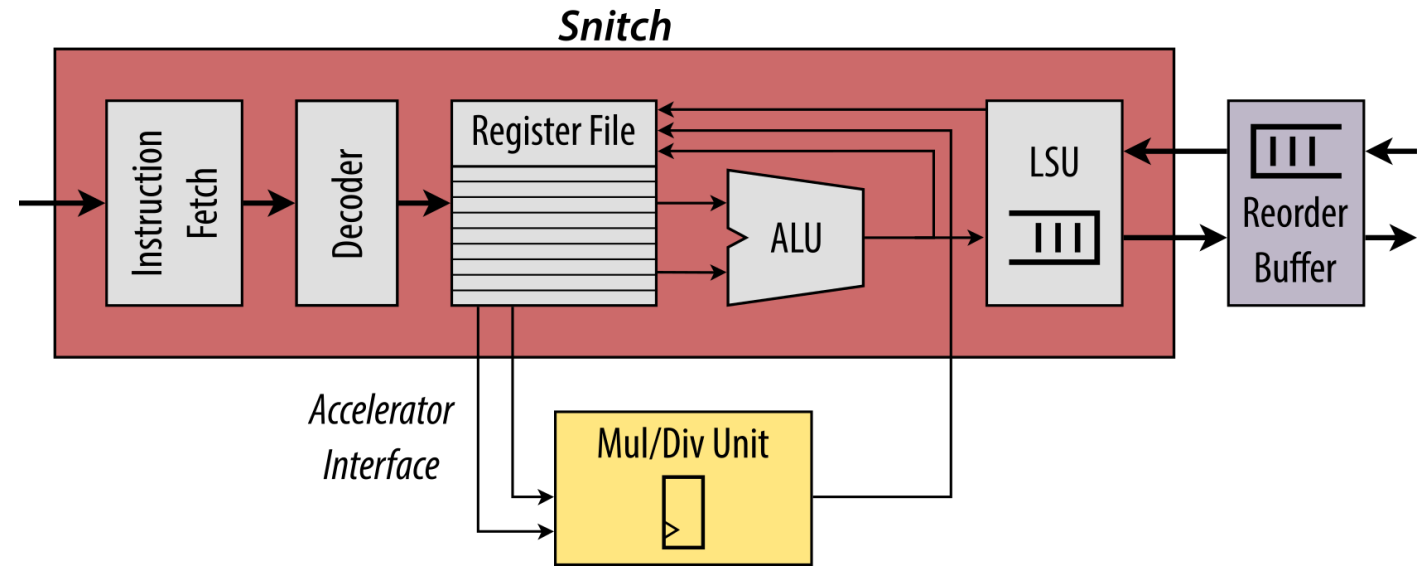
Processor: Snitch

- RISC-V 32-bit IMA
- Single-stage processor
 - Small area
- Accelerator interface
 - Pipelined
 - Complex instructions
- Tolerates multiple outstanding transactions
 - Reorder Buffer
- Snitch + Accelerator: 44 kGE



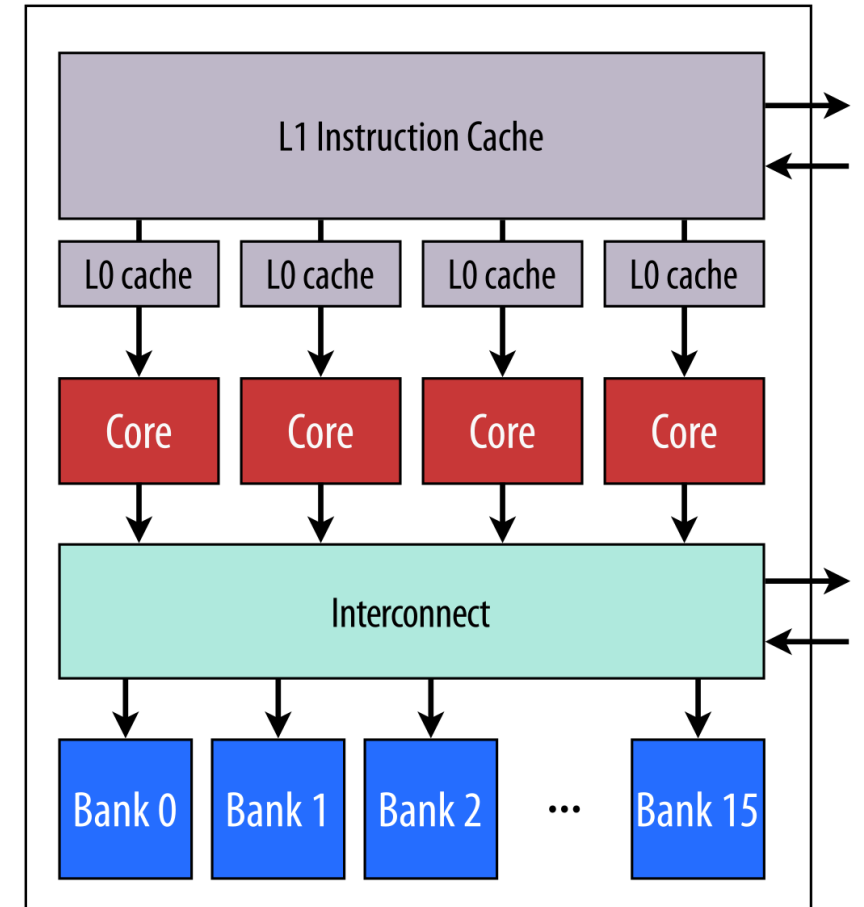
Processor: Snitch

- RISC-V 32-bit IMA
- Single-stage processor
 - Small area
- Accelerator interface
 - Pipelined
 - Complex instructions
- Tolerates multiple outstanding transactions
 - Reorder Buffer
- Snitch + Accelerator: 44 kGE



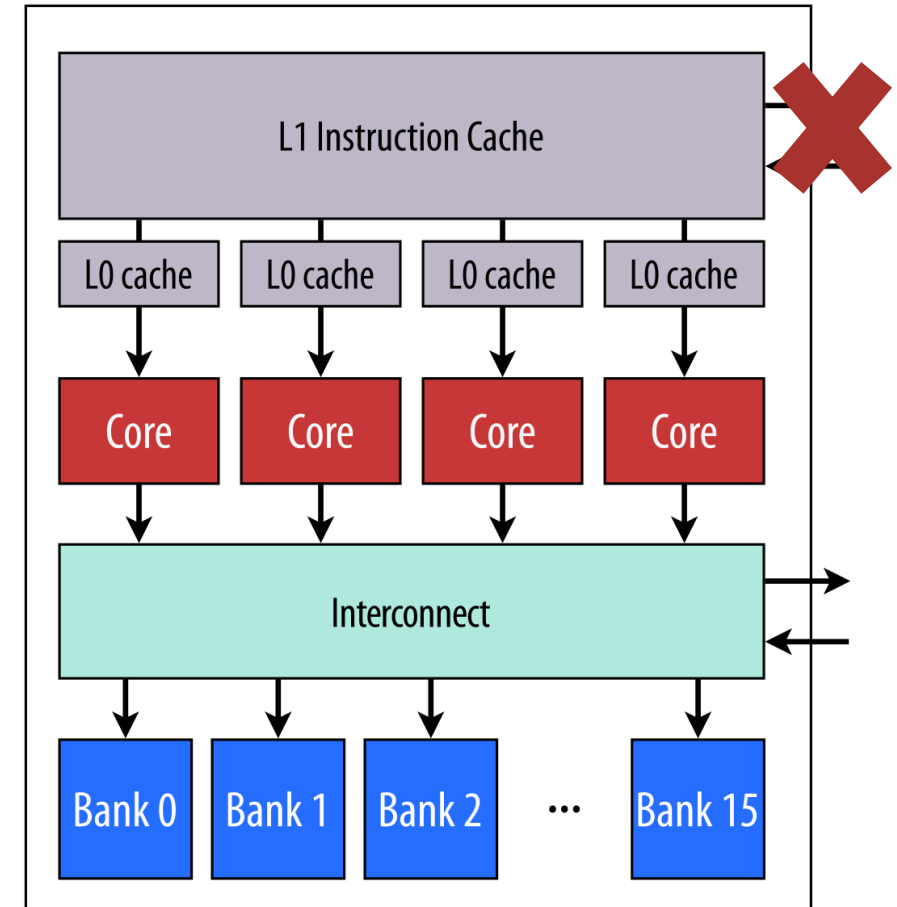
Tiles: the base unit of MemPool's hierarchy

- Four Snitch cores
 - 2 KiB shared L1 instruction cache
 - 4-way set associative
 - L0 cache private to core
 - L0 cache prefetching
 - Refill port not connected
- 16 L1 SPM Banks
 - 16 KiB
 - Accessible from local banks within one cycle



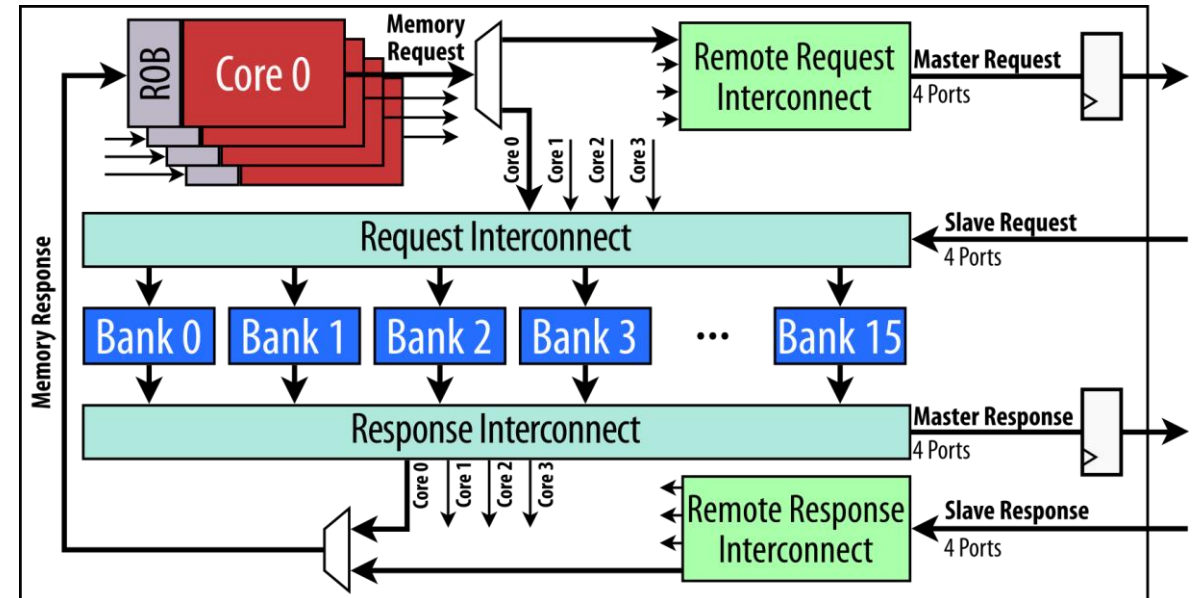
Tiles: the base unit of MemPool's hierarchy

