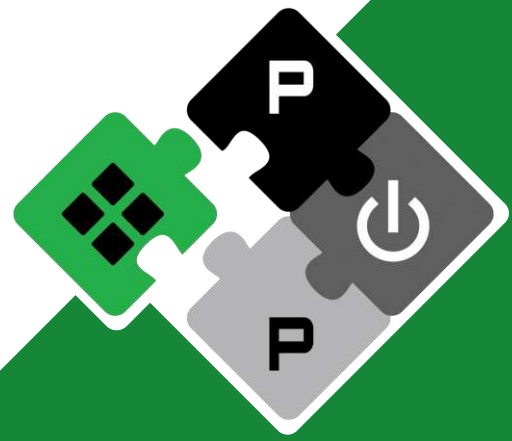
ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fast Shared-Memory Barrier Synchronization for a 1024-Cores RISC-V Many-Core Cluster

Integrated Systems Laboratory (ETH Zürich)

Marco Bertuletti mbertuletti@iis.ee.ethz.ch
Samuel Riedel sriedel@iis.ee.ethz.ch
Yichao Zhang yiczhang@iis.ee.ethz.ch
Alessandro Vanelli-Coralli avanelli@iis.ee.ethz.ch
Luca Benini lbenini@iis.ee.ethz.ch



@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

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

Many-Core Clusters

Many-Cores are more and more popular:

- They serve **large parallel workloads** (e.g. image-processing, genomics, deep-learning, telecommunications…)
- They are built assembling loosely-coupled clusters

	NPE	L1 size	PE/L1	L2 size	PE/L2
[1]Intel X86 KNL	72	32 KiB	1	1 MiB	32 (**Tile**)
[2]Esperanto ET-Soc-1	1088	1 KiB	1	4 MiB	32 (**Shire**)
[3]Tenstorrent Aegis	128	12 MiB	8	X MiB	256 (**Cluster**)
[4]Ramon RC64	64 DSP	4 MiB	64	X MiB	64 (**Cluster**)
[5]NVIDIA H-100 GPU	128 FP32 x 144 SM	256 KiB	128	60 MiB	128 (**SM**) x 144

[1] https://ark.intel.com/content/www/us/en/ark/products/95830/intel-xeon-phi-processor-7290-16gb-1-50-ghz-72-core.html
[2] https://www.esperanto.ai/wp-content/uploads/2022/05/Dave-IEEE-Micro.pdf
[3] https://tenstorrent.com/risc-v/
[4] https://indico.esa.int/event/182/contributions/1510/attachments/1398/1623/1240_-_Ginosar.pdf
[5] https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Many-Core Clusters... with L1 Shared-Memory

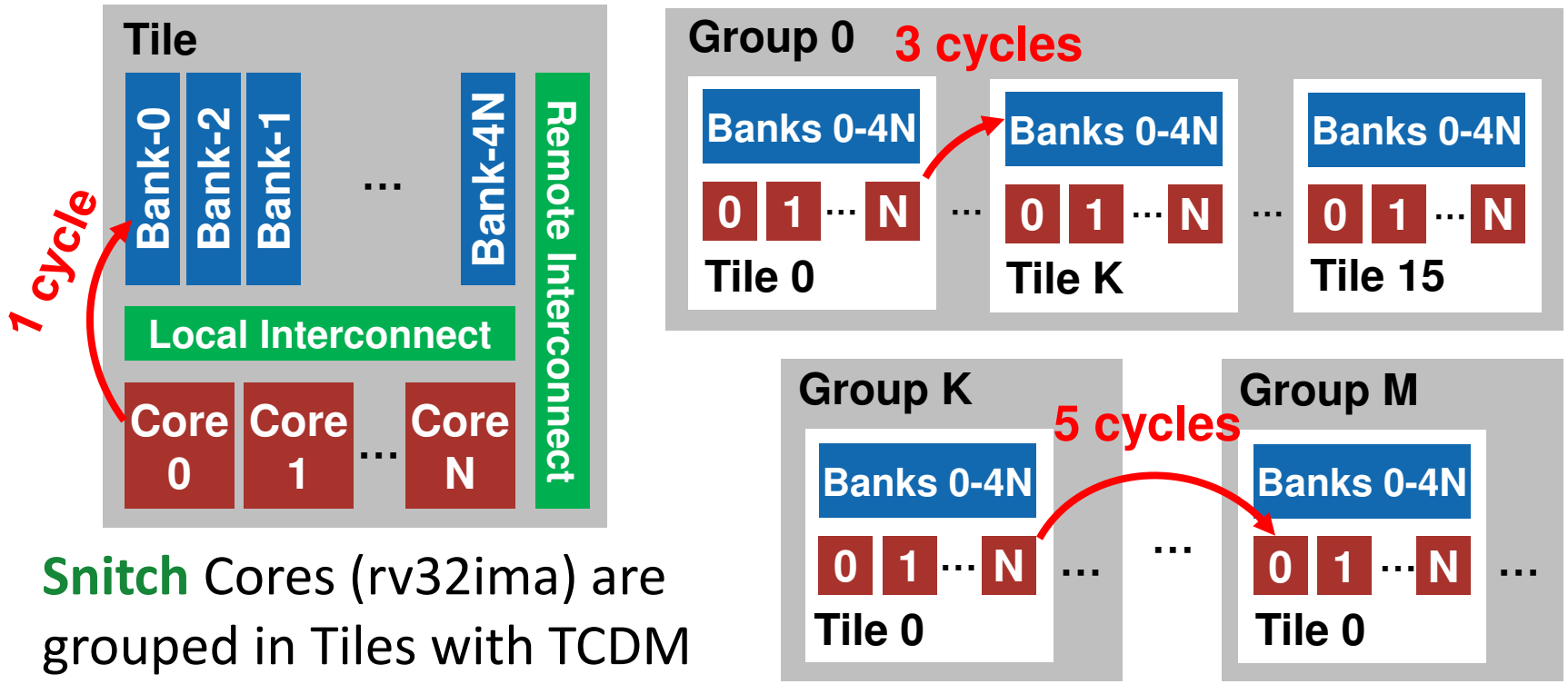
Having a large shared L1 is beneficial:

- Better global **latency tolerance** if $L1_{size} > 2*L2_{latency}*L2_{bandwidth}$ (Little's law + double buffer)
- Easier to program (data-parallel, functional pipeline…)
- Smaller data partitioning overhead

TeraPool: 1024-Cores Approximation of PRAM

MemPool → 256 cores

TeraPool → 1024 cores



Snitch Cores (rv32ima) are grouped in Tiles with TCDM

Any core can access any bank, some interconnection resources are shared → **NUMA**

- 1 cycle in Tile
- 3 cycles in Group
- 5 cycles to other Group (in the hypothesis of scaling-up TeraPool from MemPool)

