Runtime Reconfigurable Memory Hierarchy in Embedded Scalable Platforms

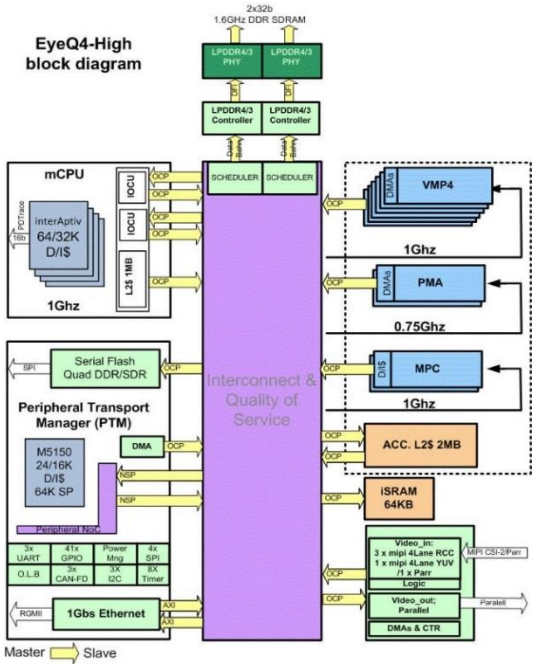
Davide Giri, Paolo Mantovani, **Luca Carloni**

Department of Computer Science

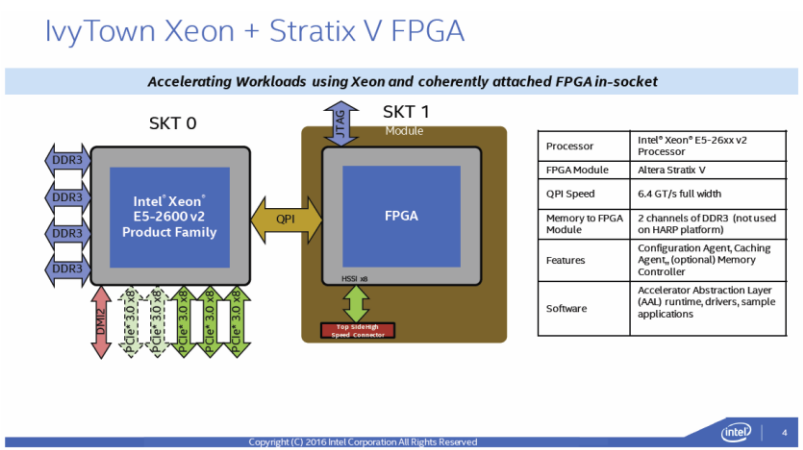
Columbia University in the City of New York

24th Asia and South Pacific Design Automation Conference (ASPDAC), January 21-24, 2019

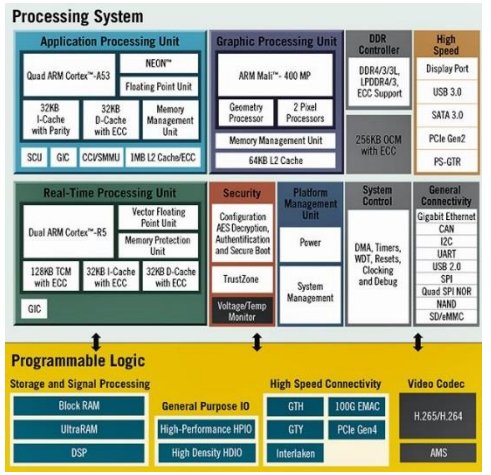
Heterogeneous Architectures Are Emerging Everywhere



[Source: www.mobileye.com/]

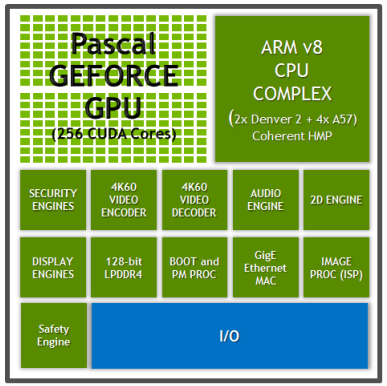


[Source: "Xeon+FPGA Tutorial @ ISCA'16"]