- Four Snitch cores
 - 2 KiB shared L1 instruction cache
 - 4-way set associative
 - L0 cache private to core
 - L0 cache prefetching
 - Refill port not connected
- 16 L1 SPM Banks
 - 16 KiB
 - Accessible from local banks within one cycle



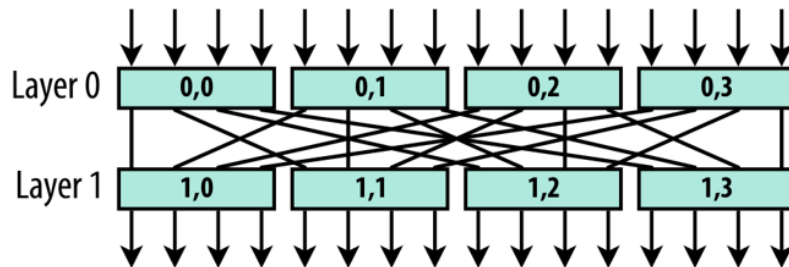
Tiles: the base unit of MemPool's hierarchy

- Request and response interconnects:
 - 8x16 FC logarithmic crossbars
- Each tile has 4 ports to access remote tiles
 - Remote request and response interconnects are 4x4 FC logarithmic crossbars
- Local banks accessible in one cycle



Assembling the MemPool cluster: Top₁ and Top₄

- First idea: connect the 64 tiles with a radix-4 butterfly network global network
 - Better scaling (wrt area and congestion) than a fully-connected crossbar
 - 16x16 radix-4 butterfly network:

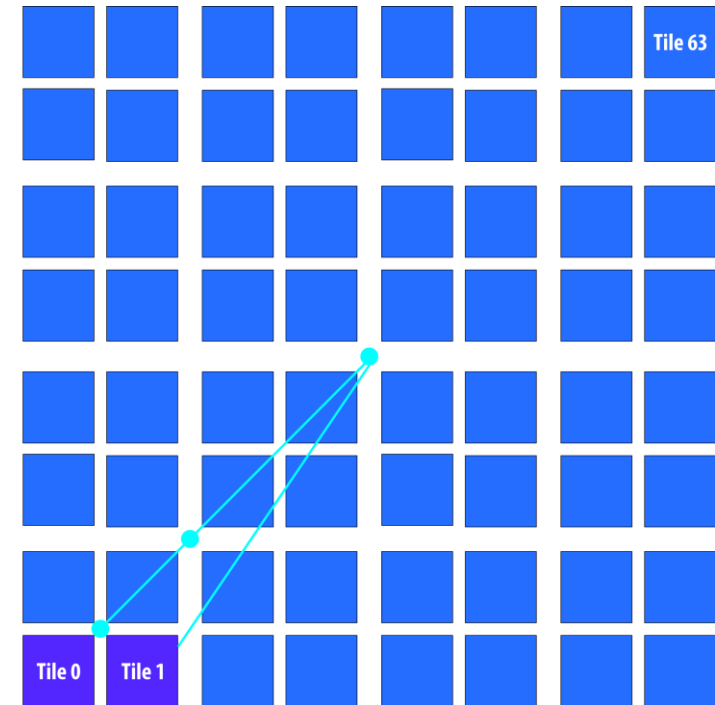


- At least 2 identical butterflies
 - Request and response networks

- Top₁: each tile has one port to access the global network
 - One 64x64 butterfly network for the requests
 - One 64x64 butterfly network for the responses
- Top₄: each tile has *four* ports to access the global networks
 - Four* 64x64 butterfly networks for the requests
 - Four* 64x64 butterfly networks for the responses
 - I.e., four times the interconnects of Top₁

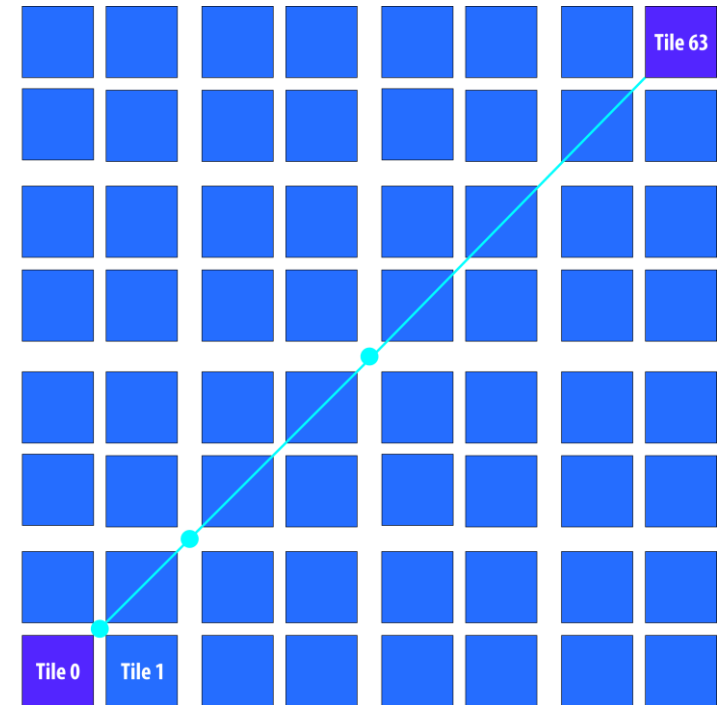
Assembling the MemPool cluster: Top₁ and Top₄

- Both Top₁ and Top₄ have a latency of *five cycles* to access any remote bank
- Regardless of how close the tiles are, requests take the same latency
 - Routing congestion at the center of the design
 - Lower efficiency
 - Unfeasibility?



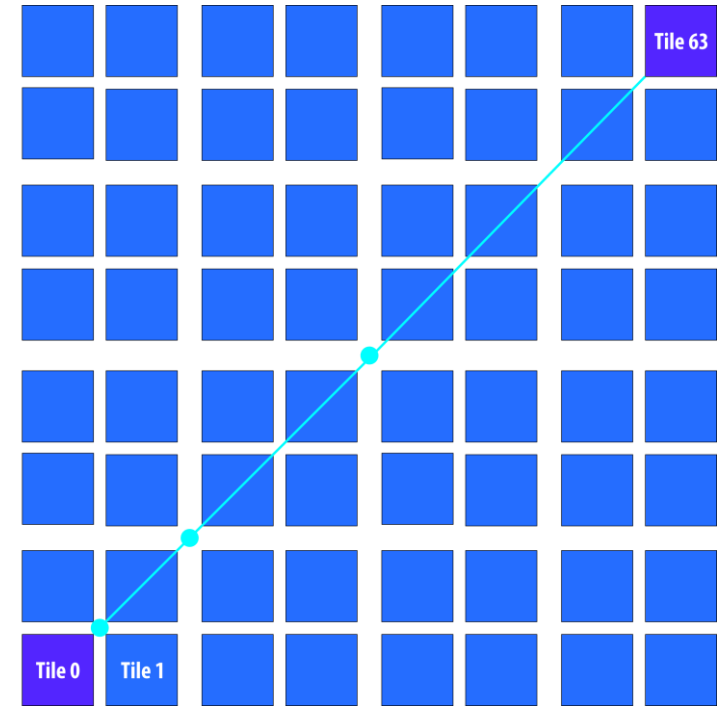
Assembling the MemPool cluster: Top₁ and Top₄

- Both Top₁ and Top₄ have a latency of *five cycles* to access any remote bank
- Regardless of how close the tiles are, requests take the same latency
 - Routing congestion at the center of the design
 - Lower efficiency
 - Unfeasibility?



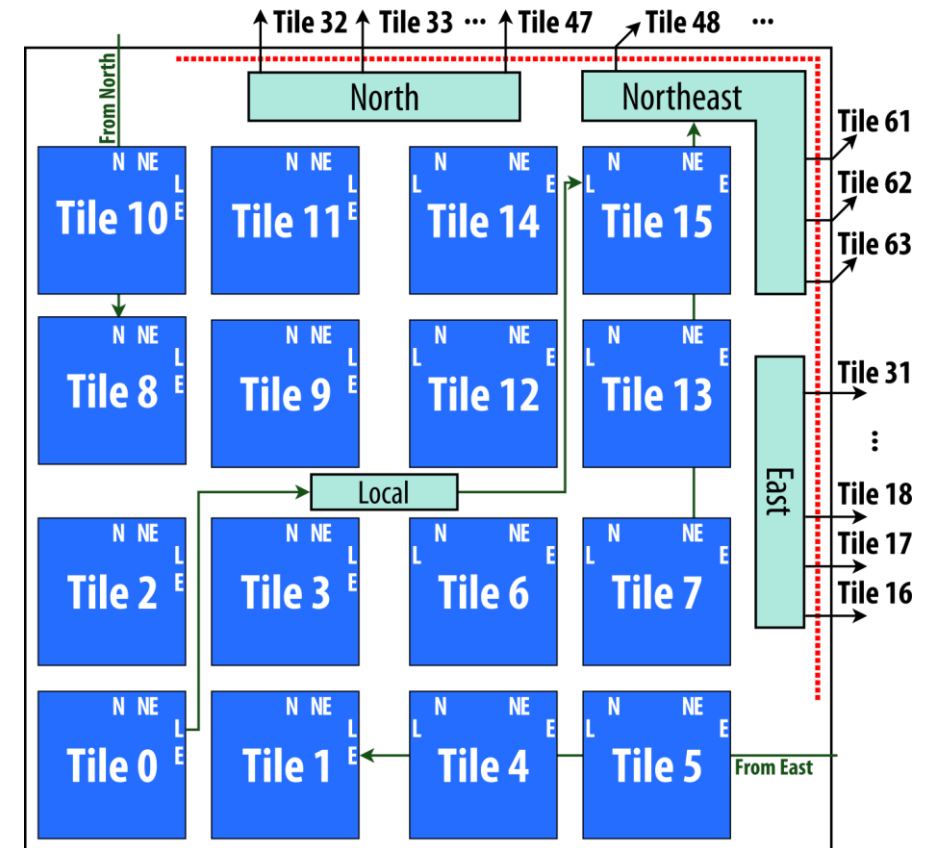
Assembling the MemPool cluster: Top₁ and Top₄