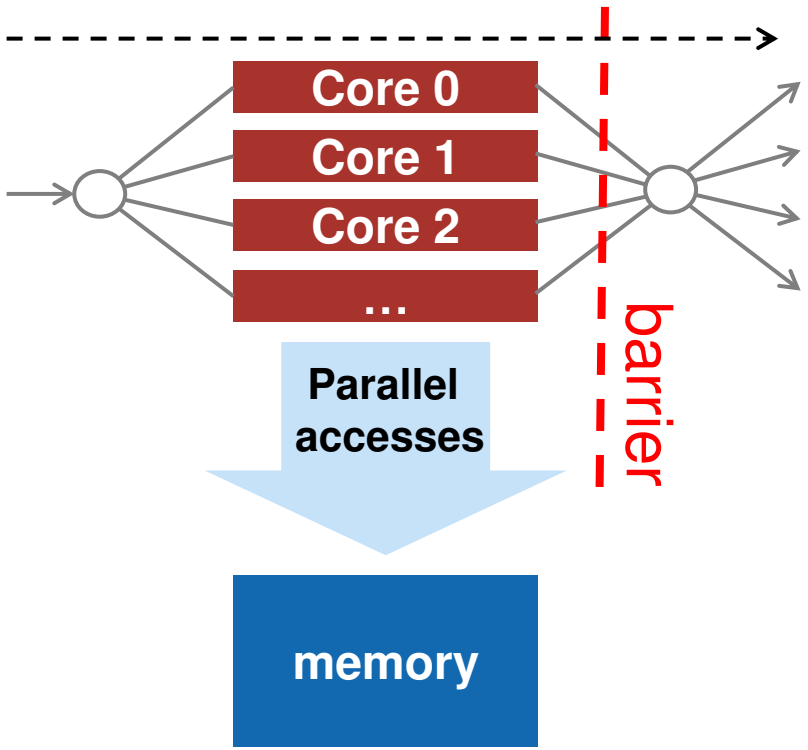
https://arxiv.org/abs/2303.17742

We target fork-join parallelism

Fork-join programming model

- Parallel execution
- Cores access memory concurrently
- Cores are synchronized and a new parallel execution can start

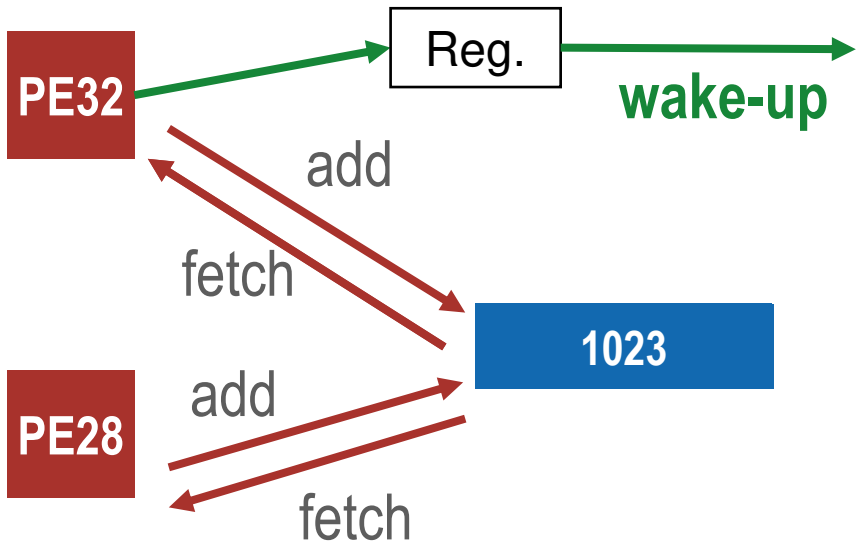
Can we push L1 shared-memory synchronization to a 1024-cores cluster?

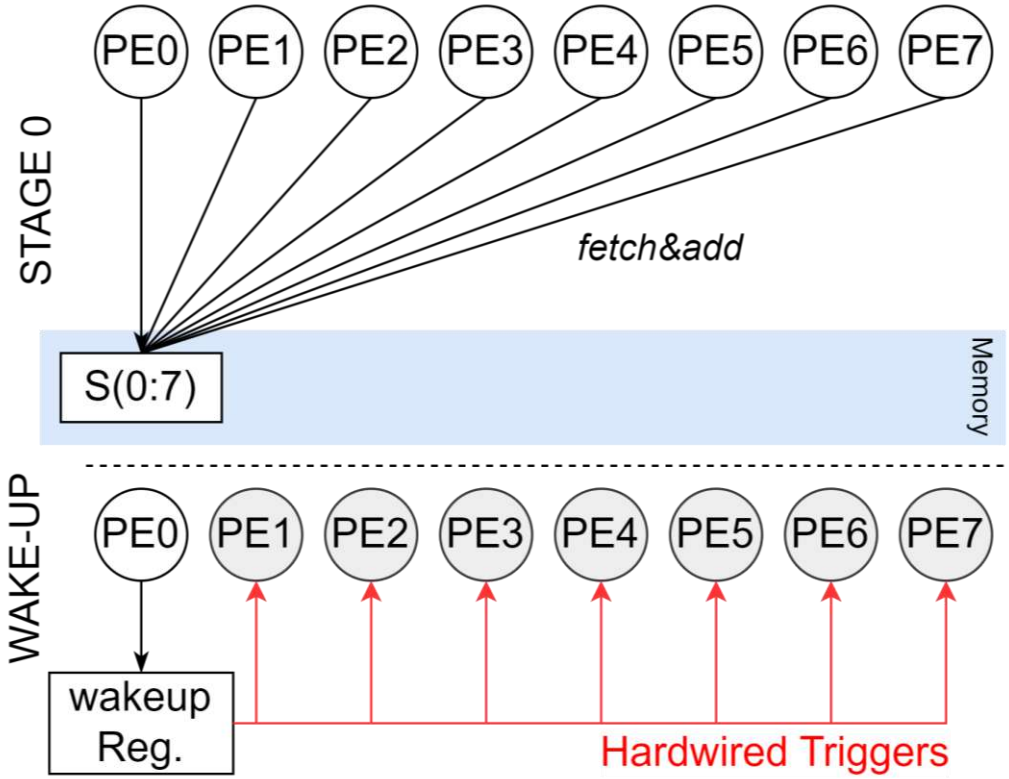


Shared Memory Synchronization

Synchronization barriers

- Arrival = atomic writes to a synch variable
- Hardwired **wake-up triggers** for departure