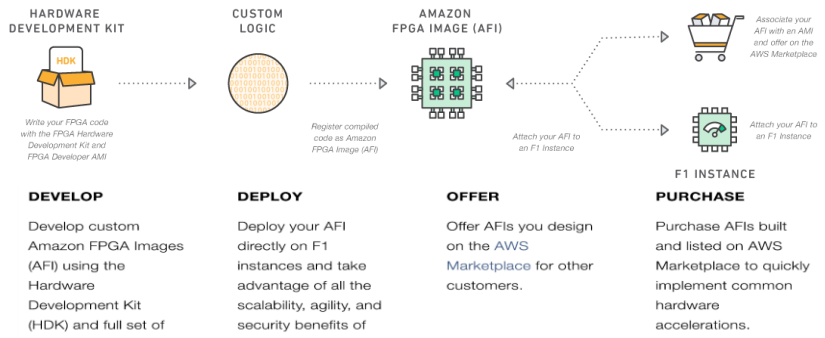


[Source: www.xilinx.com/]

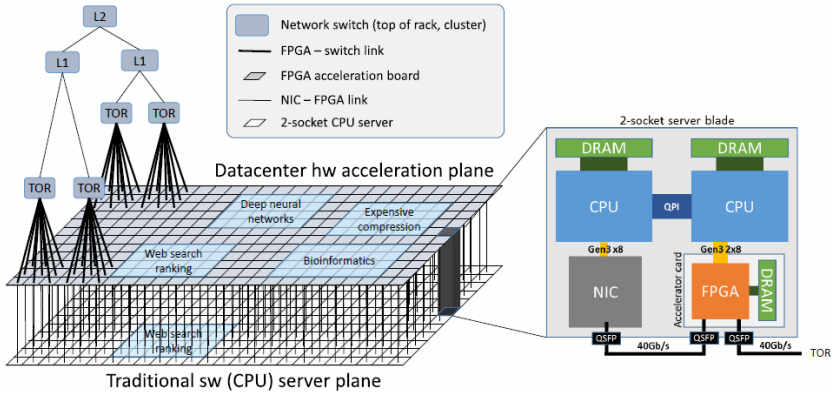
[Source: https://cloudplatform.googleblog.com/]



[Source: https://blogs.nvidia.com/]



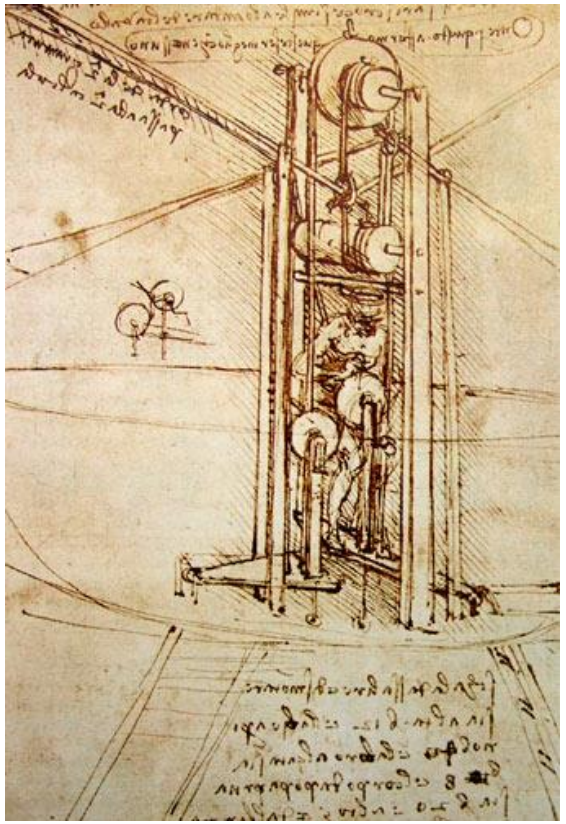
[Source: https://aws.amazon.com/ec2/instance-types/f1/]



[Source: www.microsoft.com/]

