NOC-BASED SUPPORT OF HETEROGENEOUS CACHE-COHERENCE MODELS FOR ACCELERATORS

Davide Giri
Paolo Mantovani
Luca P. Carloni

Columbia University
New York, USA

ACM/IEEE NOCS 2018
Torino, Italy

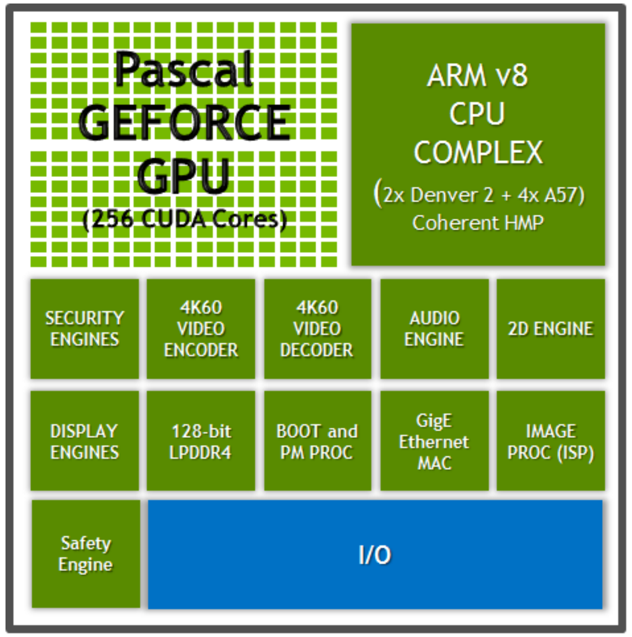
SOC TRENDS

- Heterogeneity
  - **Custom accelerators**
- NoC
- Shared memory

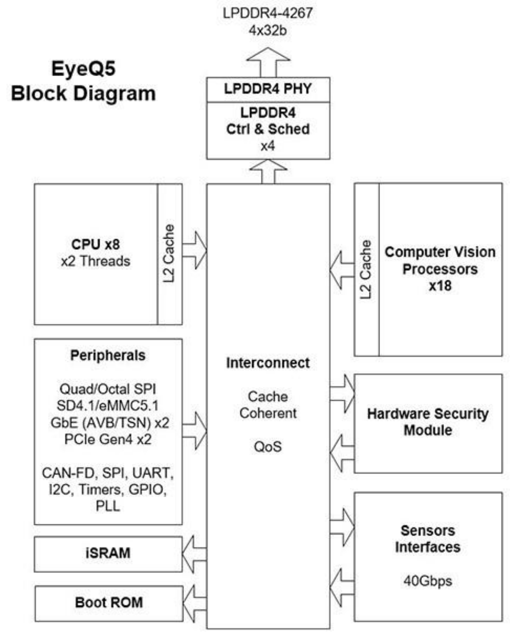
Challenges

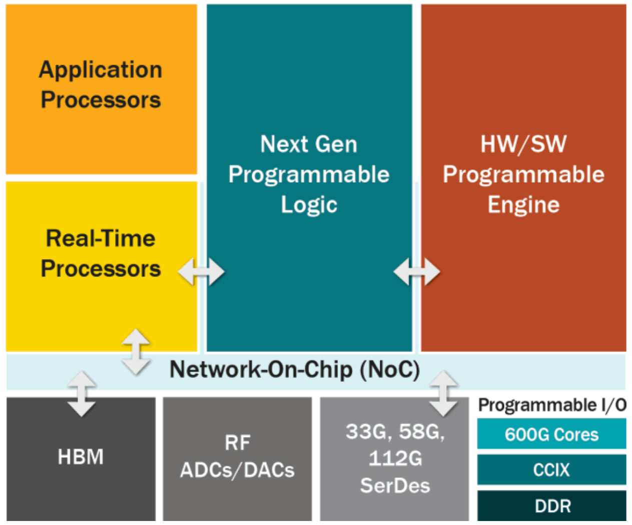
- Scalability
- Programmability

NVIDIA Parker, 2016.

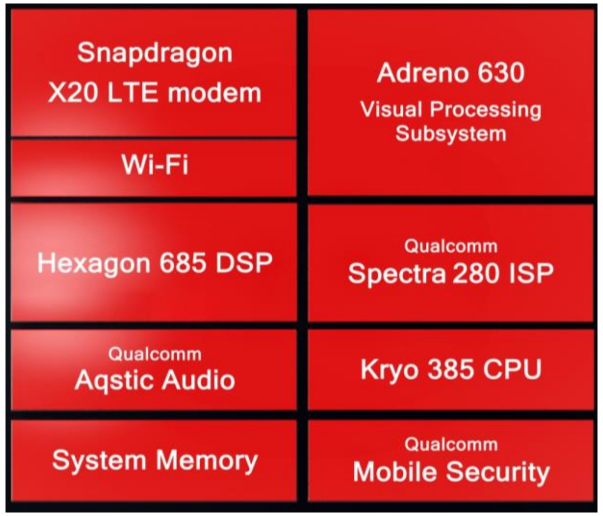


Mobileye EyeQ5, 2020.





Xilinx Everest, 2018.



Qualcomm Snapdragon 835, 2017.

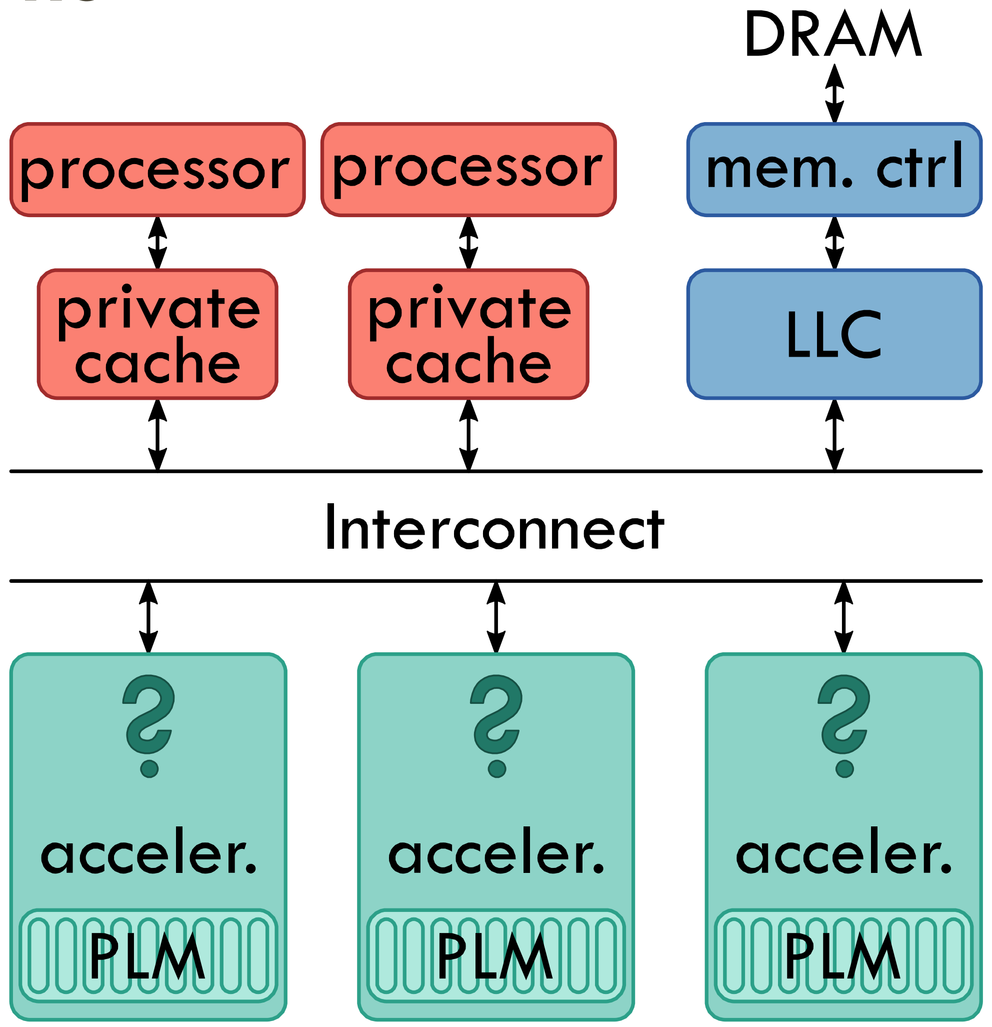
LOOSELY-COUPLED ACCELERATORS

Major speedups and energy savings:

- Highly parallel and customized datapath
- Aggressively banked private local memory (PLM)

What should the cache coherence model for accelerators be?