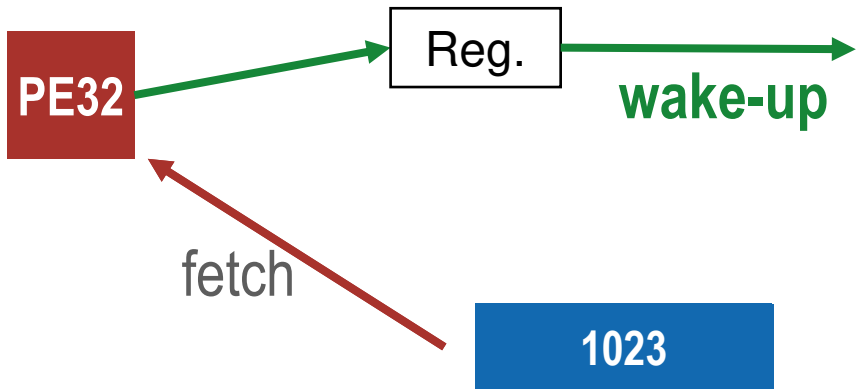


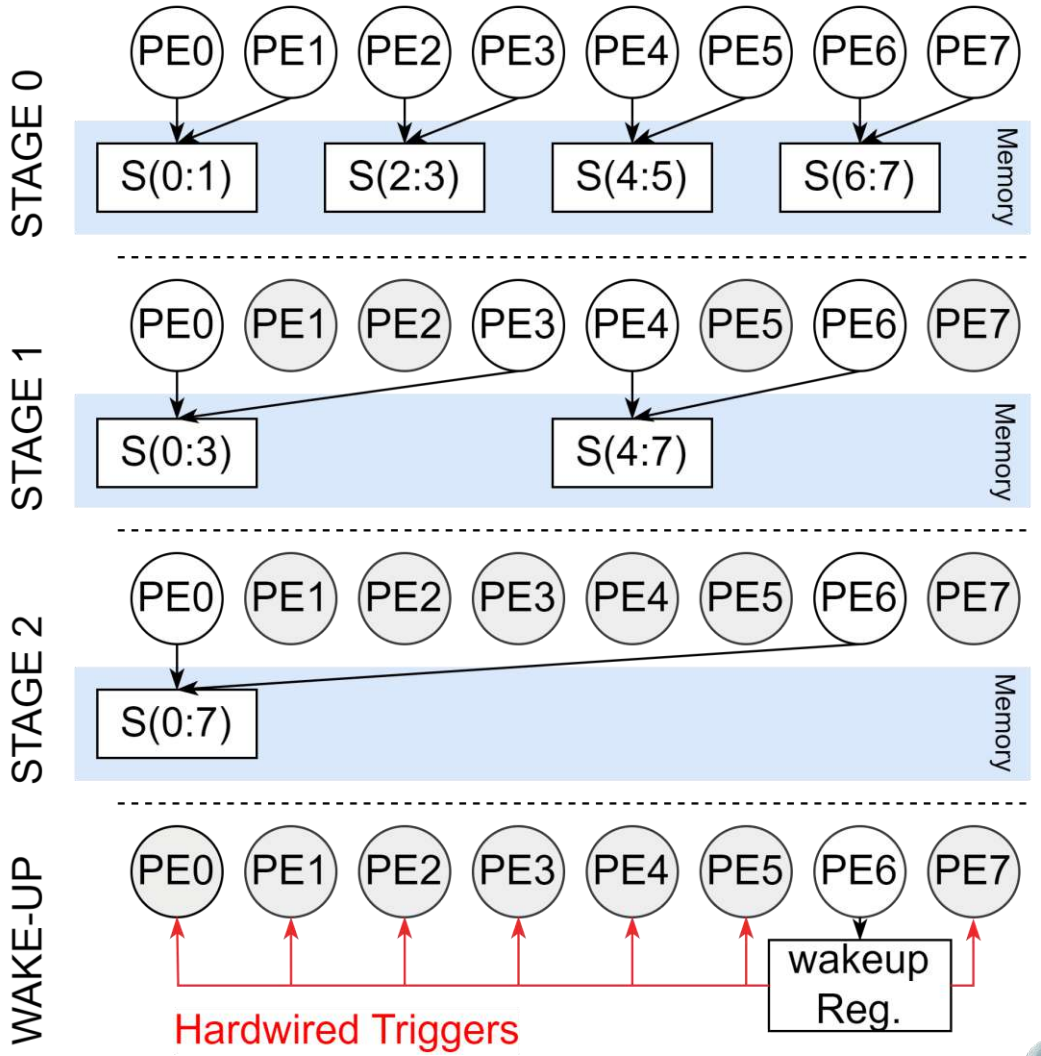
ISSUE: PEs arriving all together will contend for the same memory resource!

Shared Memory Synchronization

Synchronization barriers

- Arrival = atomic writes to a synch variable
- Hardwired **wake-up triggers** for departure





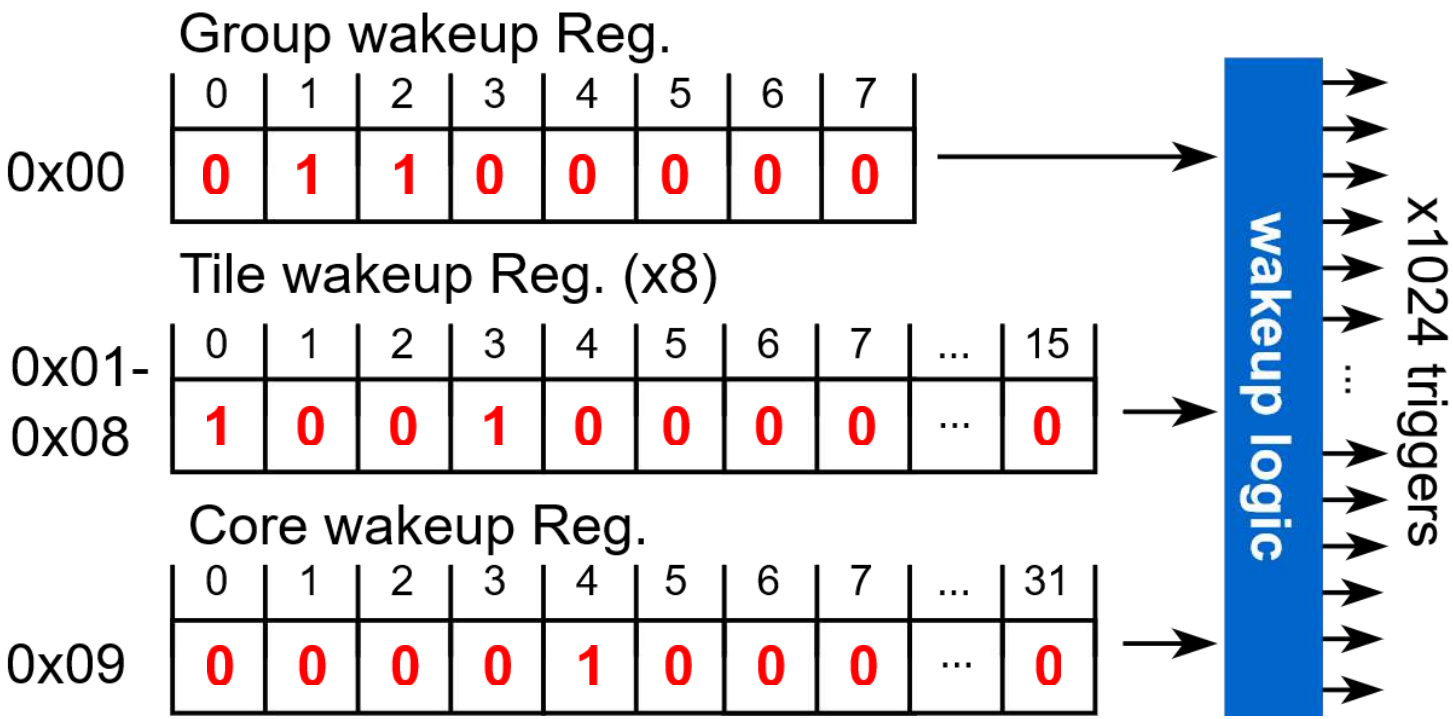
ISSUE: PEs arriving all together will contend for the same memory resource!