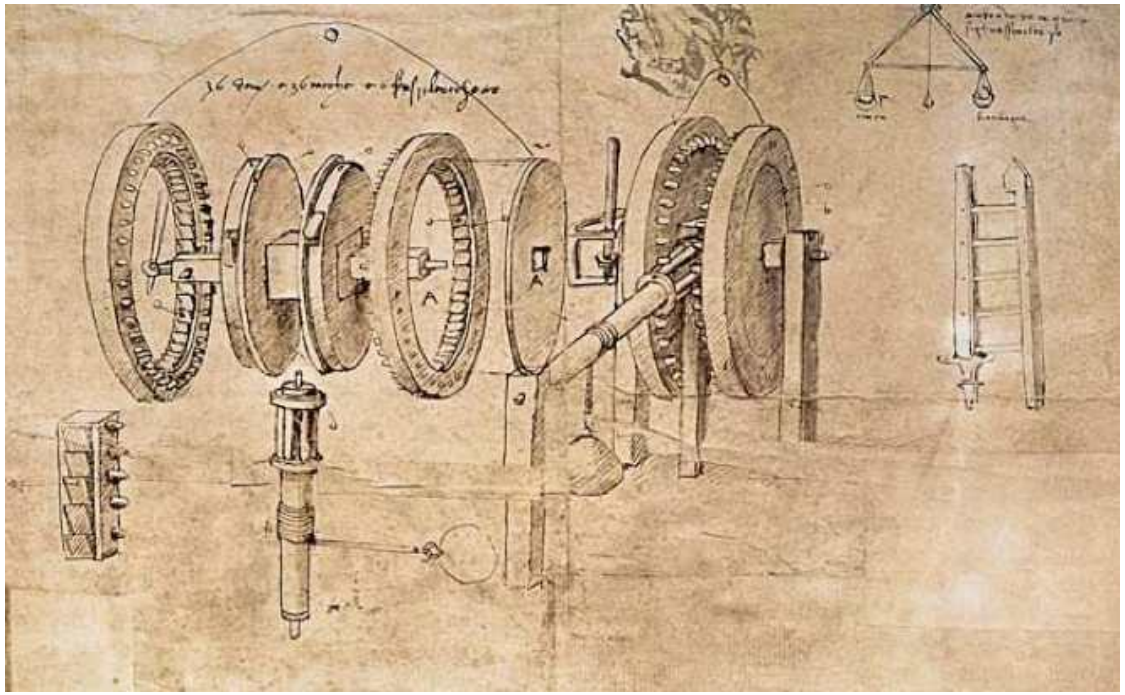
A (Perhaps Easy?) Prediction: No Single Architecture Will Emerge as the Sole Winner

- **The migration from homogeneous multi-core architectures to heterogeneous System-on-Chip architectures will accelerate, across almost all computing domains**
  - from IoT devices, embedded systems and mobile devices to data centers and supercomputers specialization will be the key to realize competitive systems
- **A heterogeneous SoC will combine an increasingly diverse set of components**
  - different CPUs, GPUs, hardware accelerators, memory hierarchies, I/O peripherals, sensors, reconfigurable engines, analog blocks…
- **The set of heterogeneous SoCs in production in any given year will be itself heterogeneous!**
  - no single SoC architecture will dominate all the markets



Where the Key Challenges in SoC Design Are…

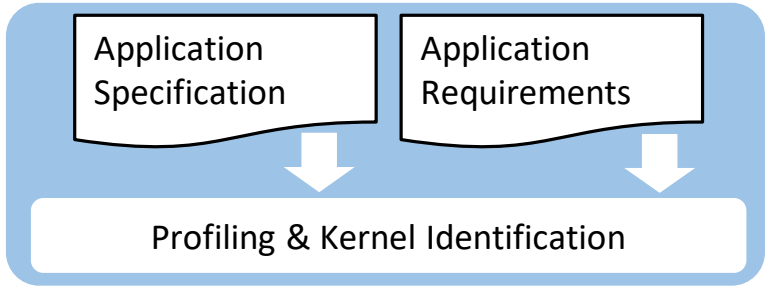
- **The biggest challenges are (and will increasingly be) found in the complexity of system integration**
  - How to design, program and validate scalable systems that combine a very large number of heterogeneous components to provide a solution that is specialized for a target class of applications?