- We identified 3 main models in literature



ACCELERATOR MODELS: FULLY COHERENT

Coherent with entire cache hierarchy

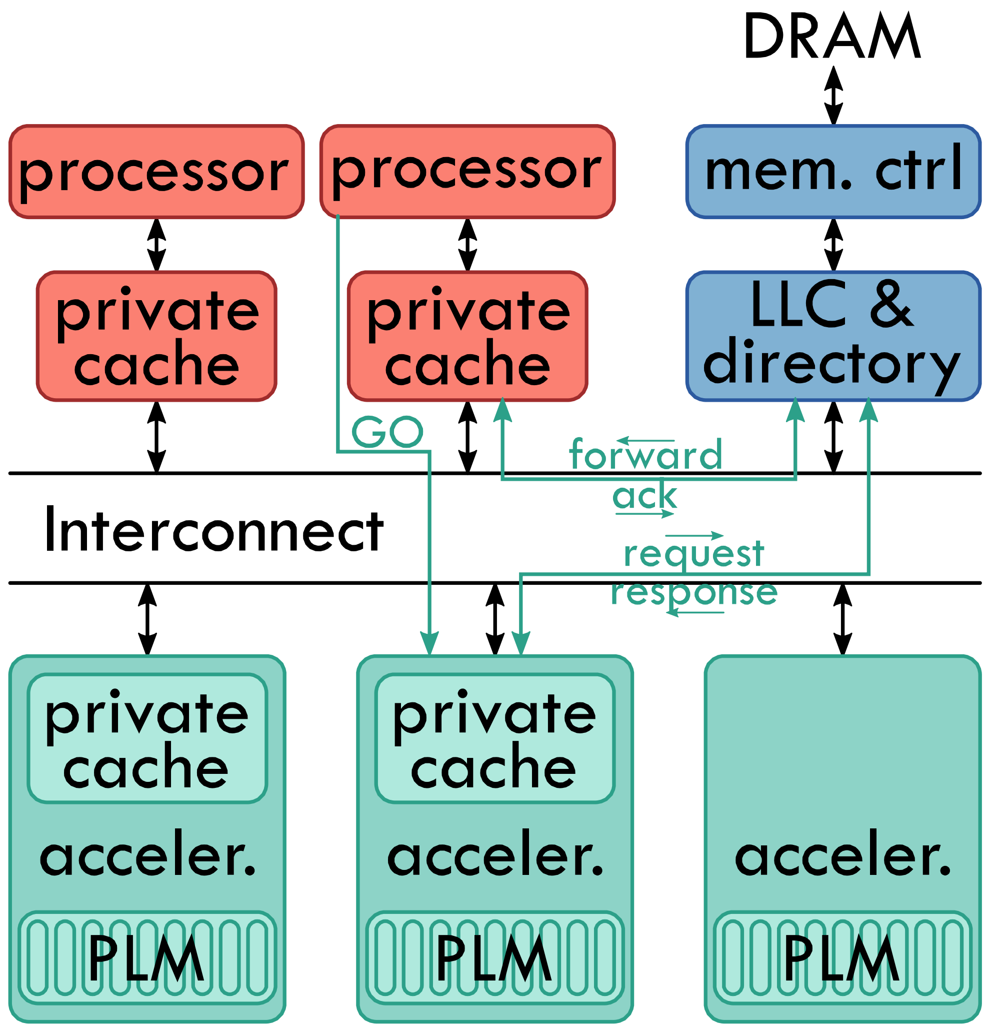
- Same coherence model as the processor

Programming requirements

- Race free accelerator execution

Implementation variants

- Generally bus-based
- Accelerators may own a cache
  - ✓ IBM CAPI, [Y. Shao et al., MICRO '16], [M. J. Lyons et al., TACO '12]
  - ✗ ARM ACE-lite



ACCELERATOR MODELS: NON COHERENT

Not coherent with cache hierarchy

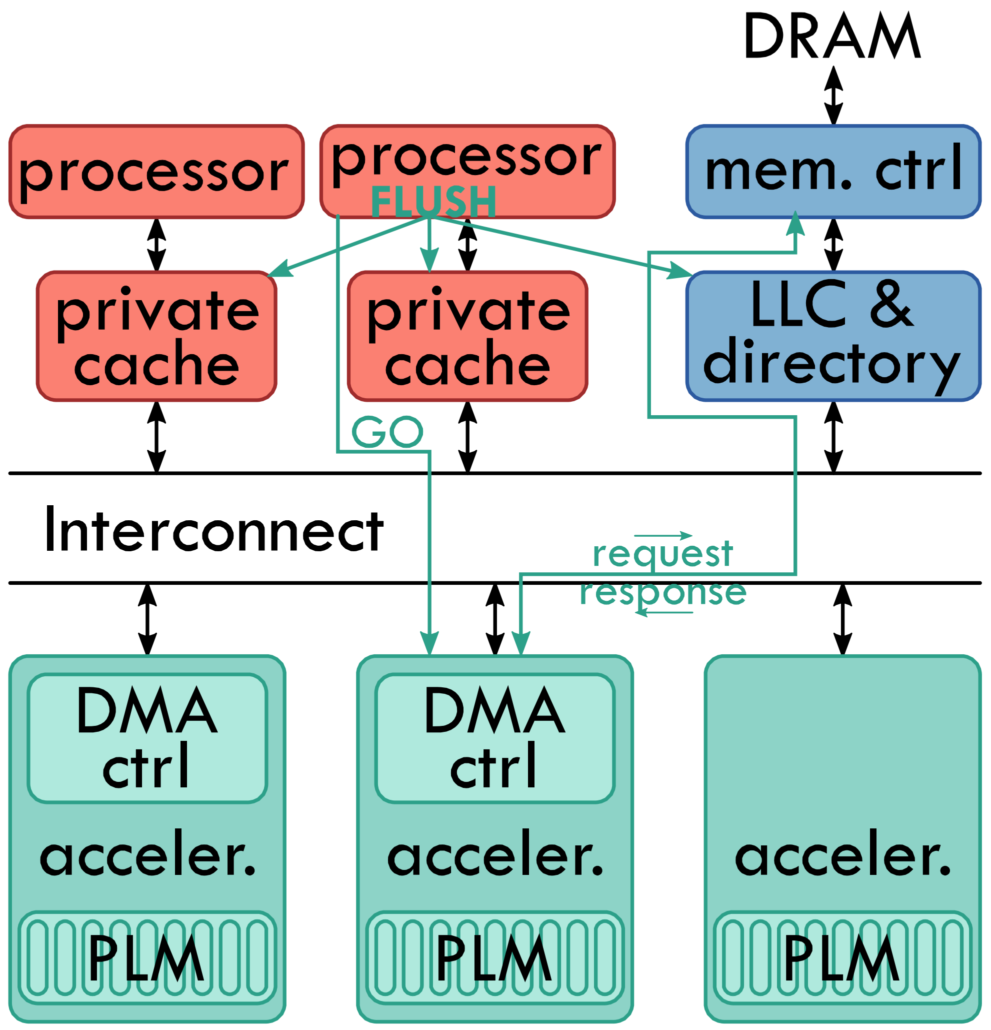
- Caches are by-passed

Programming requirements

- Race free accelerator execution
- Flush all caches prior to accelerator execution

Implementation variants

- Generally NoC-based and DMA-based
  - [Y. Chen et al., ICCD '13], [E. Cota et al., DAC '15] [Y. Shao et al., MICRO '16]