Fast Partial Synchronization of Cores

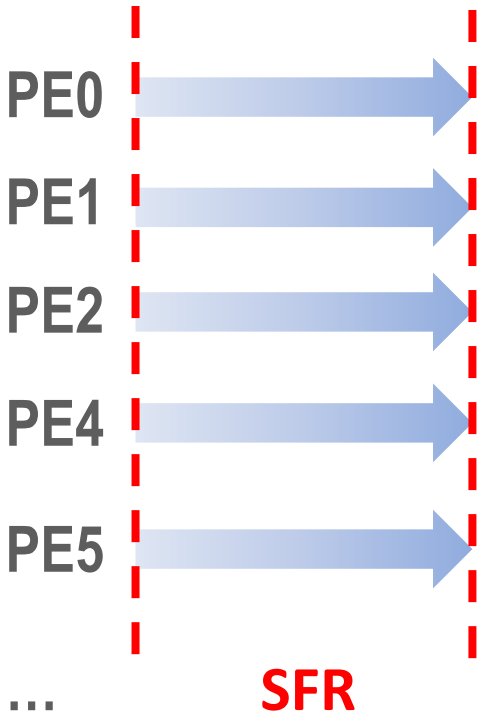
- Arrival = narrower tree of atomics
- Departure = assert subsets of the wake-up triggers →
- write a **bit-mask** in wakeup registers:
  - 1 Group wakeup register
  - 8 Tile wakeup register

PE

```
sw 0x0006 0x00 // wakeup->G1,G2
sw 0x0009 0x01 // wakeup->T0,T3 in G0
sw 0x0010 0x09 // wakeup->PE16
```

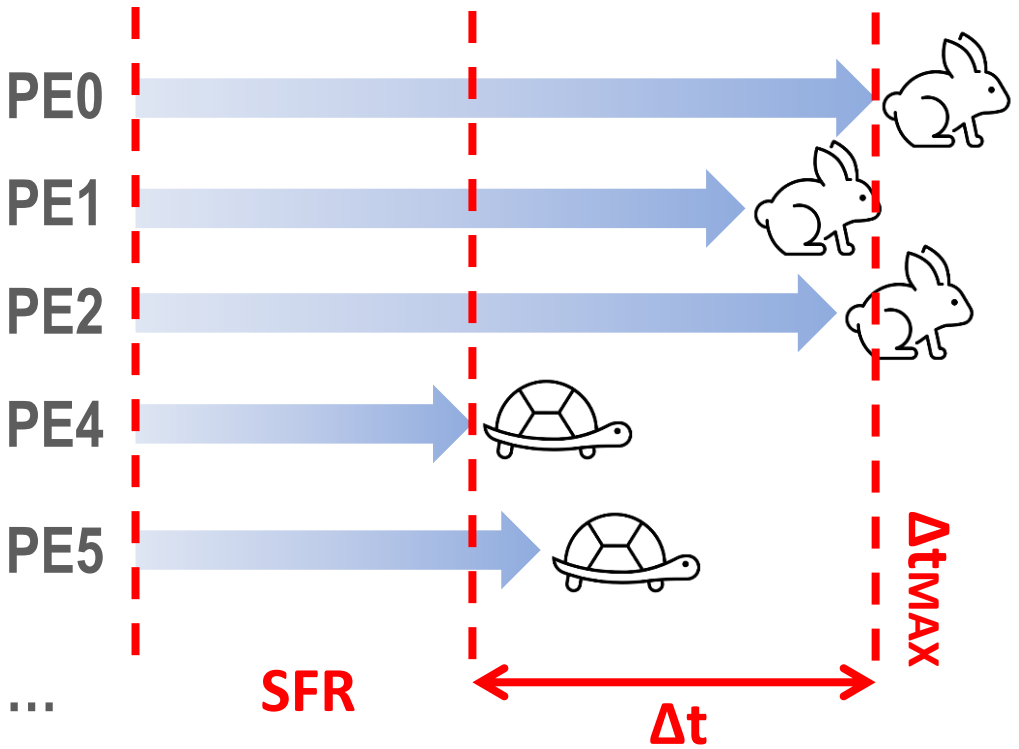


Benchmarking performance on PE's arrival time



- PEs run in parallel through a **synchronization-free region**

Benchmarking performance on PE's arrival time



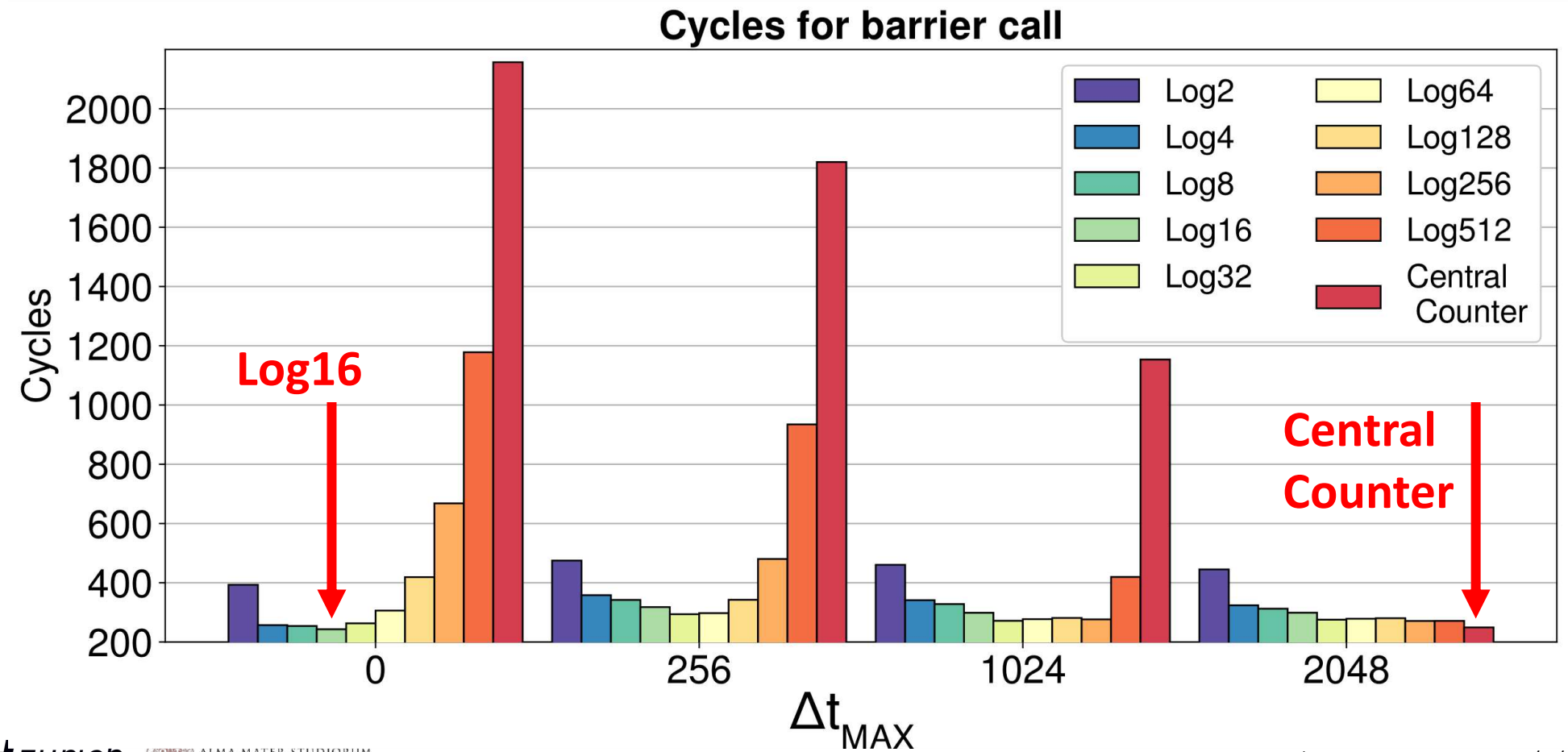
- PEs run in parallel through a **synchronization-free region**
- PEs are assigned a random delay:

$$\Delta t \in [0, \Delta t_{MAX}]$$

- There is **scattering in the arrival time** of the PEs to the barrier synchronization

Best radix depends on arrival time

- Small delays → small radices are better
- Large delays → central-counter wins



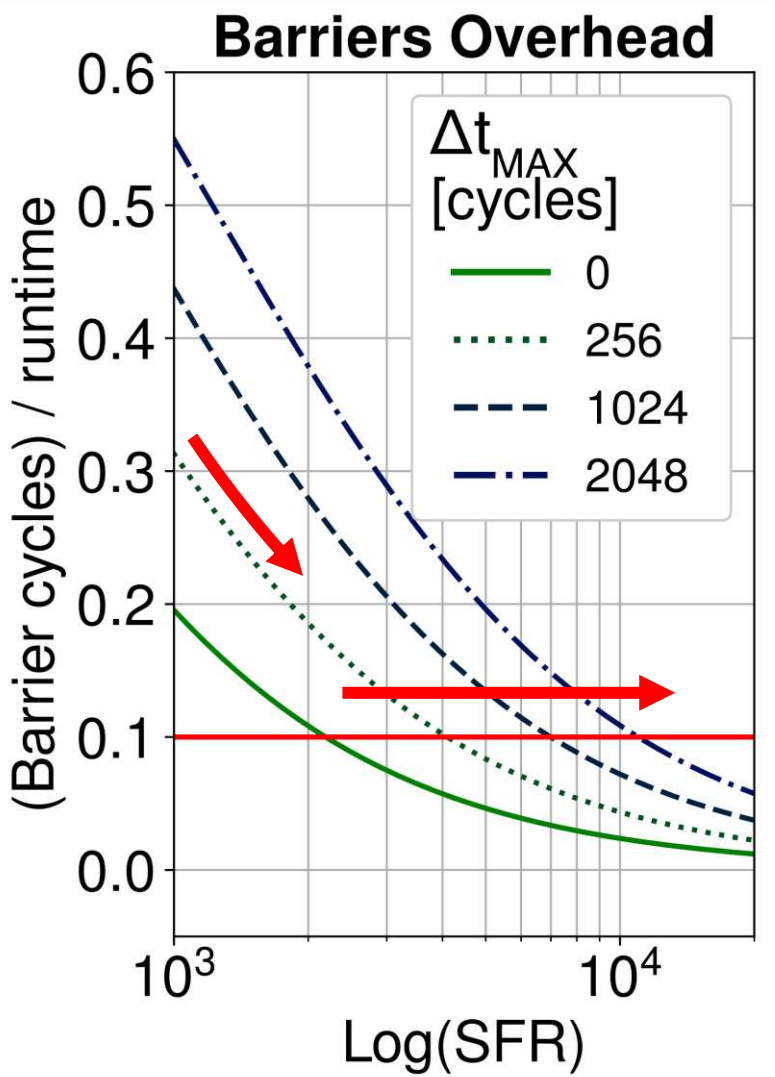
Kernel runtime for a 10% overhead?

(++ Length of SFR) → Less barrier overhead

(++ Scattered PEs) → Longer SFR for 10% overhead

SFR=**$2x10^3$** cycles **@Δt_{MAX} = 0** (Log16)

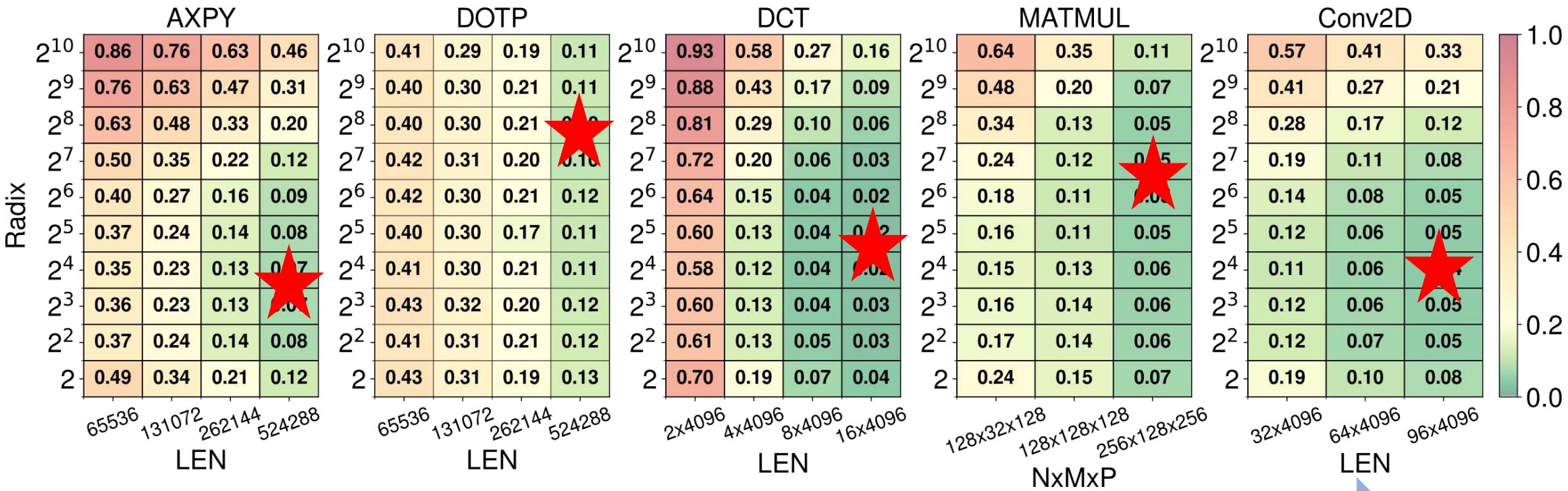
SFR=**10^4** cycles **@Δt_{MAX} = 2048** (C. Counter)



Choose the best barrier-radix depending on the kernel!