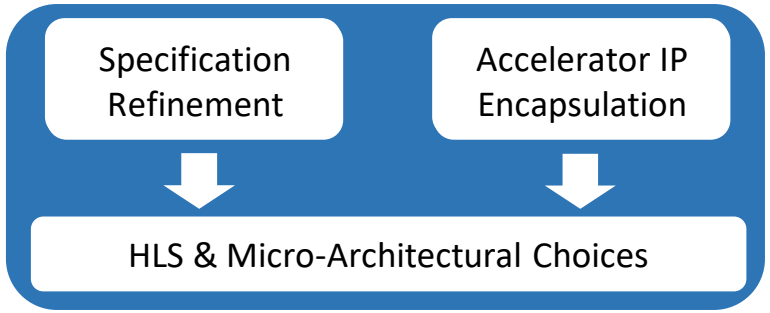


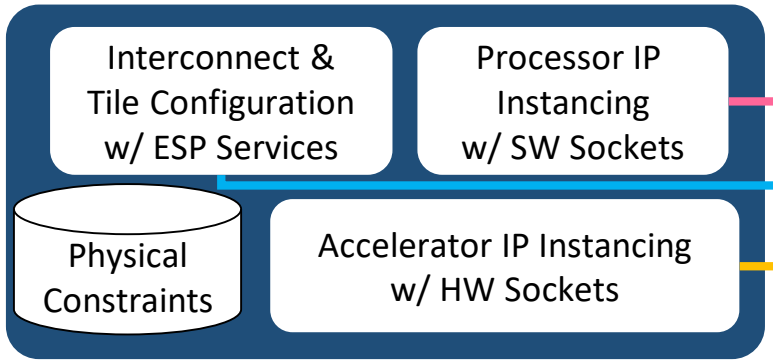
- **How to handle this complexity?**
  - raise the level of abstraction to System-Level Design
  - adopt compositional design methods with the Protocol & Shell Paradigm
  - promote Design Reuse

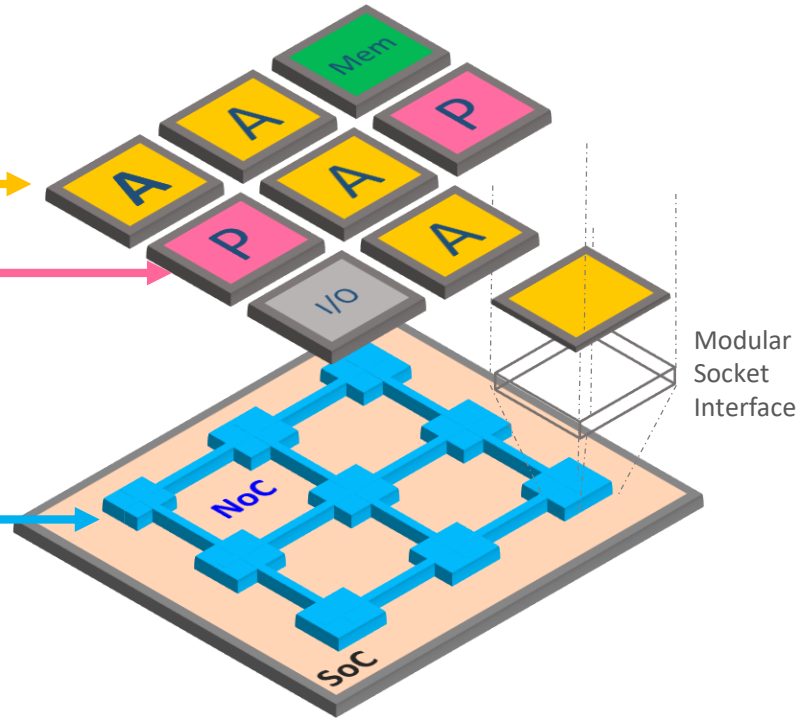
Embedded Scalable Platforms (ESP)

- The flexible architecture simplifies the integration of heterogeneous components by
  - balancing regularity and specialization
  - relying on the Protocol & Shell paradigm and scalable communication infrastructure
- The system-level design methodology promotes HW/SW co-design and is supported by
  - a mix of commercial and in-house CAD tools
  - a growing library of reusable IP blocks