ACCELERATOR MODELS: LLC COHERENT

Coherent with LLC only

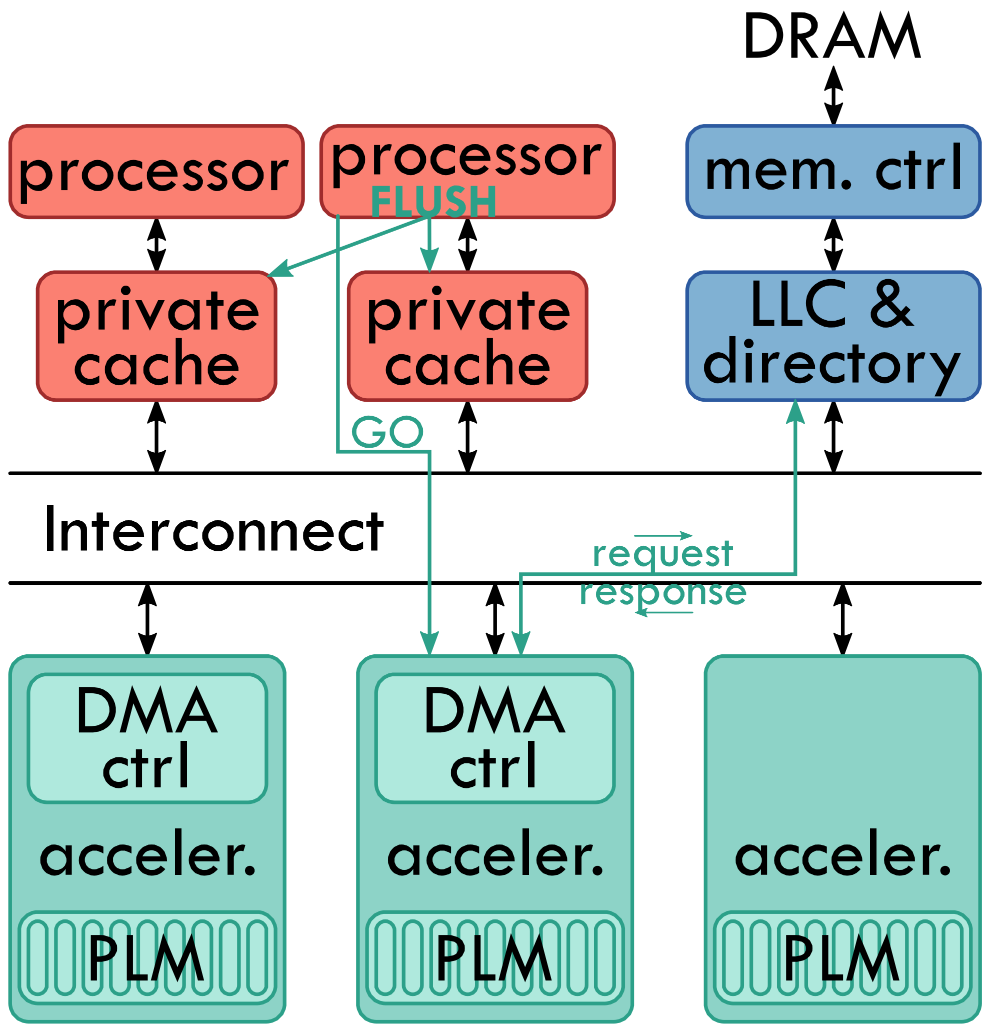
- Processors' private caches are by-passed

Programming requirements

- Race free accelerator execution
- Flush processors' private caches prior to accelerator execution

Implementation variants

- No implementation in literature
- First proposed by [E. Cota et al., DAC '15]



CONTRIBUTIONS

Protocol.

- Variation of MESI to support 3 coherence models for accelerators (NoC-based)

Coherence Models.

- Show how each model can outperform the others in some cases
- Show that the best choice of model varies at runtime

Architecture. Design of a multi-core NoC-based architecture that supports:

- Three models of coherence for accelerators
- Run-time selection of the coherence model for each accelerator
- Coexistence of heterogeneous coherence models for accelerators

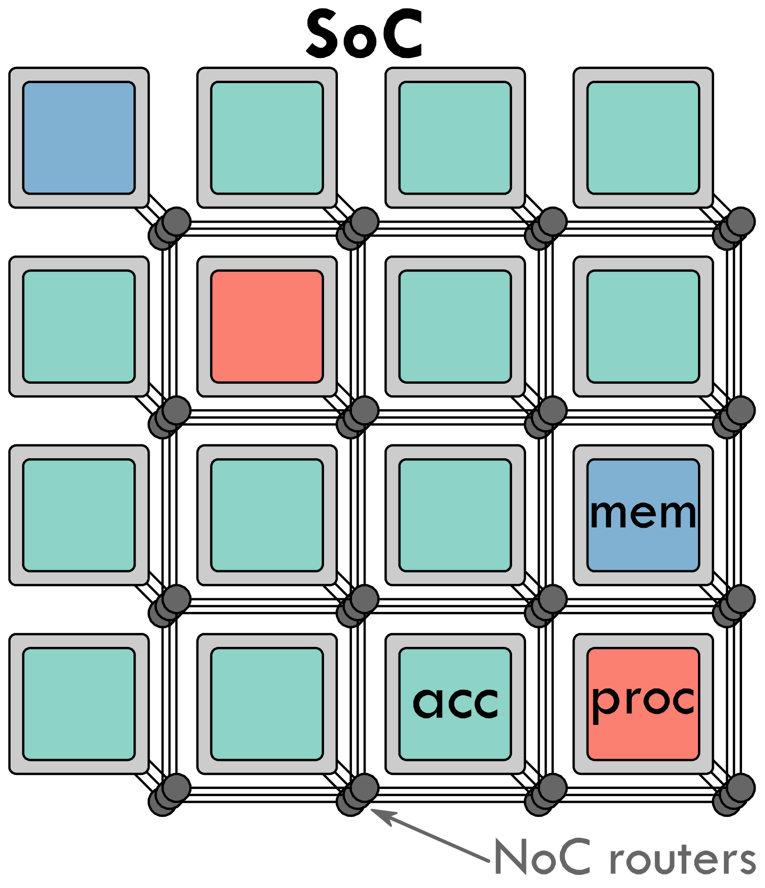
OUR SOC PLATFORM

Our design is based on an instance of **Embedded Scalable Platforms (ESP)** [L. P. Carloni, DAC '16]

- Socketed tiles
- NoC
- Easy integration and reuse of heterogeneous components

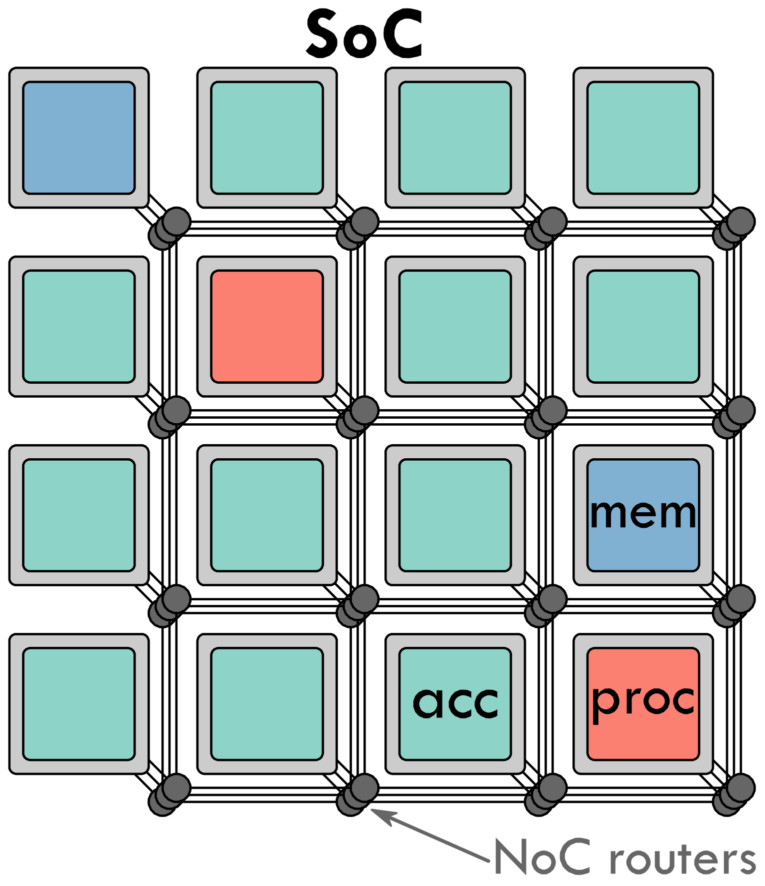
We added a cache hierarchy to ESP

- Now it can run multi-processor and multi-accelerator applications on Linux SMP



ESP: NOC

- 2D-mesh
- 1 cycle hops
- 6 physical planes to prevent deadlock and to provide sufficient bandwidth
- Point-to-point ordering required to prevent deadlock