- Both Top₁ and Top₄ have a latency of *five cycles* to access any remote bank
- Regardless of how close the tiles are, requests take the same latency
 - Routing congestion at the center of the design
 - Lower efficiency
 - Unfeasibility?
- The idea behind **Top_H**:
 - Exploring locality through another hierarchy level



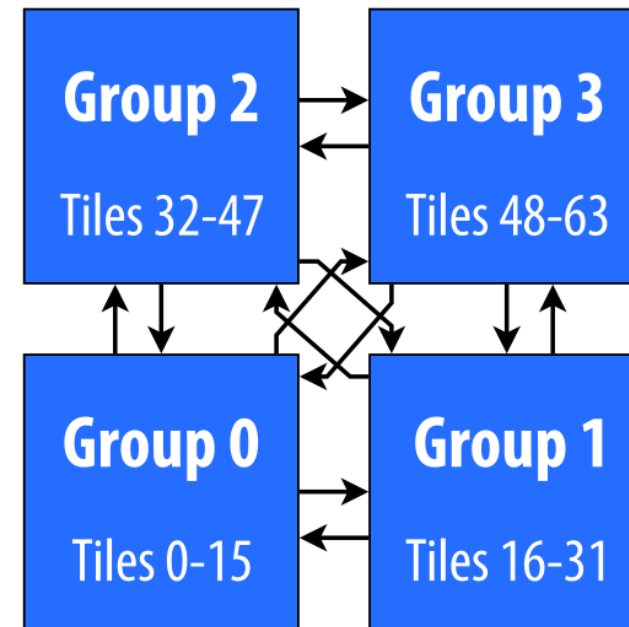
Intermediate hierarchy level: Groups

- 16 tiles compose a *group*
- Tiles in the same group can be accessed within 3 cycles
 - "Local" interconnect is realized as a 16x16 radix-4 butterfly networks
- Each group has three other 16x16 radix-4 butterfly interconnects to access remote groups
 - "North," "Northeast," and "East" interconnects
 - Register boundary implies remote groups can be accessed within 5 cycles



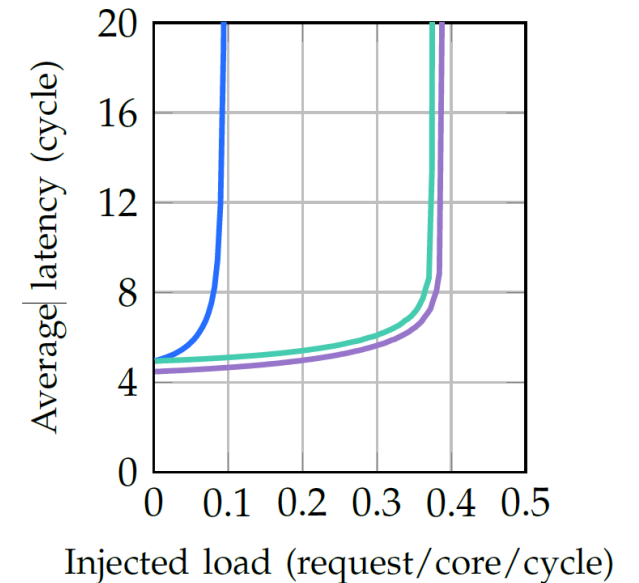
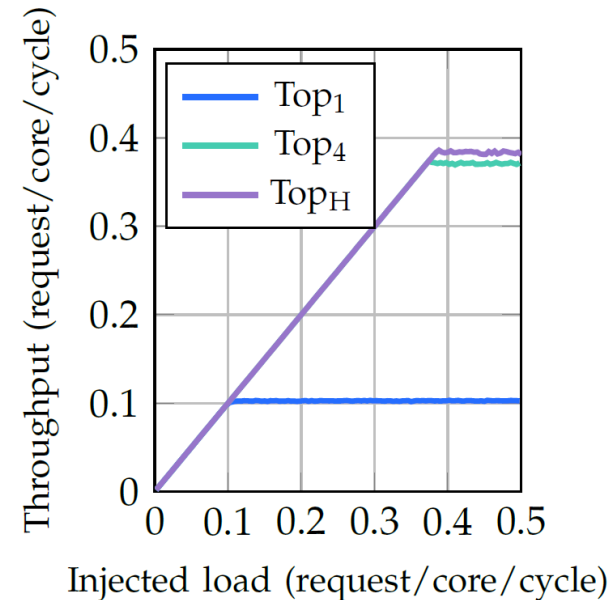
Cluster: integration of four groups

- The cluster is composed of four groups
 - No logic at this hierarchy level
 - All L1 SPM banks accessible within 5 cycles
- Particularly high congestion at the center of the design
 - Wire crossing of Groups 0-3 with Groups 1-2



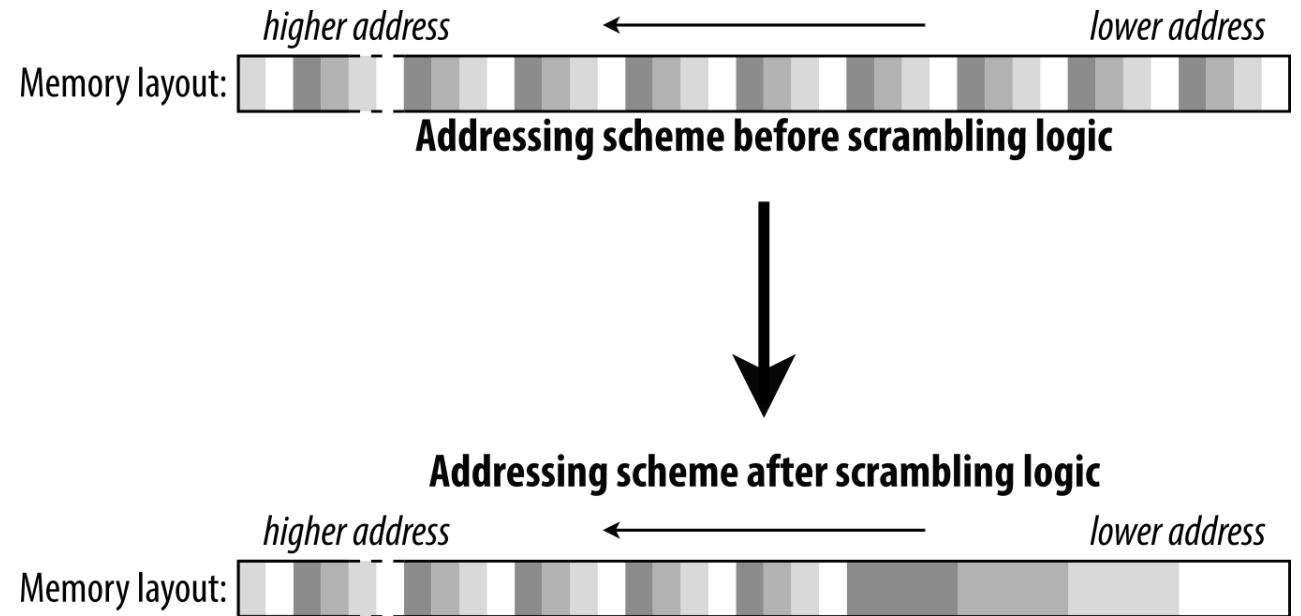
Throughput and Latency Analysis

- Cores replaced with synthetic traffic generators
- Top_1 has low throughput and high latency
 - Traffic of four cores is concentrated through a single port to the global interconnect
- Top_4 and Top_H are pretty much equivalent
 - Top_H has slightly lower latency
 - Both maintain the latency below 6 cycles for a load of 0.25 req/core/cycle