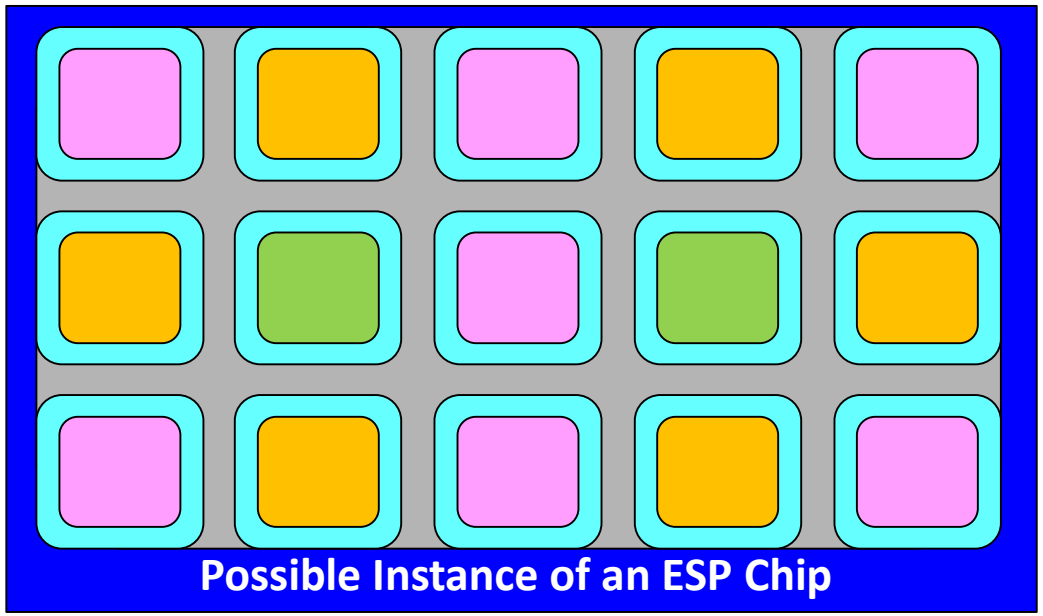






[L. P. Carloni, *The Case for Embedded Scalable Platforms*, DAC 2016]

The ESP Scalable Architecture Template



- Processor Tiles
  - each hosting at least one configurable processor core capable of running an OS
- Accelerator Tiles
  - synthesized from high-level specs
- Other Tiles
  - memory interfaces, I/O, etc.
- Network-on-Chip (NoC)
  - playing key roles at both design and run time