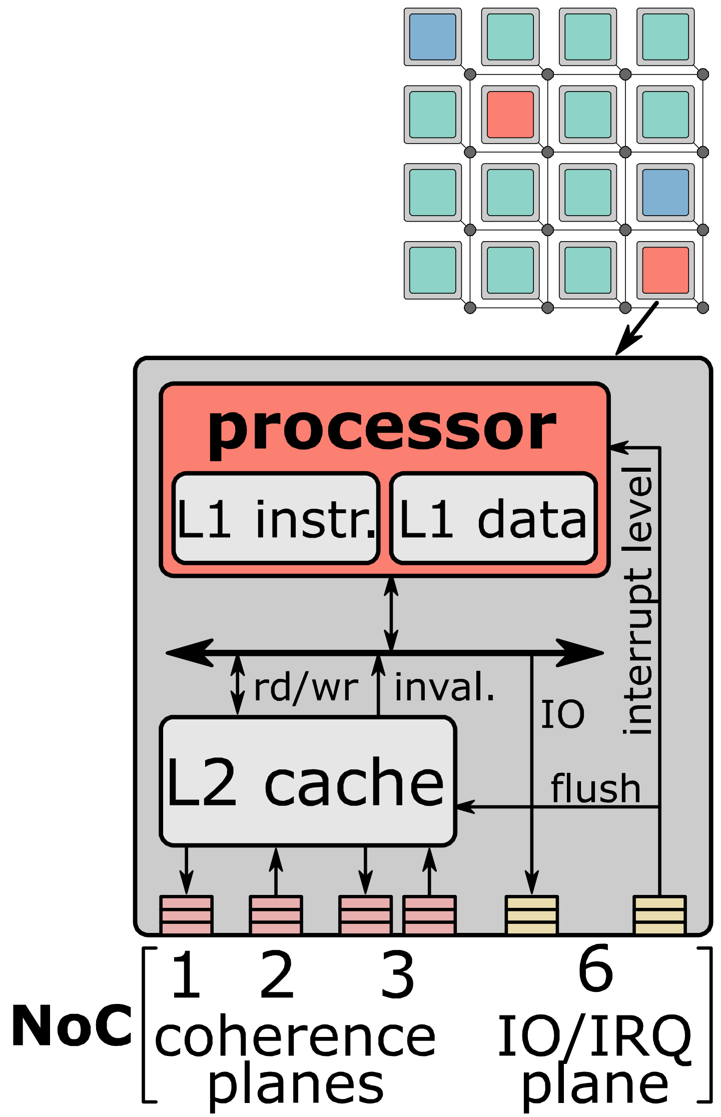
ESP: PROCESSOR TILE

Main components

- Single processor core
- L2 private cache
  - Added for this work

In this work

- Up to 2 processor tiles
- 64KB private caches
- Off-the-shelf processor with L1 write-through caches



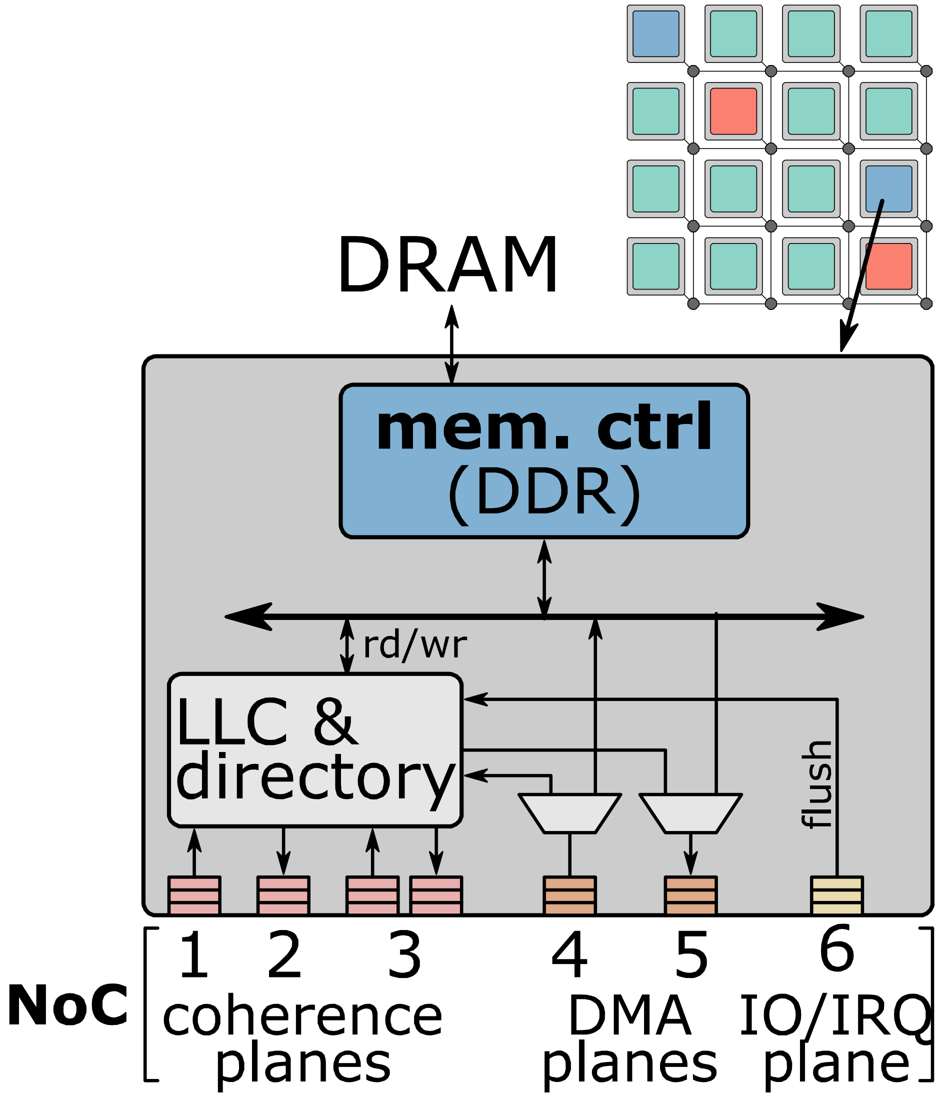
ESP: MEMORY TILE

Main components

- Memory controller
- LLC and directory
  - Added for this work
  - Can be split over multiple tiles