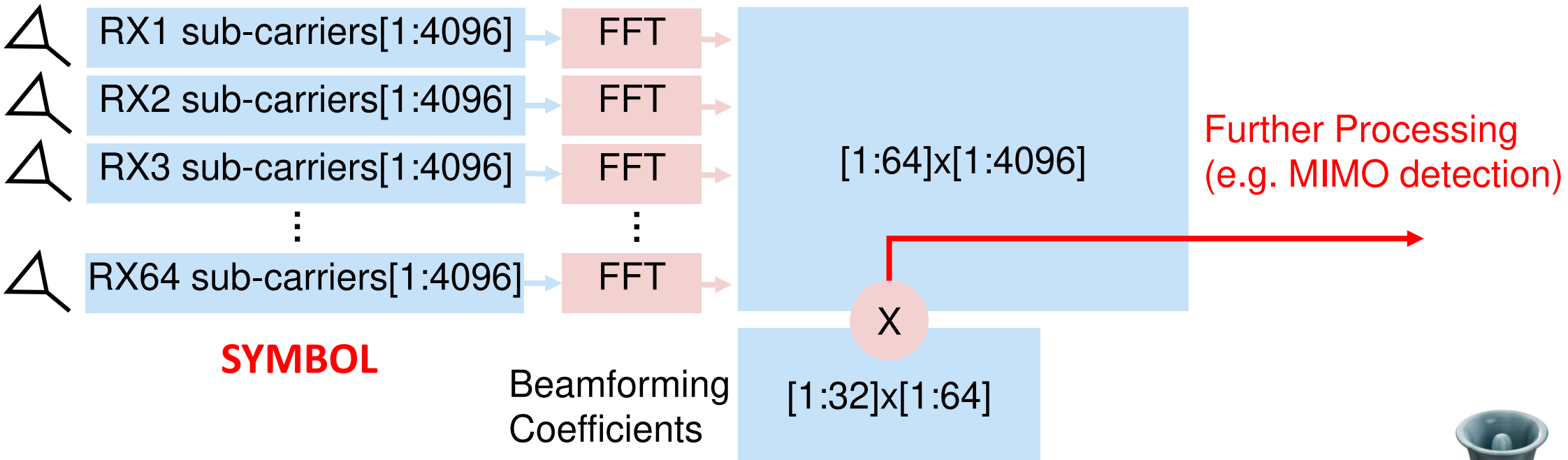
- Increasing the problem-size the overhead reduces
- Sweet spot depends on the arrival time of PEs

Runtime **synchronization overhead** between **2-10%**

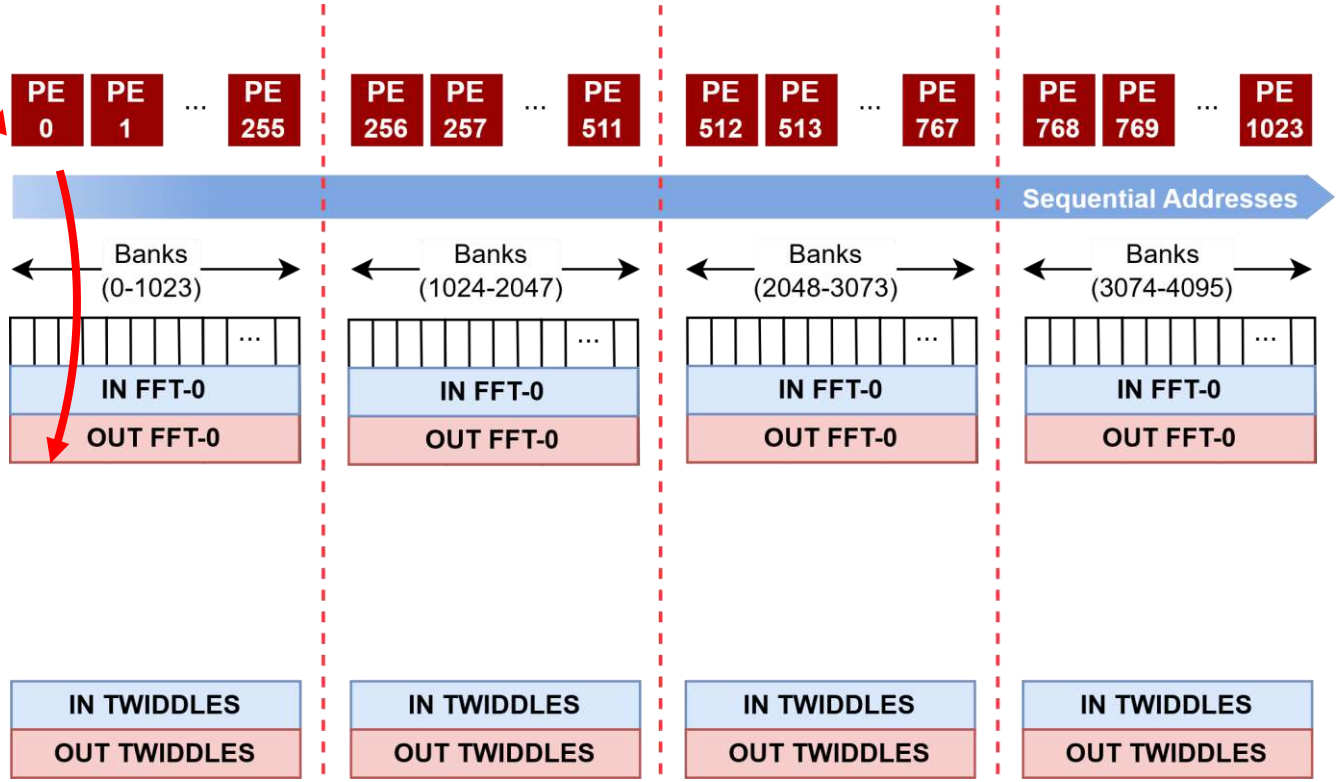


Test on a 5G critical processing: PUSCH demodulation

- **Physical Uplink Shared Channel** (requires on the fly processing)
- **OFDM demodulation** = **FFT** on large number of samples
- **Beamforming** = Change of coordinates (from the antennas coordinate space to the beams coordinate space) → **Matrix-Matrix Multiplication**