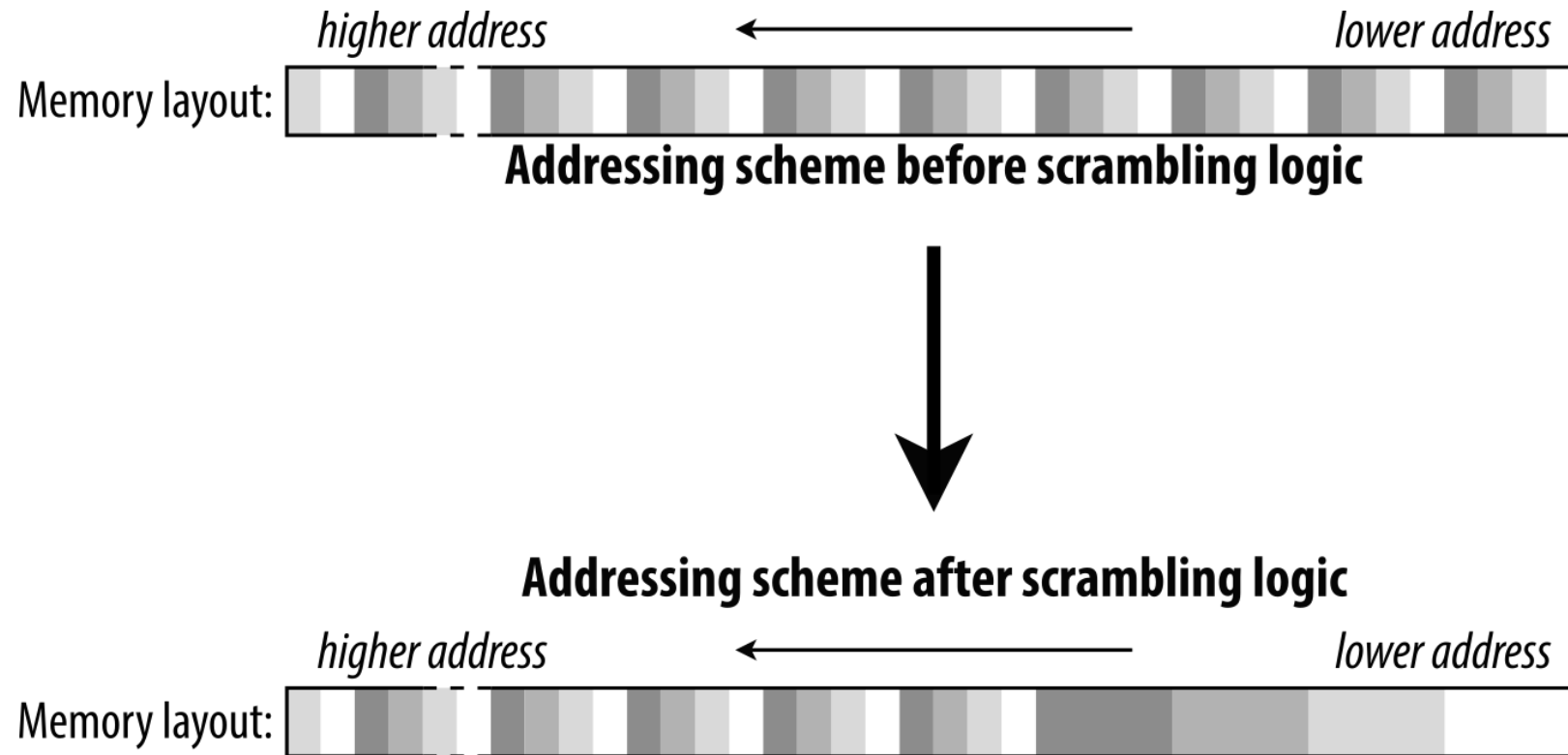


Hybrid Addressing Scheme

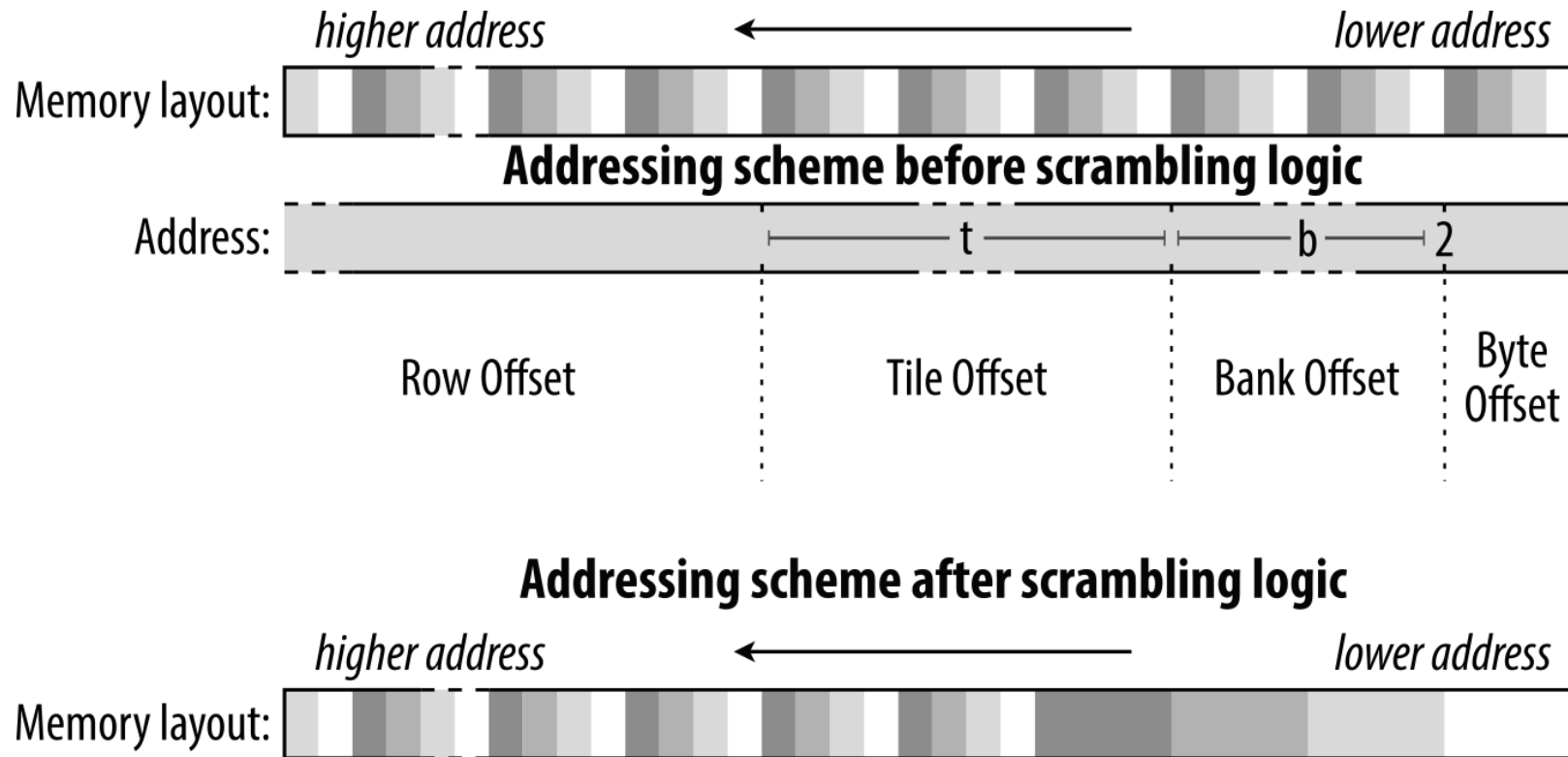
- Try to keep the requests local
 - Higher bandwidth
 - Lower latency
 - Less energy
- Hybrid memory map
 - Sequential region → local to tile
 - Interleaved region → global
- ‘Scramble address’
 - Interconnect remains unchanged