In this work

- Up to 2 memory tiles
- Up to 2MB aggregate LLC

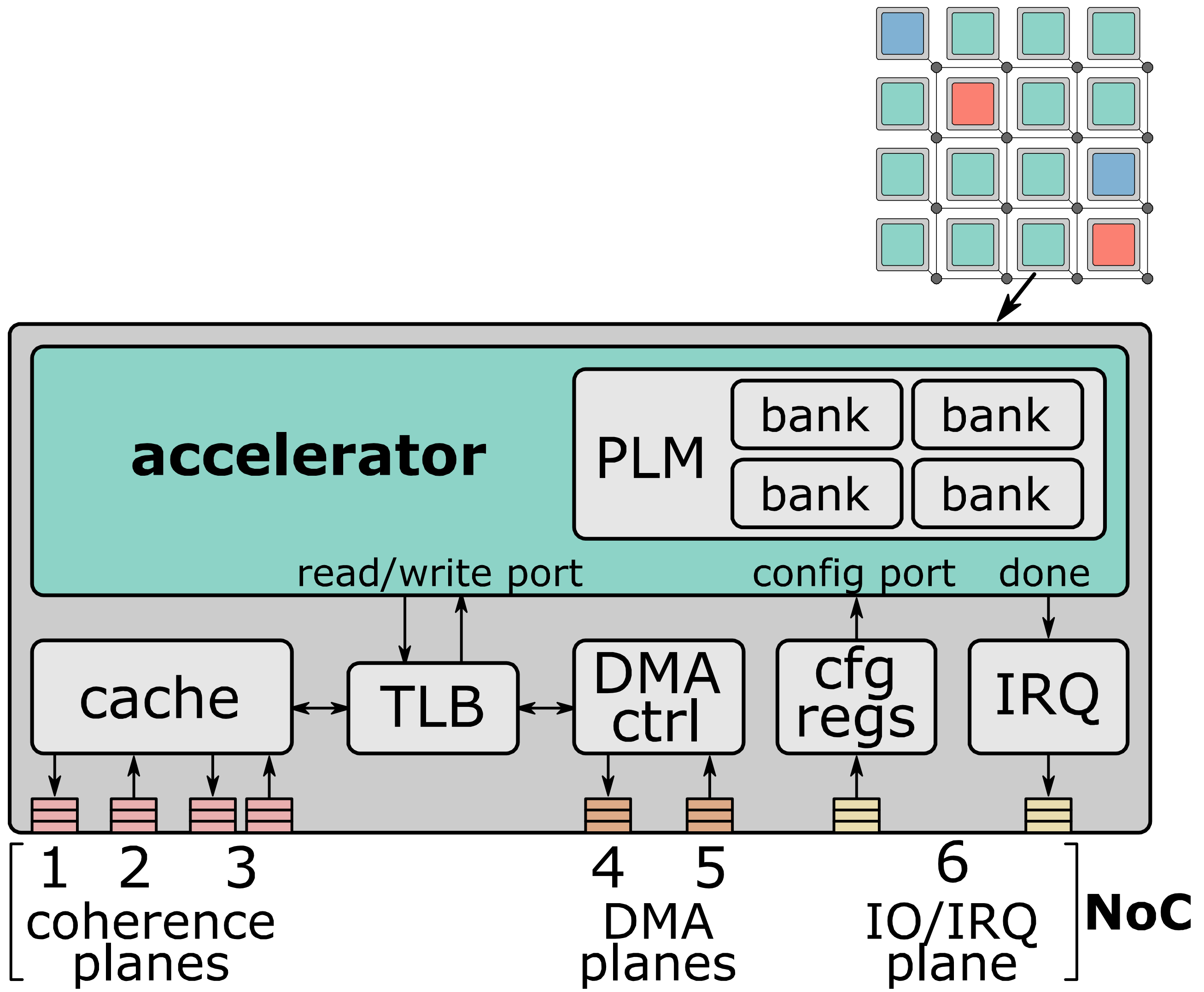


ESP: ACCELERATOR TILE

Main components

- Any accelerator complying with a simple interface
- A small TLB
- A DMA controller and/or a private cache (added for this work)

Support for **run-time selection of coherence model** through one I/O write to the configuration registers



OUR PROTOCOL

We modified a classic MESI directory-based cache-coherence protocol

- to make it work over a NoC (atomic operations)
- to support all coherence models for accelerators (recalls, flush, **LLC-coherent requests**)

Directory controller

- Write-back: add a *Valid* state and dirty bit
- Recalls
- Flush
- LLC-coherent read/write requests

Private cache controller

- L1 invalidation
- Recalls
- Flush
- Atomic operations

OUR PROTOCOL: DIRECTORY CONTROLLER EXCERPT

\ Requests State \	LLC-coherent Read	LLC-coherent Write
Invalid	Read memory Send data to requestor Go to Valid state	Read memory if misaligned Write to LLC Go to Valid state
Valid	Send data to requestor	Write to LLC
Shared	-	-
Exclusive	-	-
Modified	-	-

EXPERIMENTAL SETUP

We designed **4 custom accelerators**:

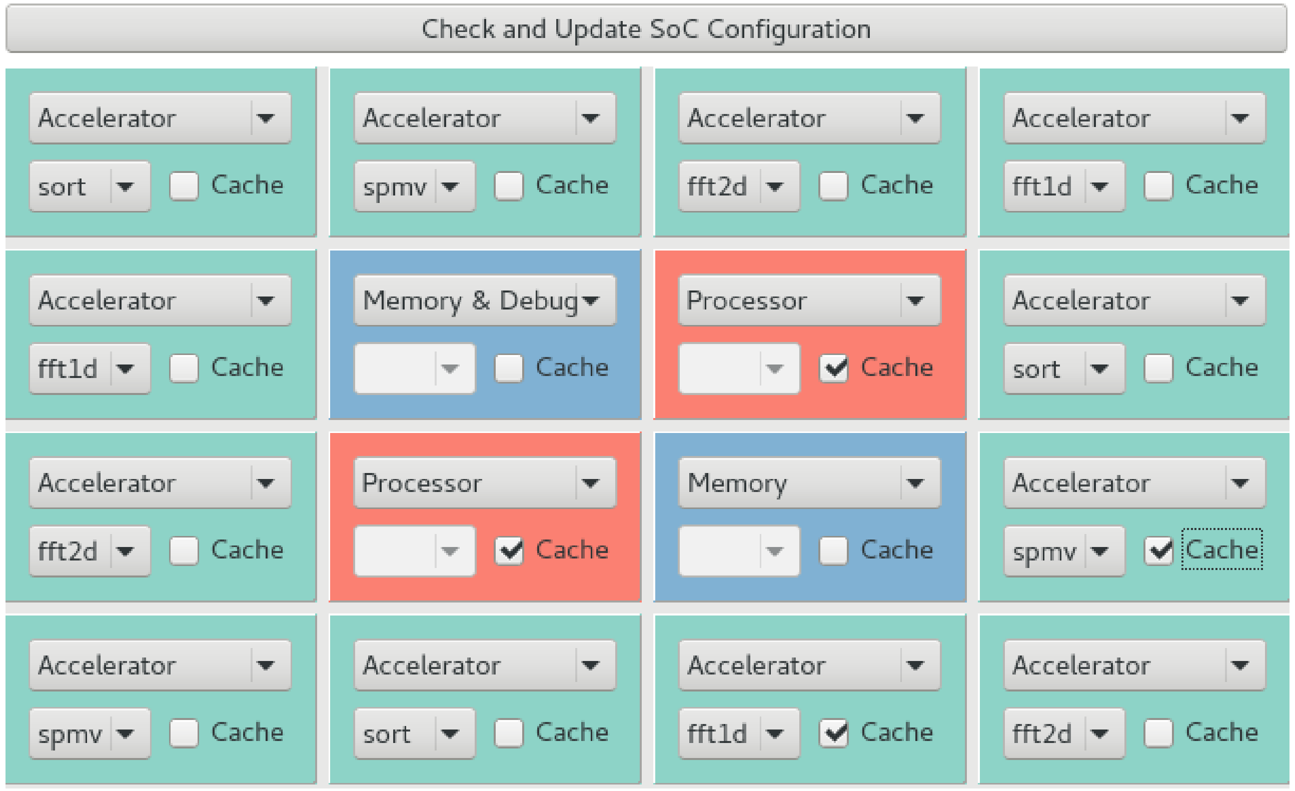
- Sort (merge and bitonic sort combined)
- Sparse Matrix-Vector Multiplication
- FFT-1D and FFT-2D

These accelerators represent a good mix of **memory access pattern** characteristics:

- Varying footprint size (32KB – 20MB)
- Streaming vs. irregular pattern
- Temporal and spatial locality

ESP's GUI:

The CAD flow from GUI to bitstream is fully automated.