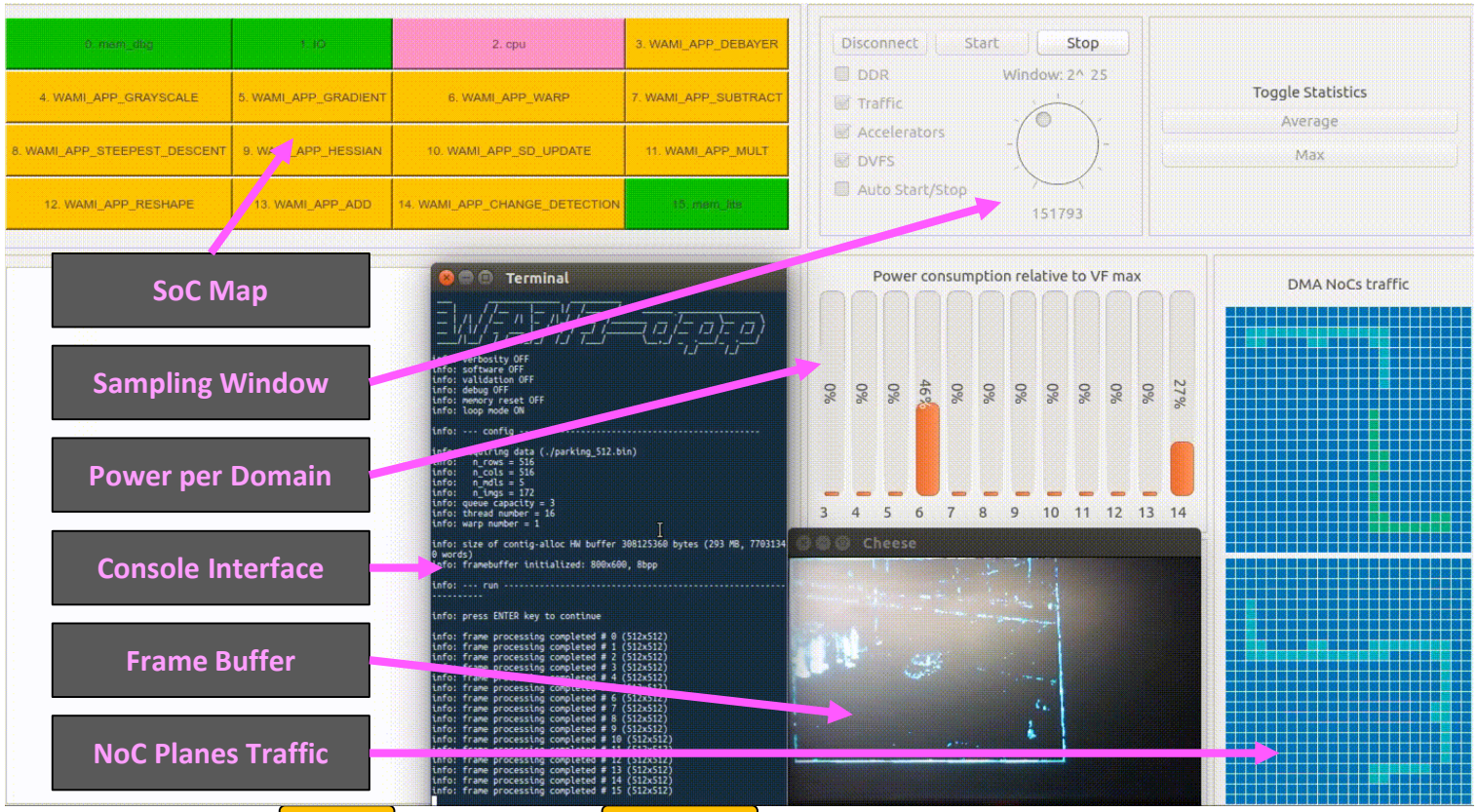
Template Properties

- Regularity
  - tile-based design
  - pre-designed on-chip infrastructure for communication and resource management
- Flexibility
  - each ESP design is the result of a configurable mix of programmable tiles and accelerator tiles
- Specialization
  - with automatic high-level synthesis of accelerators for key computational kernels

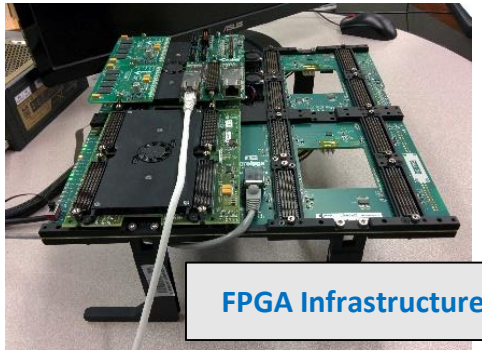
Our System-Level Design Approach: Key Ingredients

- **Develop Platforms, not just Architectures**
  - A platform combines an architecture and a companion design methodology
- **Raise the level of abstraction**
  - Move from RTL Design to System-Level Design
  - Move from ISA simulators to Virtual Platforms
  - Move from Verilog/VHDL to SystemC, also an IEEE standard
  - Move from Logic Synthesis to High-Level Synthesis (both commercial and in-house tools), which is the key to enabling rich design-space exploration
- **Adopt compositional design methods**
  - Rely on customizable libraries of HW/SW interfaces to simplify the integration of heterogeneous components
- **Use formal metrics for design reuse**
  - Synthesize Pareto frontiers of optimal implementations from high-level specs
- **Build real prototypes (both chips and FPGA-based full-system designs)**
  - Prototypes drive research in systems, architectures, software and CAD tools