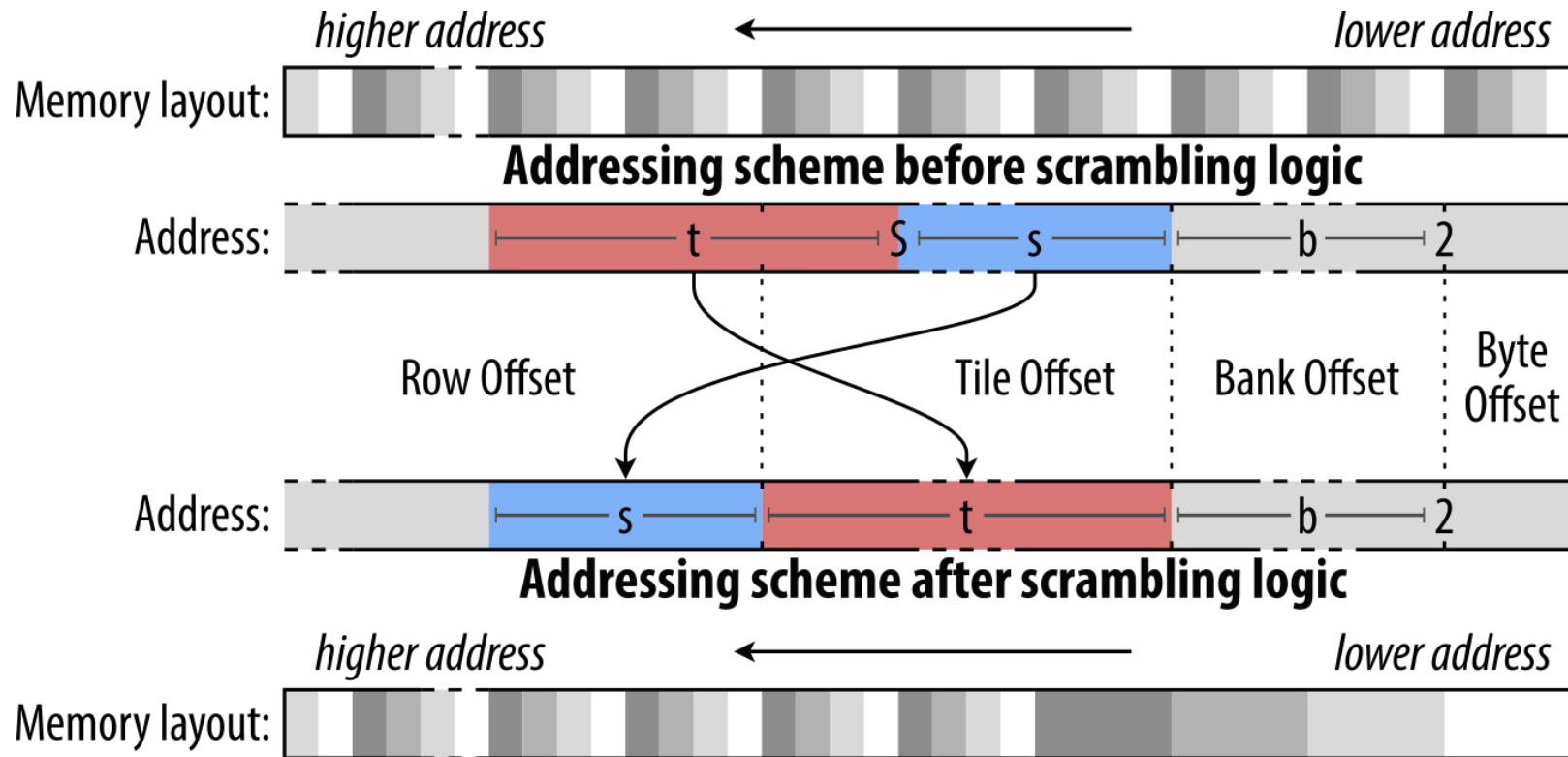
Scrambling logic



Scrambling logic

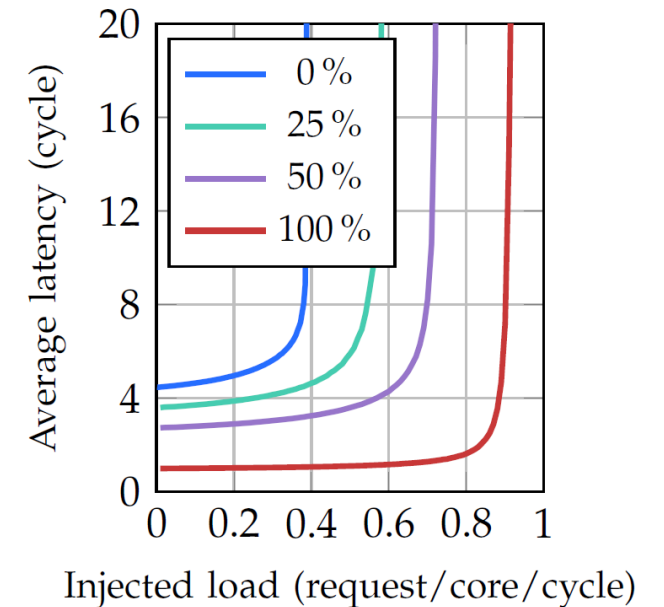
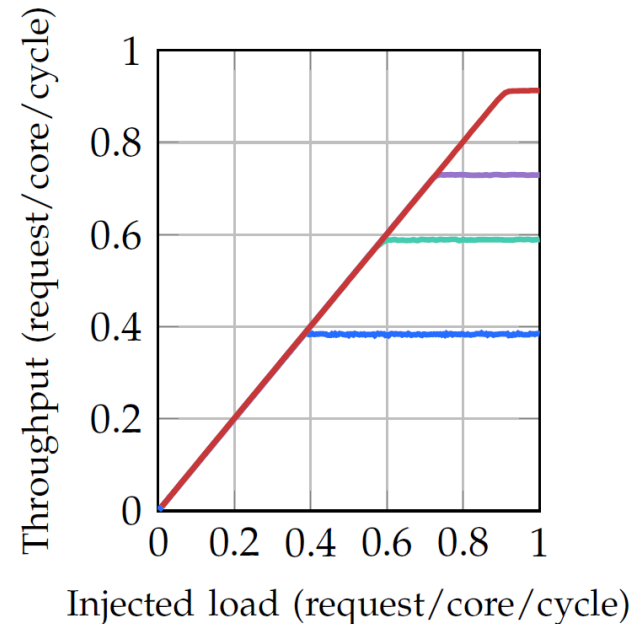


Scrambling logic



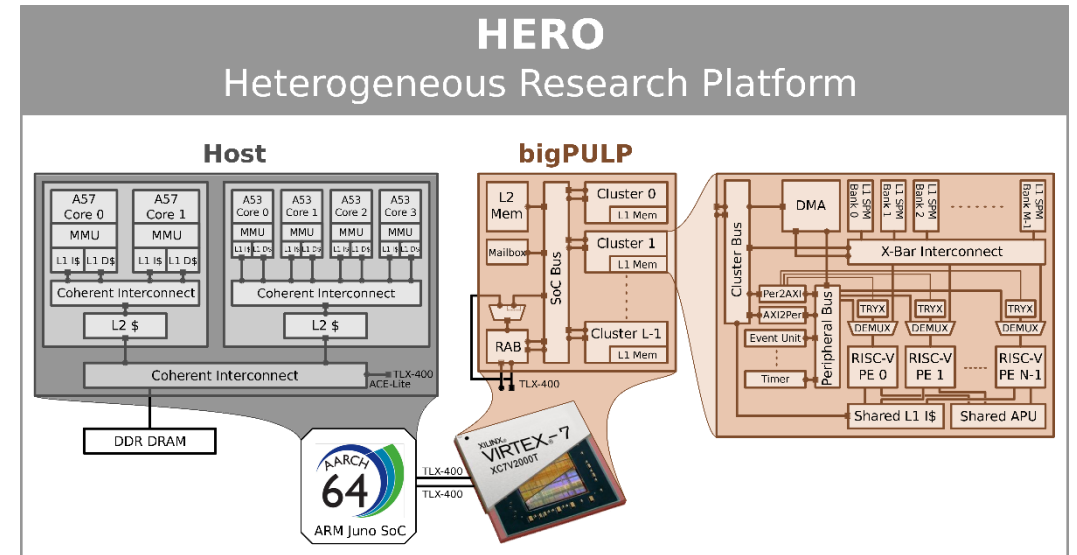
Scrambling logic: Performance Impact

- Significant performance impact
 - 50% improvement with only 25% of the accesses being local
- Low hardware overhead
 - Only a wire crossing and a mux
- Transparent to the programmer
 - Contiguous memory map
 - No aliasing



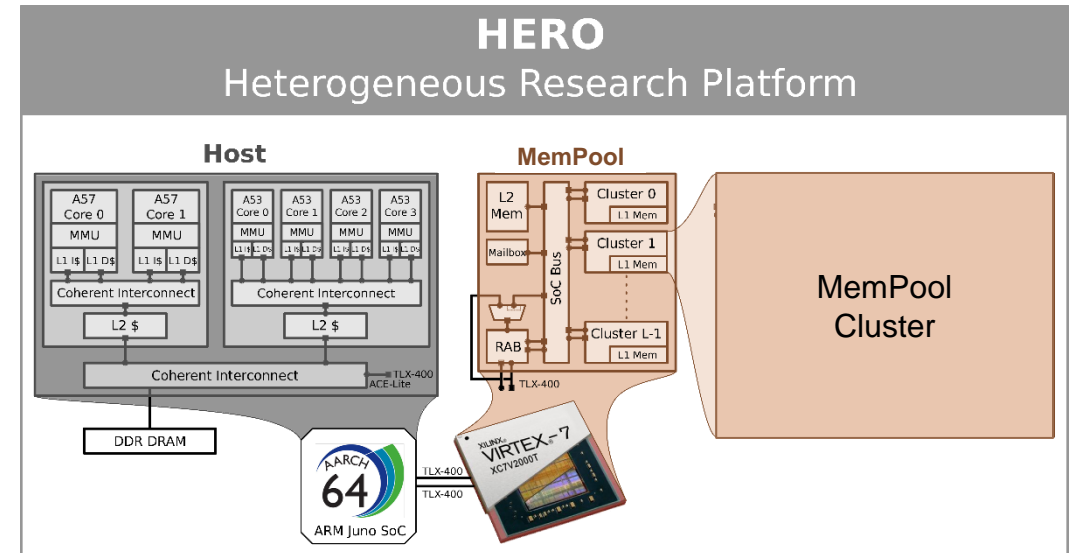
What is missing: creating a full SoC

- Instruction path
 - Unconnected at tile's port
 - We need an additional interconnect for instructions → AXI
- Cache hierarchy
- Full SoC
 - Integration with host processor
 - System around cluster
 - Leverage HERO
 - Currently carried out by student



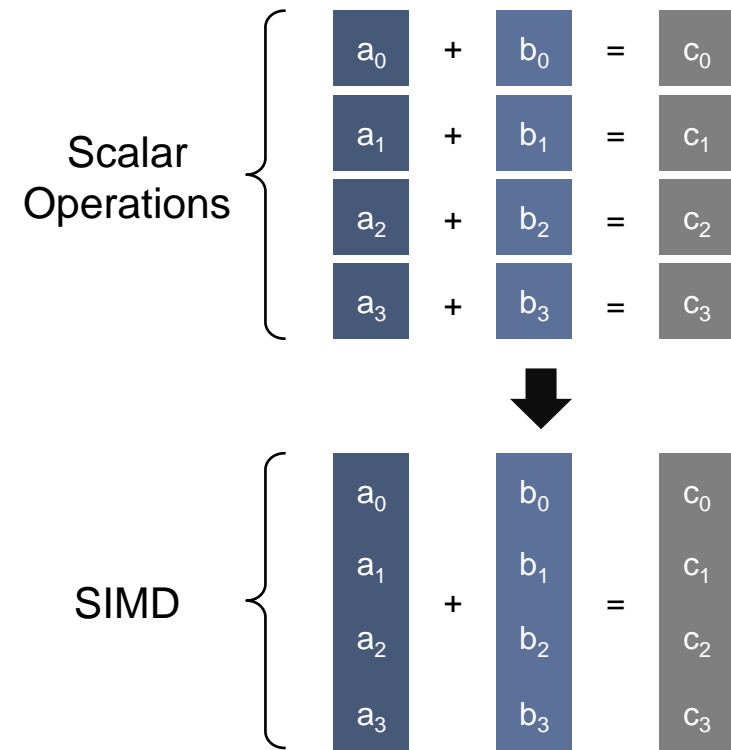
What is missing: creating a full SoC

- Instruction path
 - Unconnected at tile's port
 - We need an additional interconnect for instructions → AXI
 - Cache hierarchy
- Full SoC
 - Integration with host processor
 - System around cluster
 - Leverage HERO
 - Currently carried out by student



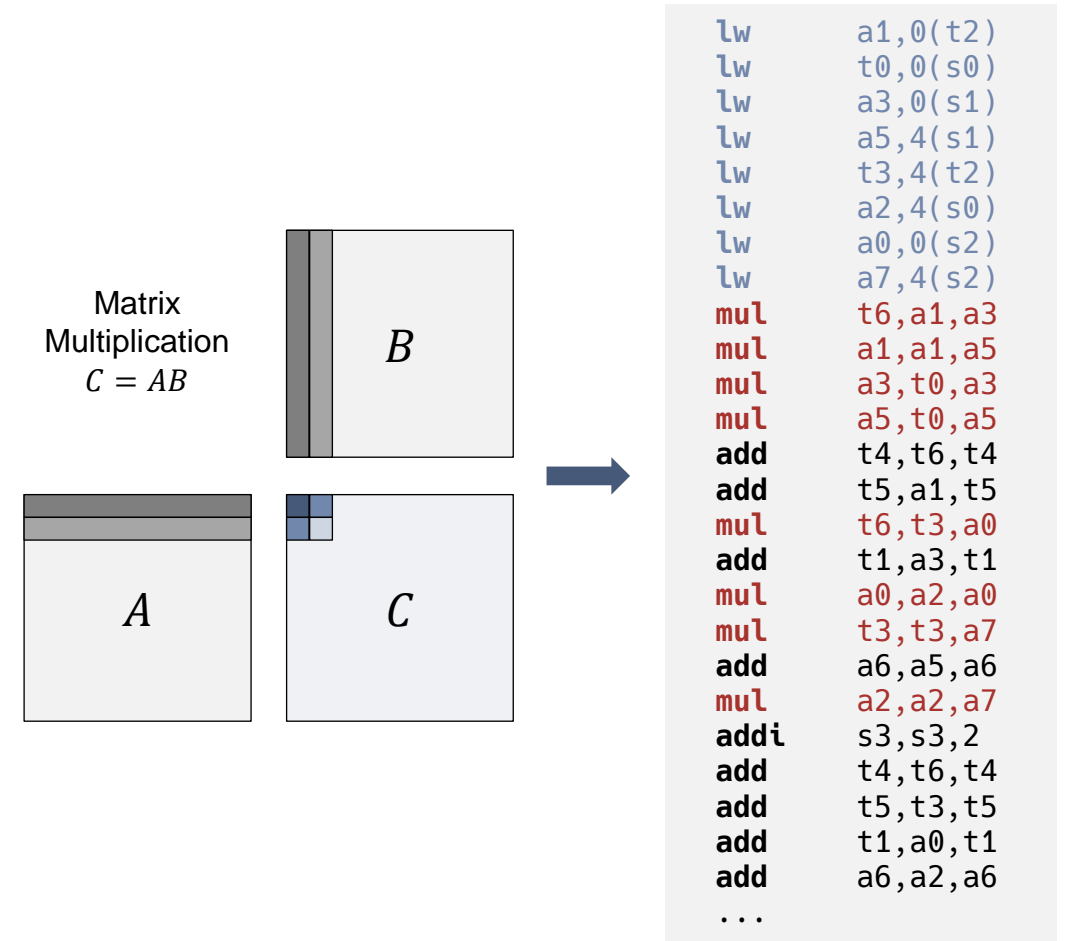
Further optimizations

- Data movement
 - We need to integrate DMAs
 - Leverage the instruction interconnect
- Synchronization
 - Implement efficient synchronization
 - Between host and accelerator
- Domain-specific instructions
 - Extend Snitch with DSP and SIMD instructions
 - Master student already started working