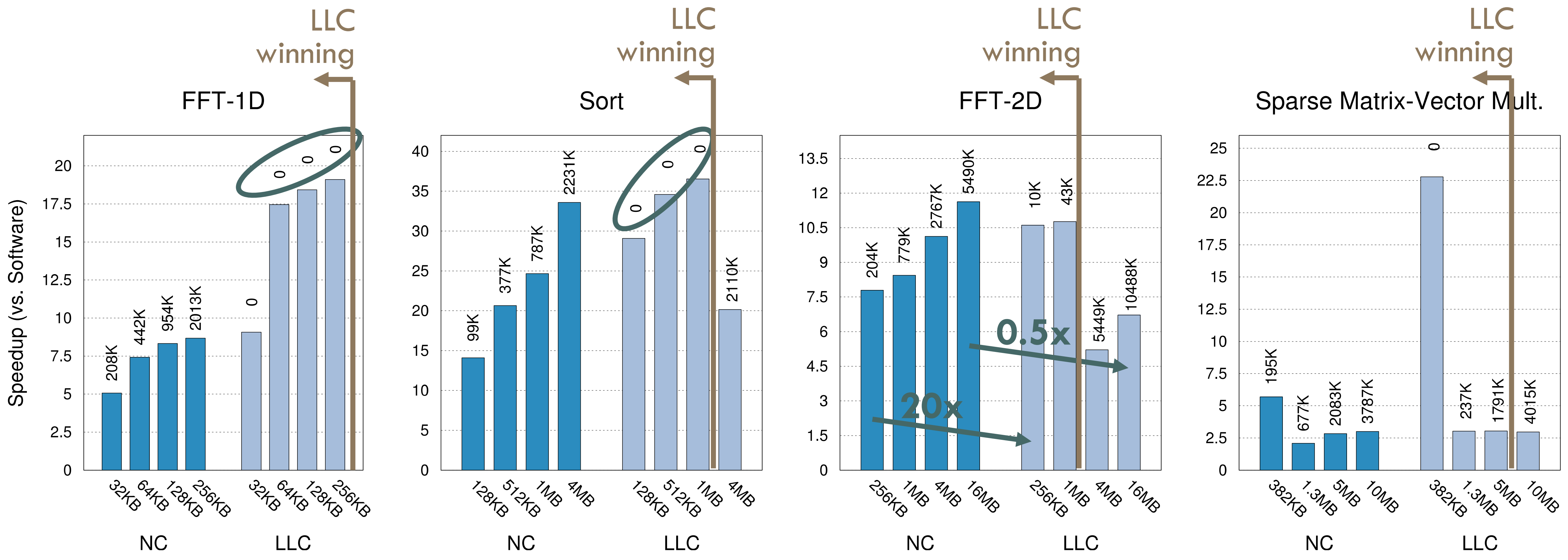


Accelerator, sort, ☐ Cache	Accelerator, spmv, ☐ Cache	Accelerator, fft2d, ☐ Cache	Accelerator, fft1d, ☐ Cache
Accelerator, fft1d, ☐ Cache	Memory & Debug, ☐ Cache	Processor, ✓ Cache	Accelerator, sort, ☐ Cache
Accelerator, fft2d, ☐ Cache	Processor, ✓ Cache	Memory, ☐ Cache	Accelerator, spmv, ✓ Cache
Accelerator, spmv, ☐ Cache	Accelerator, sort, ☐ Cache	Accelerator, fft1d, ✓ Cache	Accelerator, fft2d, ☐ Cache

We deployed our SoC on FPGA and we executed applications on Linux SMP.

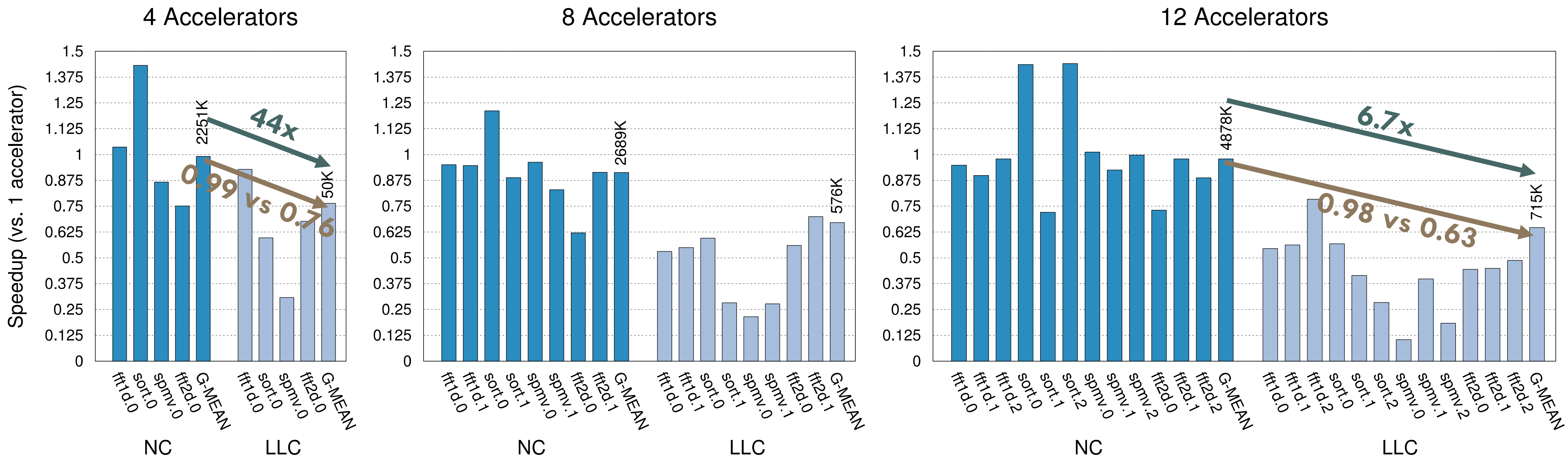
RESULTS: SINGLE ACCELERATOR

NC = non-coherent
LLC = LLC-coherent
Speedup ▬
DRAM accesses ▬



RESULTS: MULTIPLE ACCELERATORS

NC = non-coherent
LLC = LLC-coherent
Speedup ▬
DRAM accesses ▬



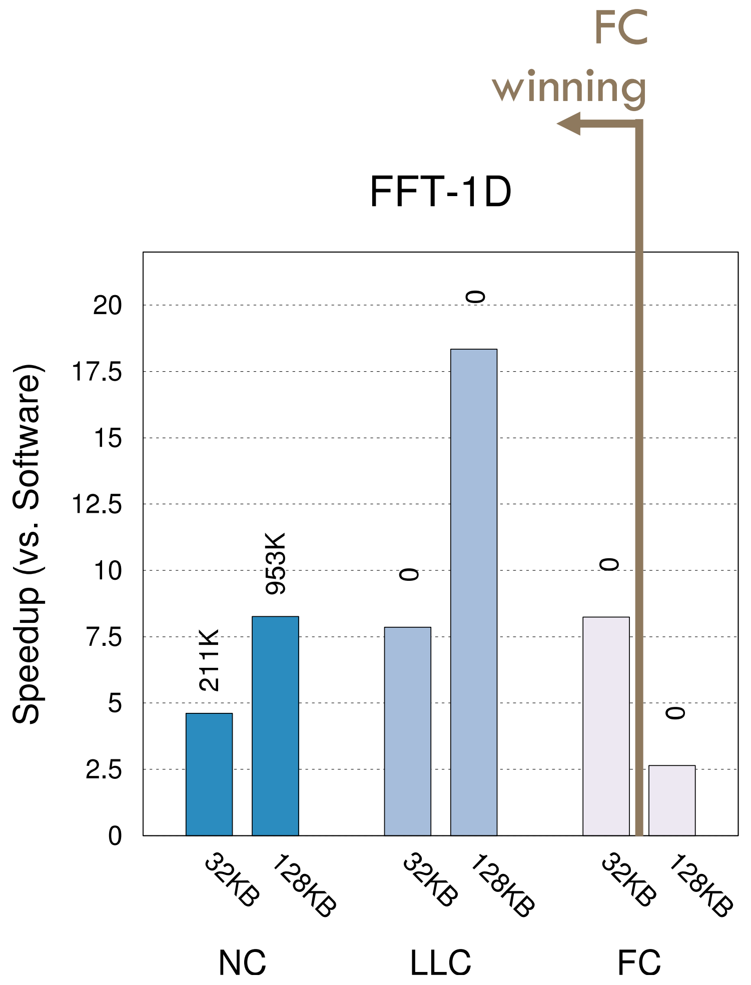
Dataset size: 256KB to 512KB

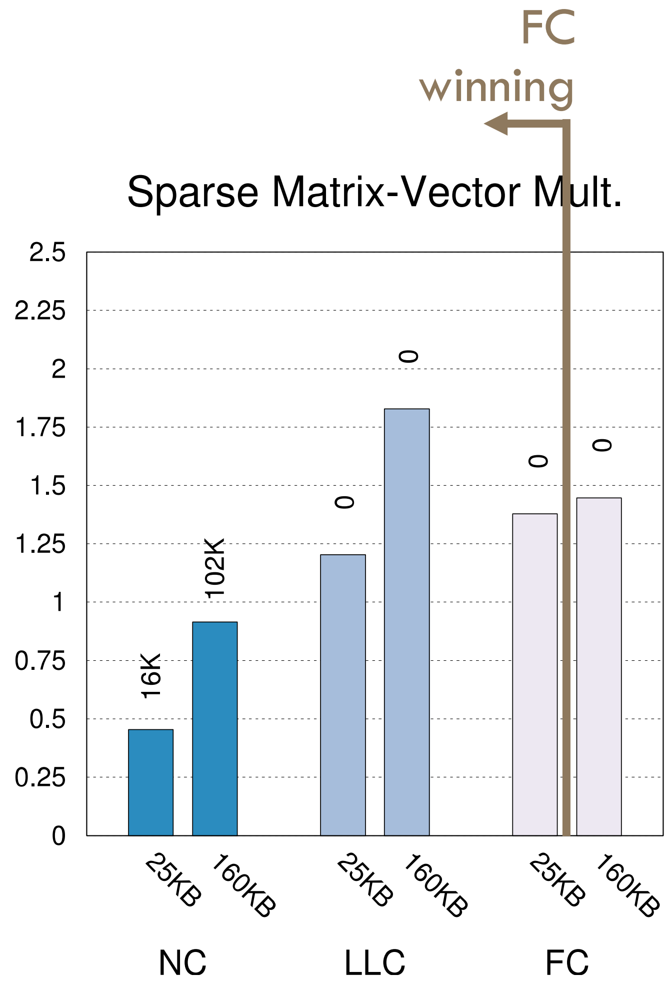
RESULTS: FULLY-COHERENT ACCELERATORS

The fully-coherent model can win for workloads whose data structures fit the accelerator's private cache.