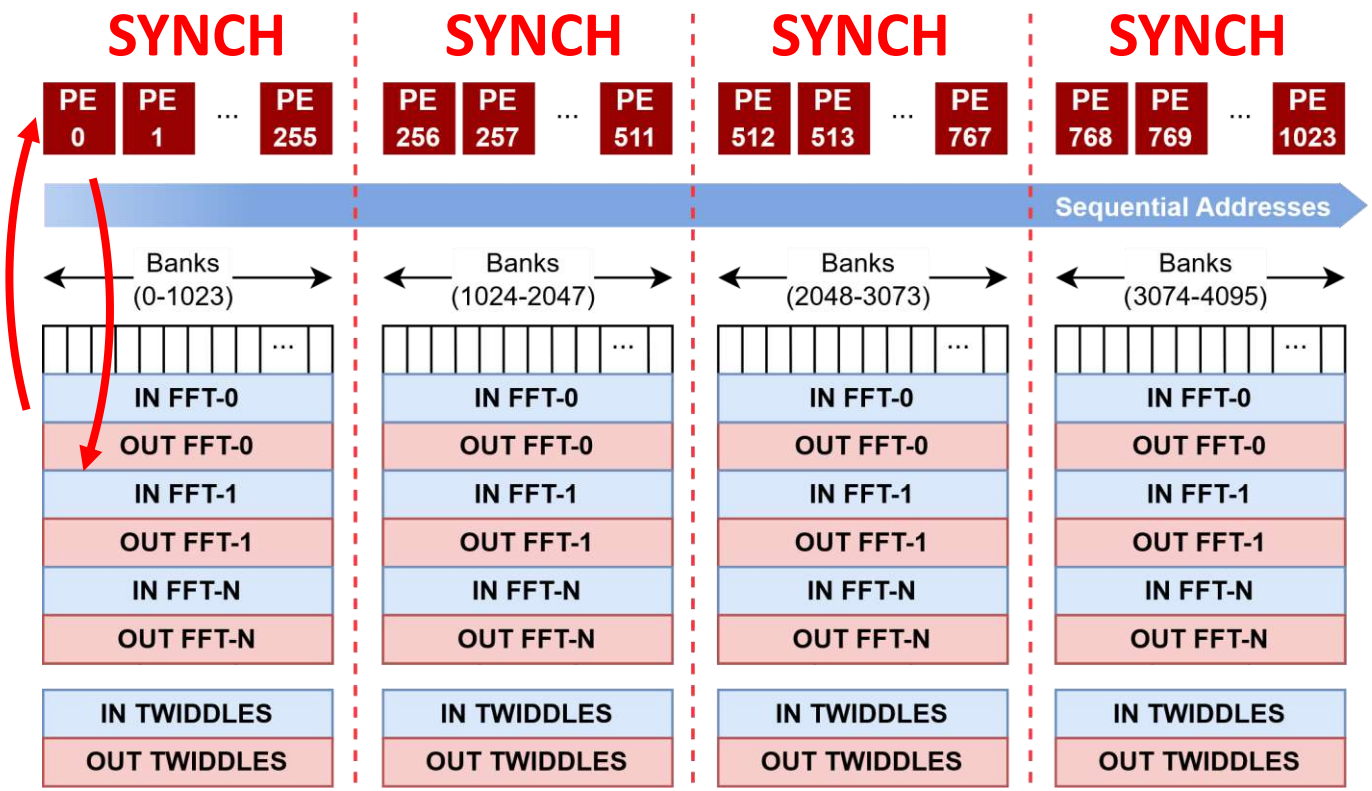
Test on a 5G critical processing: PUSCH demodulation



Radix-4 Cooley-Turkey FFT (multi-stage synchronization kernel):

- A 4096-points FFT is assigned to 256 cores:
  - Load from local memory
  - Store in local memory of PEs using the data next

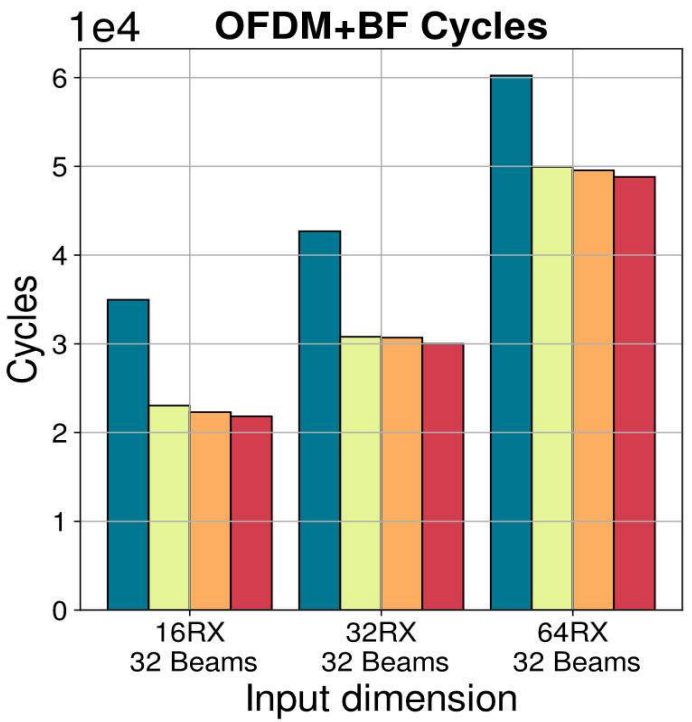
Test on a 5G critical processing: PUSCH demodulation