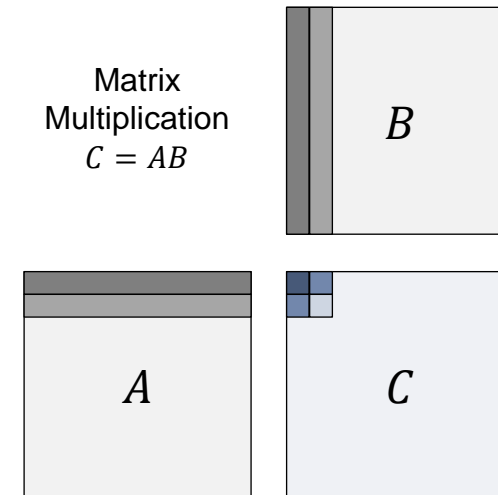
Benchmarks: hiding the latency

- Support both LLVM and GCC
- Instruction scheduling
 - Hide load latency
 - Hide latency of accelerator
- Using handwritten assembly
- Halide is working for MemPool
 - We provide a minimal Halide runtime
 - Have a few simple kernels working
 - Halide makes use of instruction scheduling



Runtime and kernels

- We have a minimal runtime
- Allocate the stack in the sequential memory region
- Benchmark with three kernels
 - Matrix multiplication $C = AB$
 - 2D convolution \rightarrow mostly local accesses
 - DCT \rightarrow only local accesses



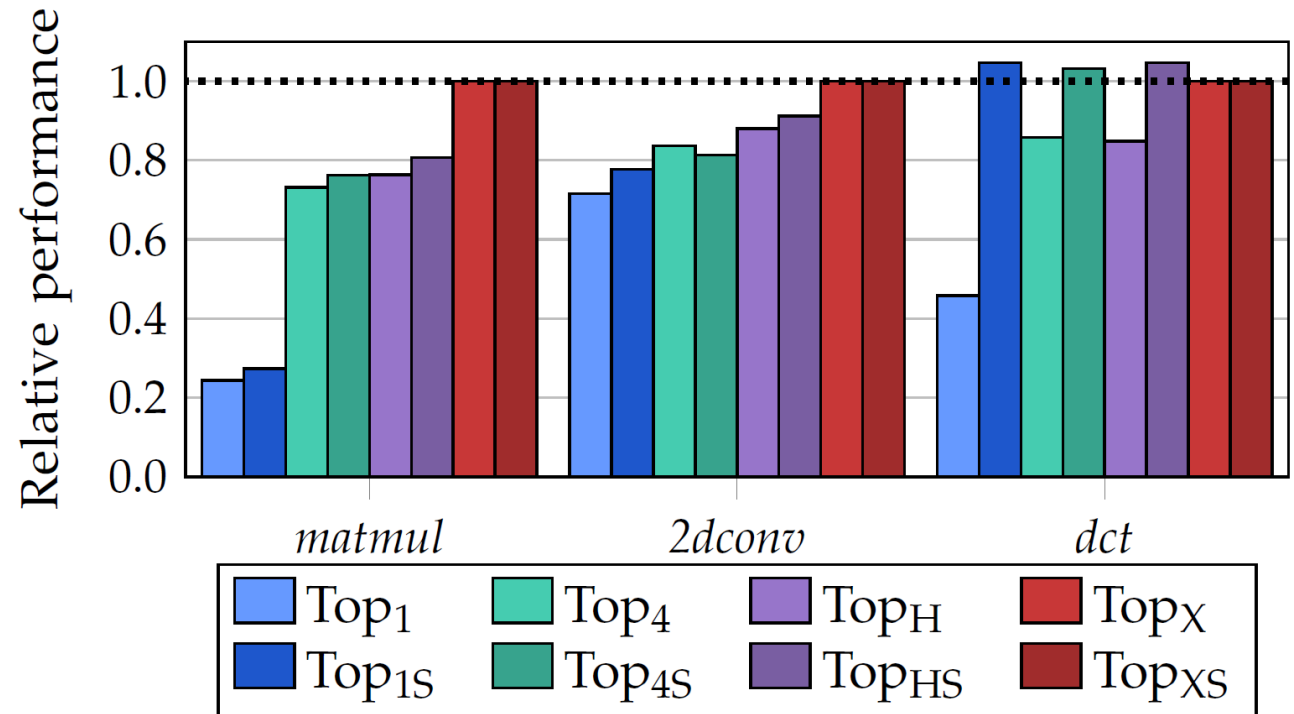
```

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
...

```

Benchmarks: can we compete with the impossible?

- Baseline system
 - 256 cores in one cluster
 - 256x1024 FC logarithmic crossbar
 - 2 cycles access latency
 - Physically infeasible
- Top_H outperforms $\text{Top}_{1/4}$
- Mostly global accesses
 - Still achieve 80% of the baseline's performance
- Only local accesses
 - Even slightly outperform baseline

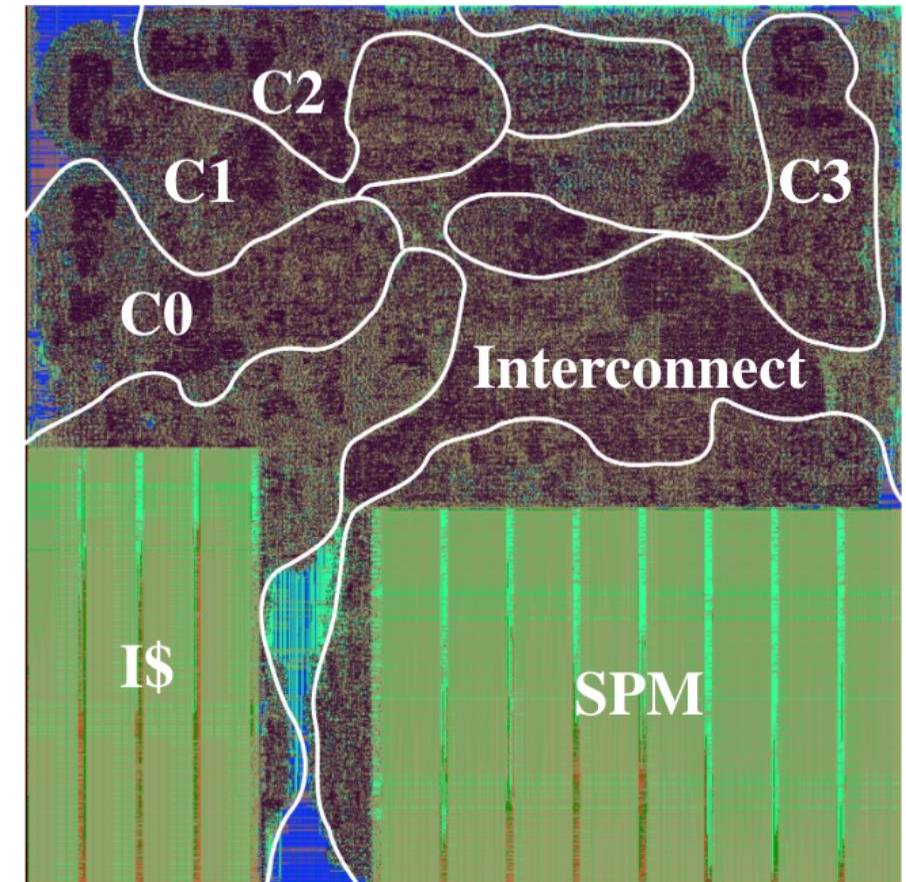


Compiler and software: what's to come?

- LLVM
 - Implement support for custom DSP instructions
 - Optimize instruction scheduling
- Halide
 - Move to more complex kernels → Exploit Halide's strengths
 - Extend Halide's model to support heterogeneous compilation
 - Look into Halide's auto-scheduler for cross-compilation