No flush needed.

NC = non-coherent
LLC = LLC-coherent
FC = fully-coherent
Speedup ▬

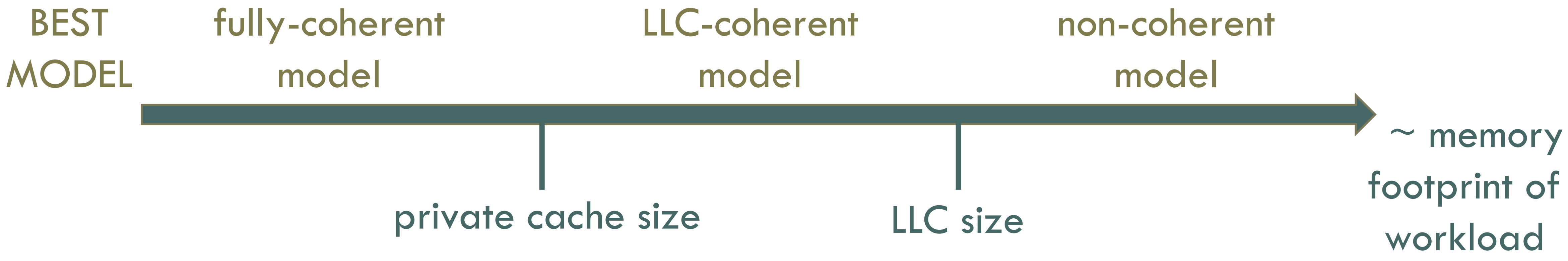




RESULTS: SUMMARY

- The best coherence model varies with the accelerator **workload size** and with the **number of active accelerators** in the system.
- LLC-coherent and fully-coherent models can significantly **reduce accesses to DRAM.**

RULE OF THUMB



CONCLUSIONS

- There is **no absolute winner** among the coherence models.
  - Workload size, caches size and number of active accelerators influences the best choice → Hence, the best choice can vary at runtime.
- We proposed a **cache-coherence protocol** that supports all three coherence models in a NoC-based SoC:
  - Fully-coherent, LLC-coherent, non-coherent.
- We designed a NoC-based SoC architecture enabling
  - **Coexistence** of heterogeneous coherence models operating simultaneously.
  - **Run-time selection** of the coherence model for each accelerator.

THANK YOU!

Any question?

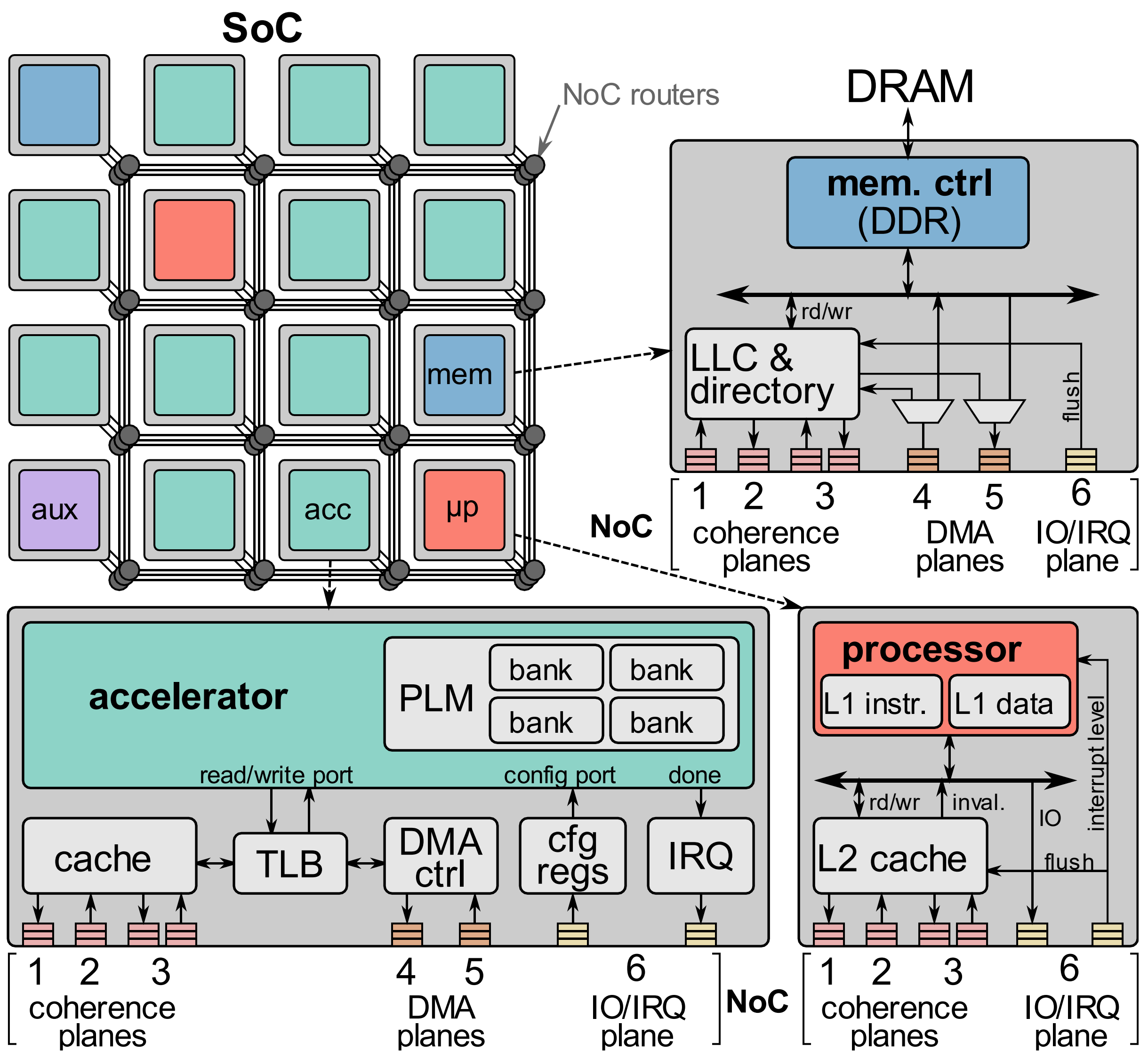
Davide Giri
Paolo Mantovani
Luca P. Carloni

NOC-BASED SUPPORT OF HETEROGENEOUS CACHE-COHERENCE MODELS FOR ACCELERATORS

BACKUP

ESP: PROGRAMMABILITY

- The accelerator driver is invoked by an application to offload a task.
- Accelerator tiles handle virtual memory without interrupting the processor cores
- We use locks to enforce race free execution of the accelerators. Additionally:
  - During the execution of **non-coherent** accelerators, we ensure that there exists only a single copy of the data.
  - For **LLC-coherent** accelerators data can be present both in DRAM and in the LLC.
- The flush phase becomes a negligible overhead for large accelerator workloads