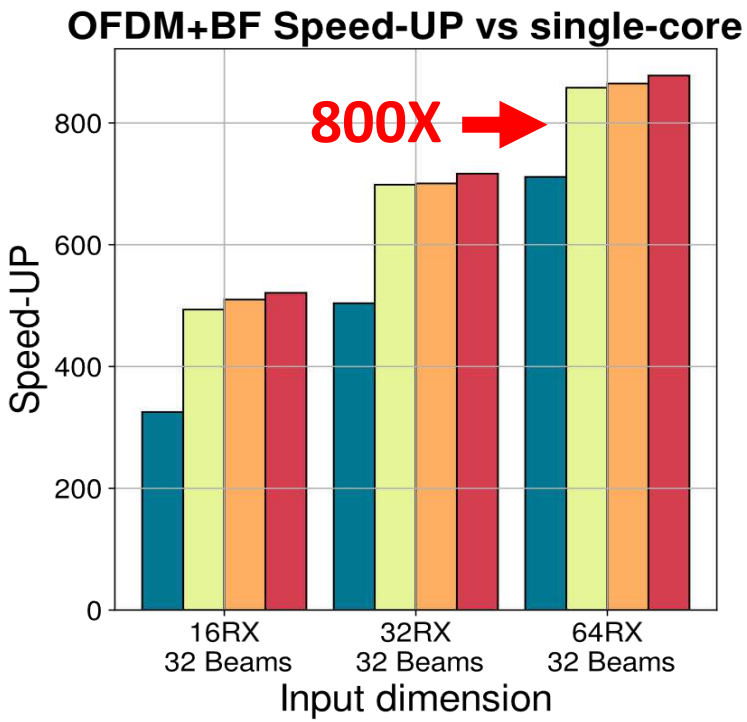


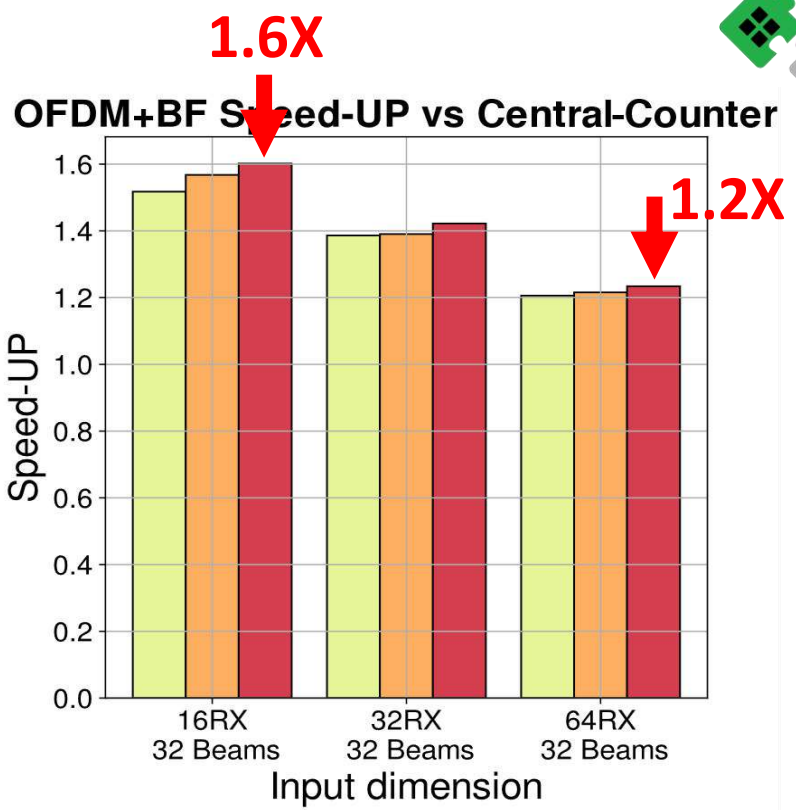
Radix-4 Cooley-Turkey FFT (multi-stage synchronization kernel):

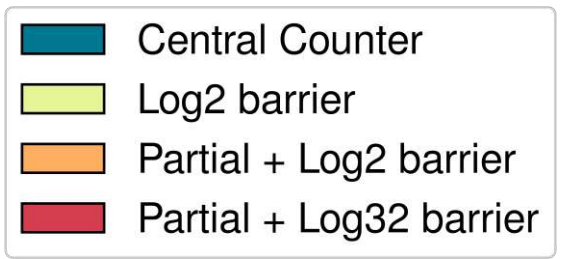
- A 4096-points FFT is assigned to 256 cores:
  - Load from local memory
  - Store in local memory of PEs using the data next
- Synchronization between stages → PEs execute the same stage for independent FFTs, to minimize synchronization overhead

Large Speed-UP and small overhead









< 50k-cycles for 1 symbol

Speed-UP > 800X vs single-core execution

1.6X – 1.2X Speed-UP of fine-tuning vs Central-Counter

Conclusions

In the PRAM-like 1024-cores **TeraPool** many-core cluster the **barrier selection is an important phase of kernel optimization!**

Choosing the best barrier-radix + hardware support for a fast departure from synchronization gives:

- A **synch. overhead of 2%-10%** on kernels for signal-processing
- **6.2% synch. overhead** on **5G PUSCH demodulation**

`github.com/pulp-platform/mempool`

PULP Platform

Open Source Hardware, the way it should be!

Marco Bertuletti mbertuletti@iis.ee.ethz.ch
Samuel Riedel sriedel@iis.ee.ethz.ch
Yichao Zhang yiczhang@iis.ee.ethz.ch
Alessandro Vanelli-Coralli avanelli@iis.ee.ethz.ch
Luca Benini lbenini@iis.ee.ethz.ch

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI – Universitá di Bologna
Viale del Risorgimento 2
Bologna, Italy

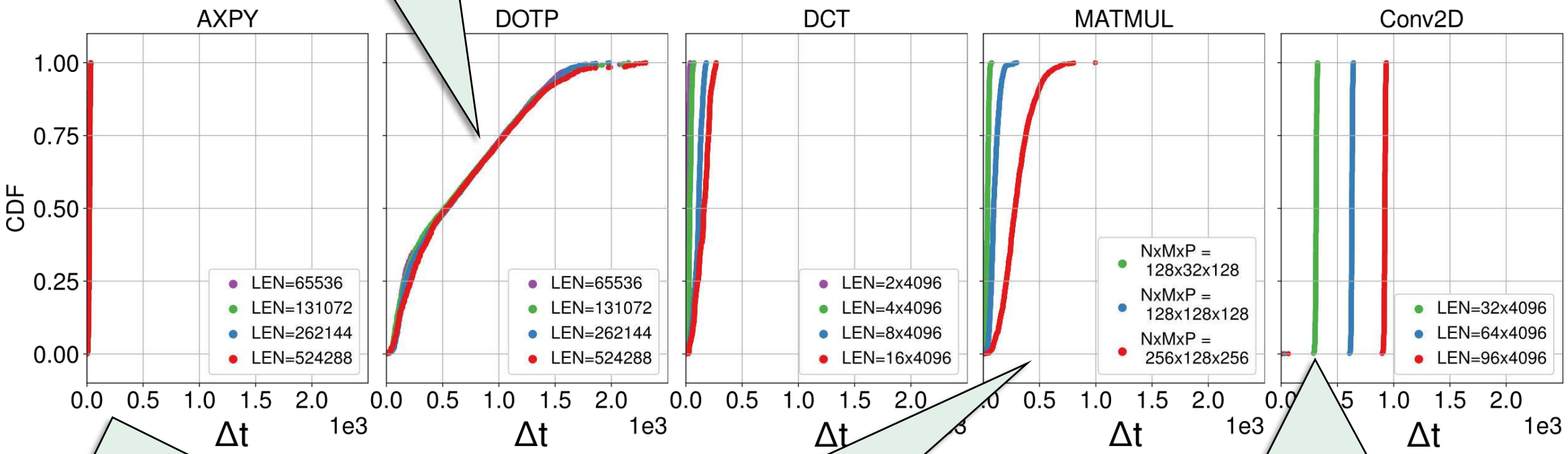
@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

Arrival time of PEs is a property of the kernel

Delay due to **reduction**



No **memory access contentions** → PE' arrival time is the same

DCT and MATMUL larger scattering depending on the **input dimension**

Work-imbalance → offset between PEs