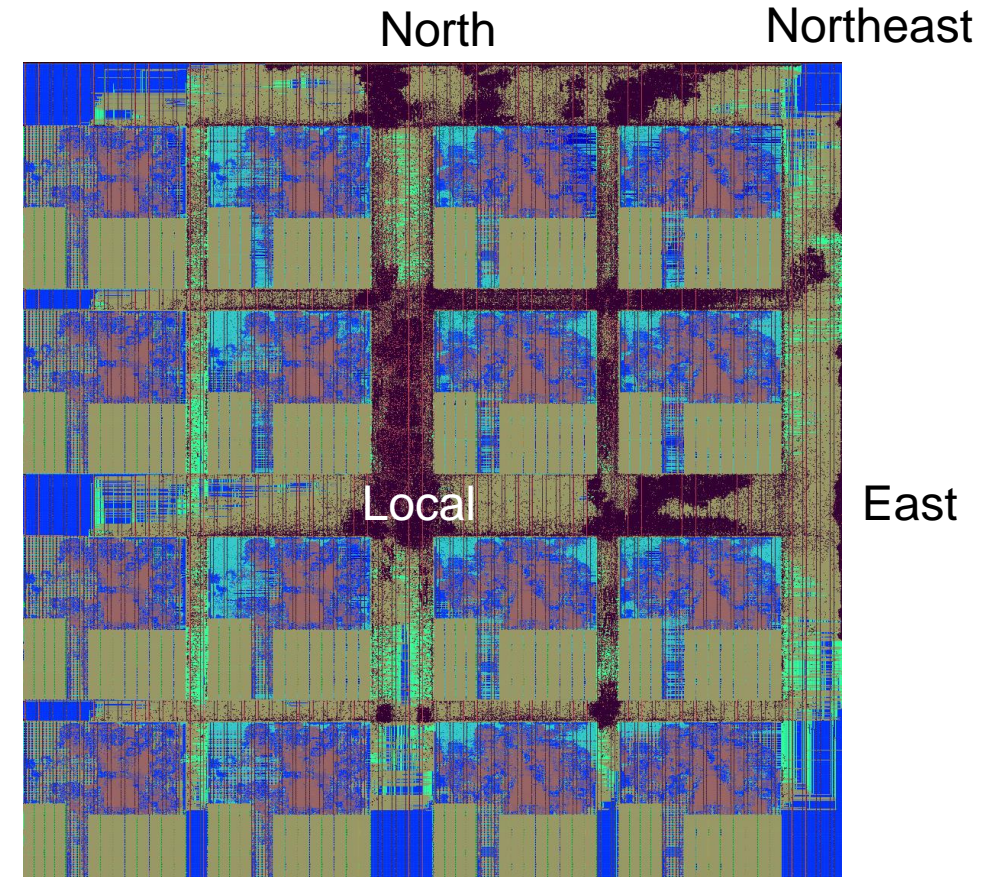
GF22 back-end results for the tile

- Synopsys flow:
 - DesignCompiler 2019.12 for synthesis and IC Compiler II 2019.12 for PnR
- Pretty dense and compact tile, using minimum routing resources:
 - 425 μm x 425 μm (908 kGE)
 - 72.8% utilization
 - I\$ memory macros: area inefficient
 - Longest path: I\$ - Snitch - SPM
 - Routed up to layer C4
 - Leaving four layers for above-the-tile routing



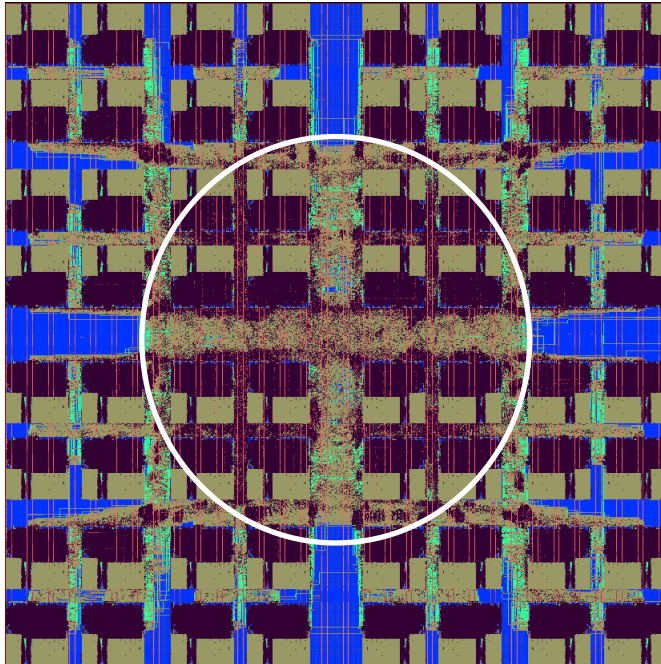
GF22 back-end results for the group

- Two limiting factors:
 - Routing congestion:
 - 4 interconnects competing for routing resources
 - Propagation delay:
 - The wires need to cross long distances → high utilization of upper routing layers
- 2.12 mm x 2.12 mm
 - 64% of the group area is occupied by the tiles
- Routed using the whole metal stack
 - Four layers more than the tiles
- Critical path is 34 gates long (27 buffers)
 - Wire propagation delay is 37% of the critical path

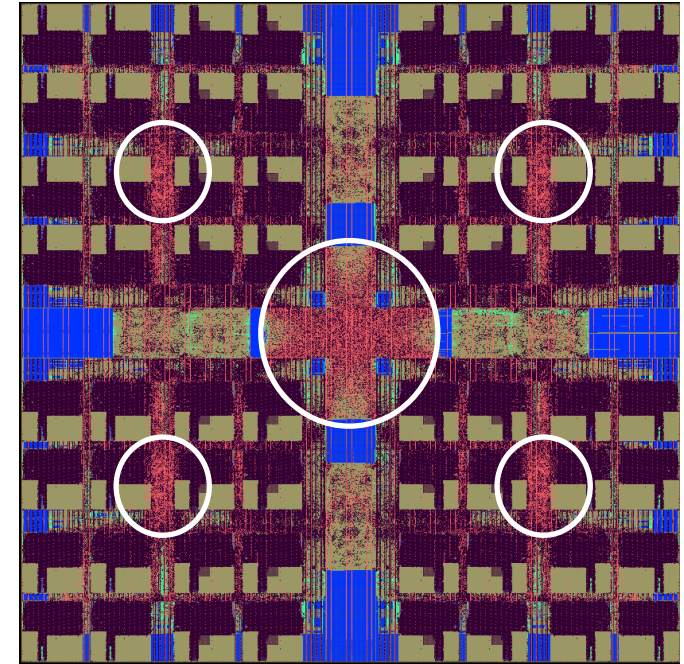


PnR of the MemPool cluster

- Top_4 : unfeasible (routing resources)
- Top_1 : feasible.
 - Cells scattered throughout the design.



- Top_H : 4.6 mm x 4.6 mm macro
 - 55% of the area is covered by the tiles
 - Five main congestion hot-spots



Power Analysis

- Extraction with PrimeTime 2019.12 at typical conditions (TT/0.80V/25°C)
- Activities extracted from running the matrix multiplication of two 64x64 matrices
- Each tile consumes 20.9 mW
 - Most of it by the I\$ (39.5%), the cores (26.6%) and the the L1 SPM (12.6%)
 - The interconnects only consume 1.7 mW, less than 10% of the total consumption
- MemPool consumes 1.55 W
 - The tiles are responsible for 86% of that
 - The global interconnect consumes 211 mW, only 14% of the total consumption

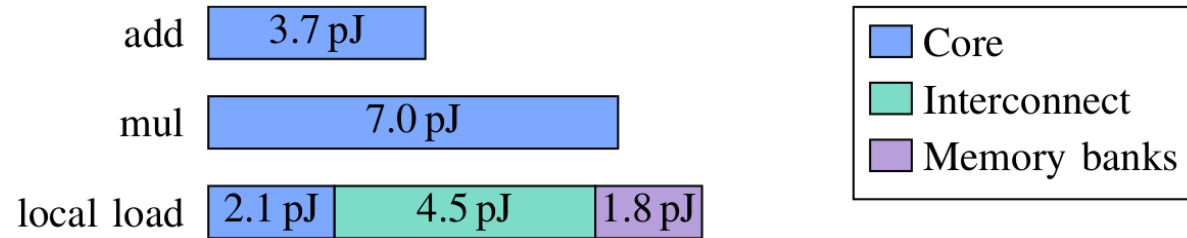
Energy consumption

- Breakdown of the energy consumption per instruction:



Energy consumption

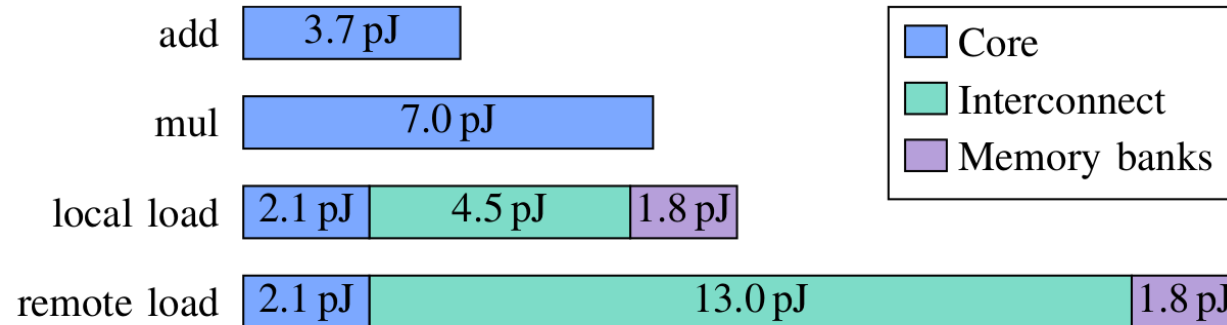
- Breakdown of the energy consumption per instruction:



- Local loads consume about as much energy as a *mul*
 - About half of it, 4.5pJ, by the interconnect

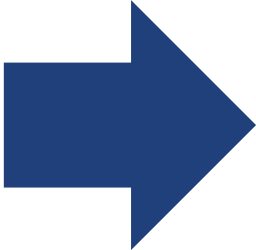
Energy consumption

- Breakdown of the energy consumption per instruction:



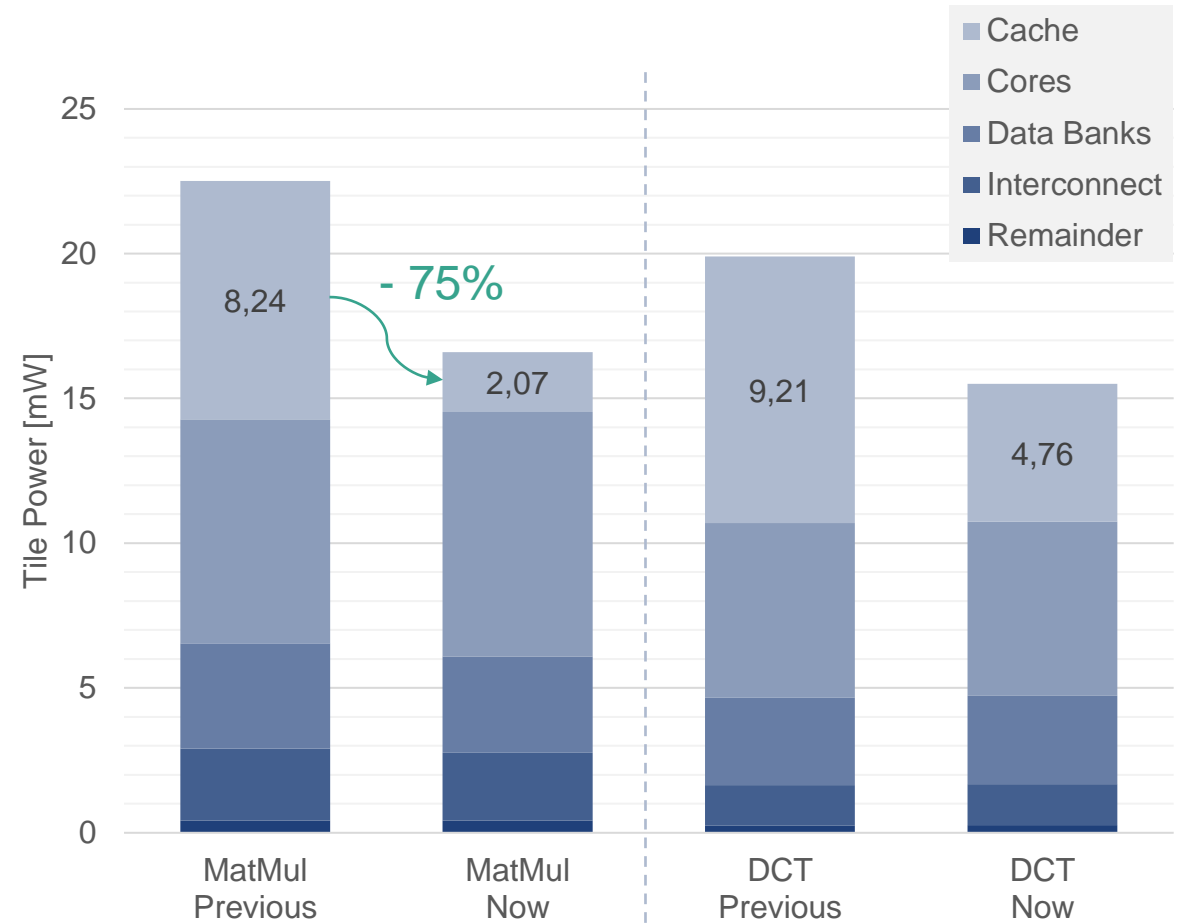
- Local loads consume about as much energy as a *mul*
 - About half of it, 4.5pJ, by the interconnect
- Remote loads consume 3x the energy of a local load
 - Despite crossing the whole cluster, twice!

Instruction cache optimization

- I\$ consumes almost 40% of the tile's power
→ Optimize and evaluate the cache architecture
 - Previous instruction cache
 - 2 KiB shared L1 I\$
 - 4-way set associative
 - Parallel lookup
 - SRAM based tag and data
 - 128-bit cache line width
 - Private L0 cache per core
 - 4 lines (16 instructions)
 - Register based
 - New instruction cache
 - 2 KiB shared L1 I\$
 - 2-way set associative
 - Serial lookup
 - Latch based tag and SRAM based data
 - 256-bit cache line width
 - Private L0 cache per core
 - 4 lines (32 instructions)
 - Latch based
- 

Instruction cache power breakdown

- Matrix multiplication now fits into L0
 - No fetching from L1
 - Cache's power reduced by 75%
 - Tile's power reduced by 5.9 mW
→ 377.6 mW for whole cluster

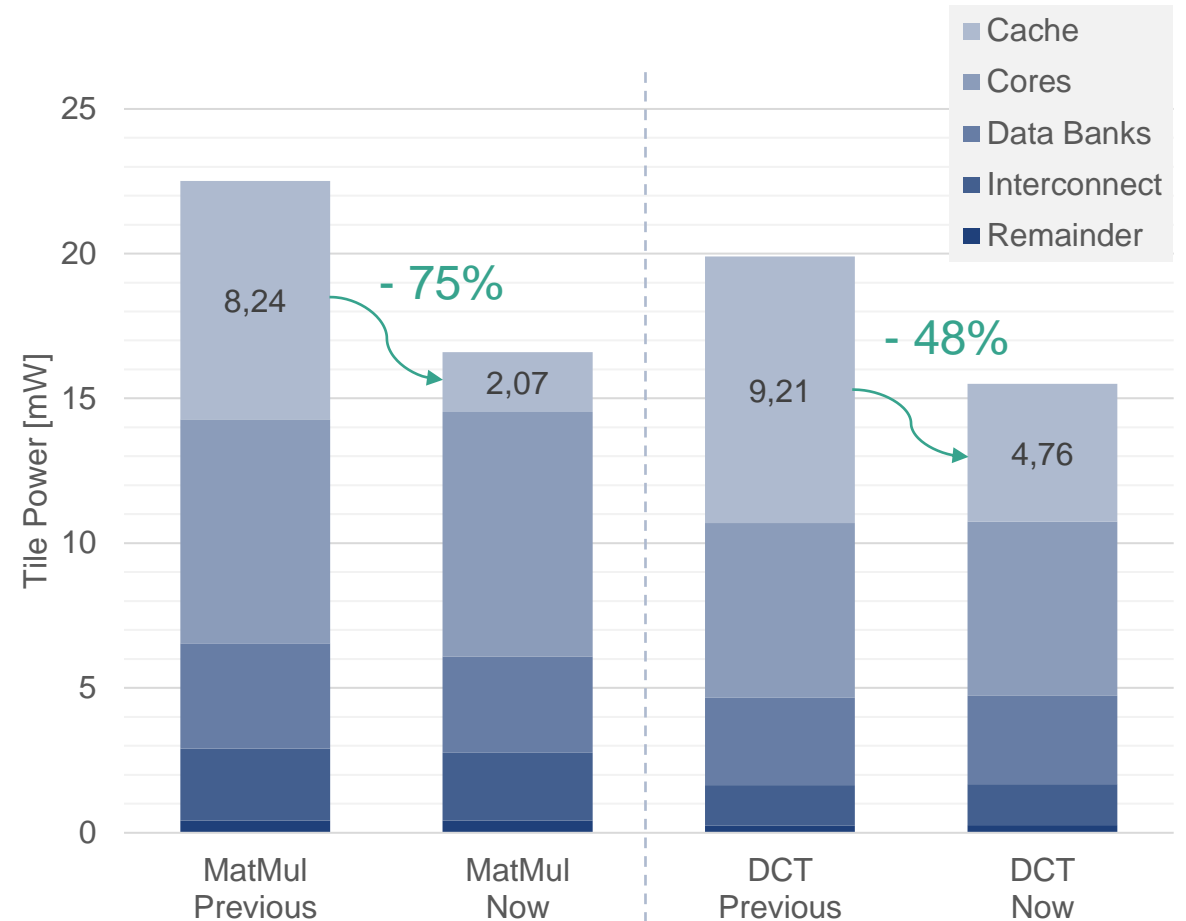


Results extracted from post-synthesis netlist at typical conditions (TT/0.80V/25°C) running at 500 MHz

Instruction cache power breakdown

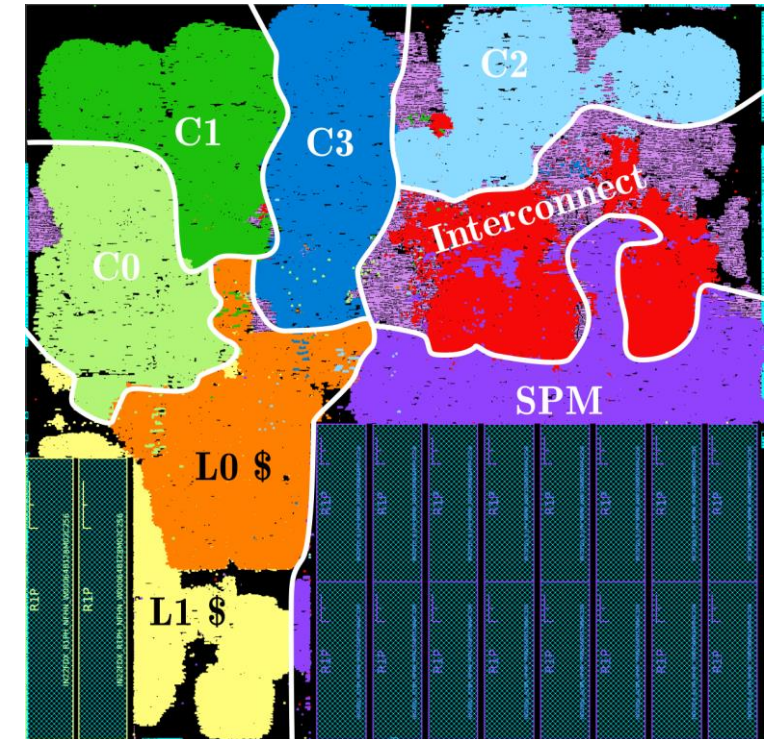
- Matrix multiplication now fits into L0
 - No fetching from L1
 - Cache's power reduced by 75%
 - Tile's power reduced by 5.9 mW
→ 377.6 mW for whole cluster
- DCT does not fit into L0
 - Continuous pre-fetching from L1
 - Cache's power reduced by 48%
 - Tile's power reduced by 4.4 mW
→ 281.6 mW for whole cluster

Results extracted from post-synthesis netlist at typical conditions (TT/0.80V/25°C)
running at 500 MHz



Instruction cache summary

- Architecture
 - Cache line width doubles → L0 size doubles
 - L1 size remains constant, 4-way → 2-way associative, parallel → serial lookup
 - Reducing the number of banks from four to two
- Reduce cache's area
 - 149 kGE → 123 kGE
- Significant power improvement
 - 5.9 mW if the kernel fits into L0
 - 4.4 mW if the kernel does not fit into L0
- Application runtime changes marginally



Conclusion: MemPool, where do things currently stand?

- Top_H currently working at 700 MHz, in typical conditions, using GF 22FDX technology
 - 480 MHz in worst-case conditions
- Performance is at 80% of the performance of a unfeasible baseline system
 - Considering three key benchmarks
 - The scrambled addressing scheme helps on power consumption, energy, performance, ...
- Remote loads consume only 3x the energy of a local load
 - Not a hefty price at all
 - The consumption of the remote loads is only 4.5x the energy consumption of an *add...*
 - At a highly-efficient ultra-small core

Next steps: MemPool, what now?

- Ongoing convergence of many work packages:
 - Snitch is being incremented with DSP instructions
 - SoC infrastructure is being built around MemPool
 - FPGA deployment of MemPool as the target of a Semester Project currently underway
- Tape-out of a proof-of-concept "MinPool" system at the Spring 2021 semester
 - Objective: the highest core-count of any of the previous PULP systems so far
 - But not at the 256-core MemPool level
 - Looking for students at ETHZ