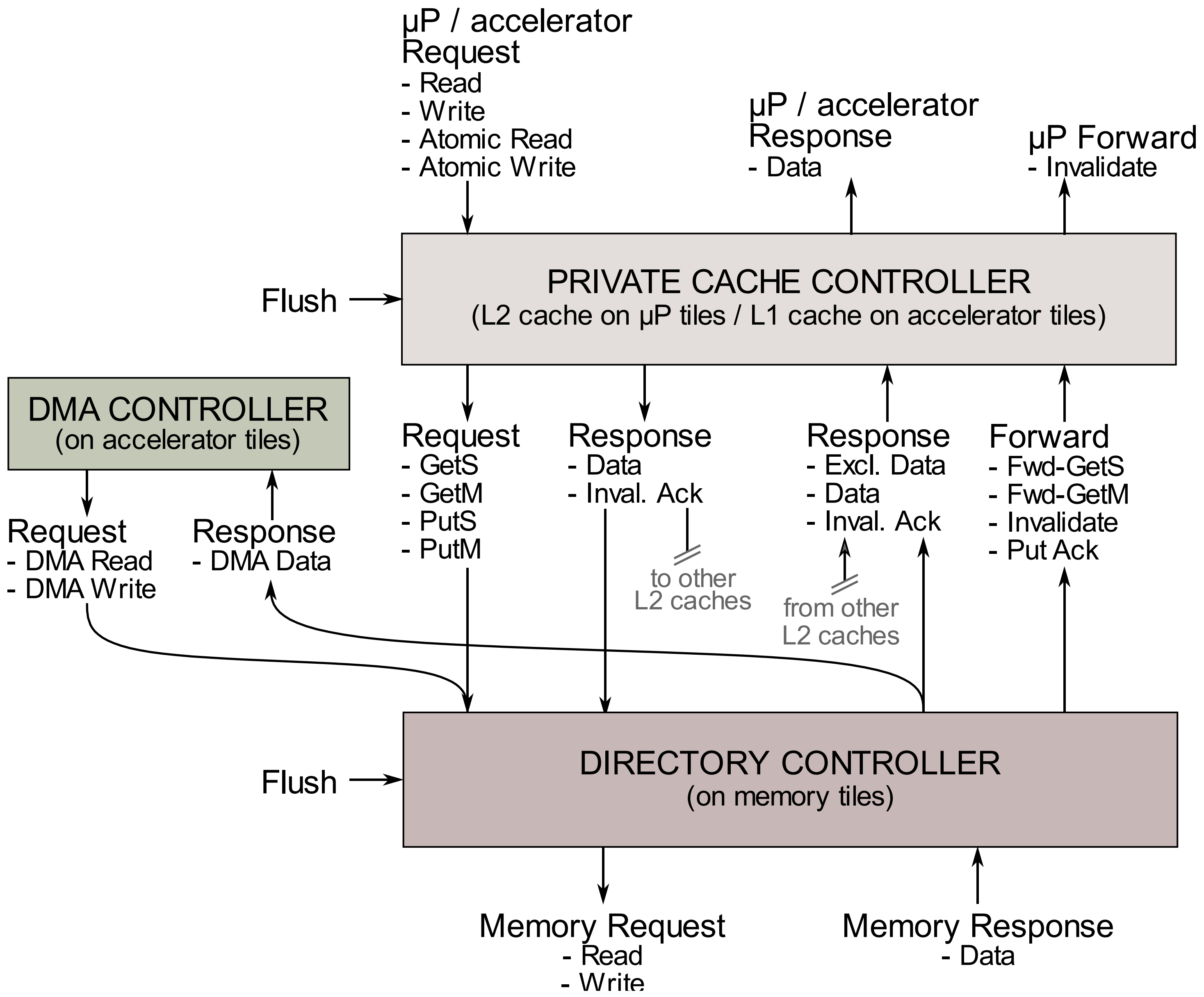


ESP: CACHES

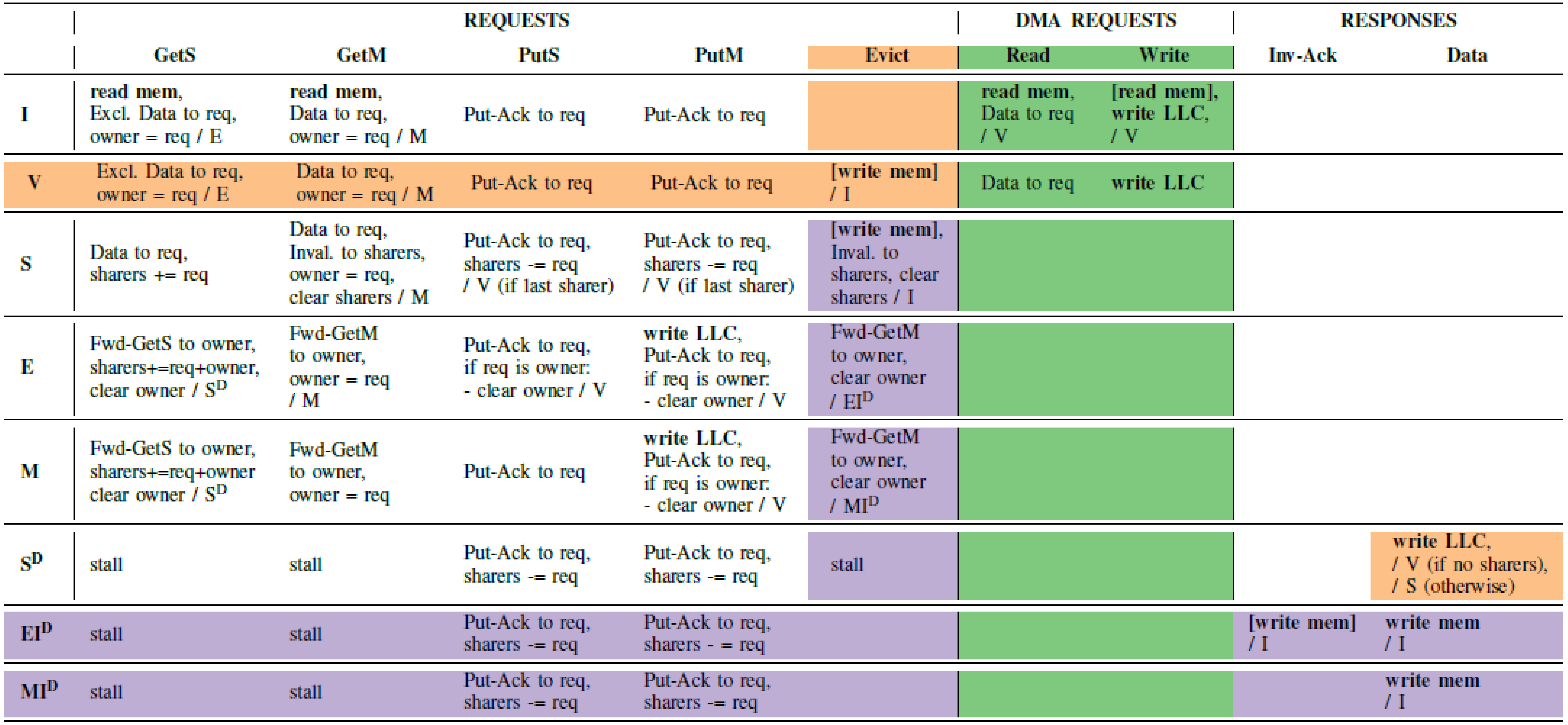
- Designed in SystemC and implemented through HLS.
- Configurable sets, ways and the number of sharers and owners.
- The device driver can select which caches to flush.

For this work:

- LLC: 2 MB
- Private caches: 64KB



OUR PROTOCOL: DIRECTORY CONTROLLER EXCERPT



	REQUESTS					DMA REQUESTS		RESPONSES	
	GetS	GetM	PutS	PutM	Evict	Read	Write	Inv-Ack	Data
I	**read mem,** Excl. Data to req, owner = req / E	**read mem,** Data to req, owner = req / M	Put-Ack to req	Put-Ack to req		**read mem,** Data to req / V	**[read mem],** **write LLC,** / V		
V	Excl. Data to req, owner = req / E	Data to req, owner = req / M	Put-Ack to req	Put-Ack to req	**[write mem]** / I	Data to req	**write LLC**		
S	Data to req, sharers += req	Data to req, Inval. to sharers, owner = req, clear sharers / M	Put-Ack to req, sharers -= req / V (if last sharer)	Put-Ack to req, sharers -= req / V (if last sharer)	**[write mem],** Inval. to sharers, clear sharers / I				
E	Fwd-GetS to owner, sharers+=req+owner, clear owner / S^D	Fwd-GetM to owner, owner = req / M	Put-Ack to req, if req is owner: - clear owner / V	**write LLC,** Put-Ack to req, if req is owner: - clear owner / V	Fwd-GetM to owner, clear owner / EI^D				
M	Fwd-GetS to owner, sharers+=req+owner clear owner / S^D	Fwd-GetM to owner, owner = req	Put-Ack to req	**write LLC,** Put-Ack to req, if req is owner: - clear owner / V	Fwd-GetM to owner, clear owner / MI^D				
S^D	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers -= req	stall				**write LLC,** / V (if no sharers), / S (otherwise)
EI^D	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers - = req				**[write mem]** / I	**write mem** / I
MI^D	stall	stall	Put-Ack to req, sharers -= req	Put-Ack to req, sharers -= req					**write mem** / I