Example of an ESP Based-Design: FPGA Prototype to Accelerate Wide-Area Motion Imagery



Motion Detection from WAMI-Application



FPGA Infrastructure

- **Design:** Complete design of WAMI-App running on an FPGA implementation of an ESP architecture
  - featuring 1 embedded processor, 12 accelerators, 1 five-plane NoC, and 2 DRAM controllers
  - SW application running on top of Linux while leveraging multi-threading library to program the accelerators and control their concurrent, pipelined execution
  - Five-plane, 2D-mesh NoC efficiently supports multiple independent frequency domains and a variety of platform services

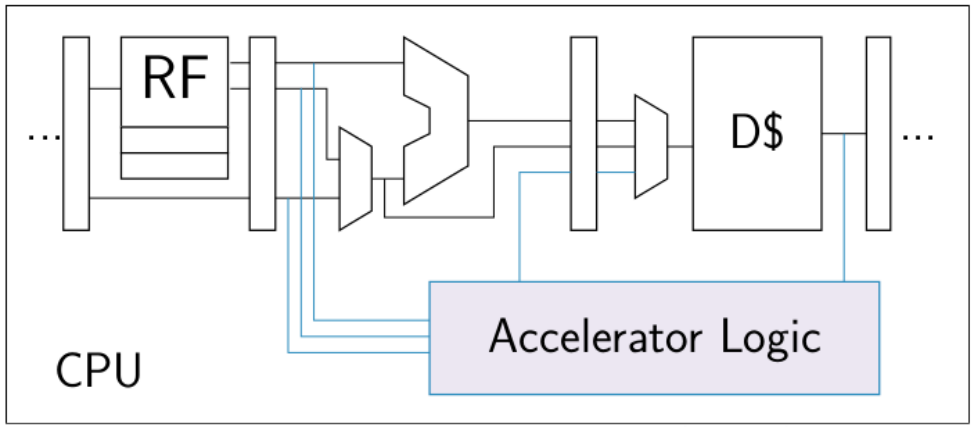
[P. Mantovani , L. P. Carloni et al., *An FPGA-Based Infrastructure for Fine-Grained DVFS Analysis in High-Performance Embedded Systems*, DAC 2016]

How to Couple Accelerators, Processors and Memories?

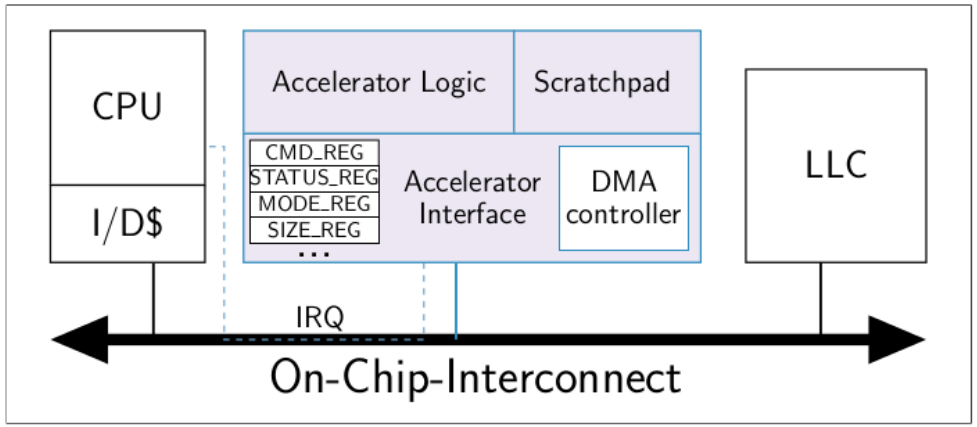
- There are two main models of coupling accelerators with processors, memories
  - Tightly-Coupled Accelerators
    - designed with the processor core
    - located within the processor core
    - execute fine-grain tasks on small datasets
    - typically accessed via specialized instructions
  - Loosely-Coupled Accelerators
    - designed independently from the processor core
    - located outside the processor core
    - execute coarse-grain tasks on large datasets
    - typically accessed via device drivers

[E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, An Analysis of Accelerator Coupling in Heterogeneous Architectures, DAC'15]

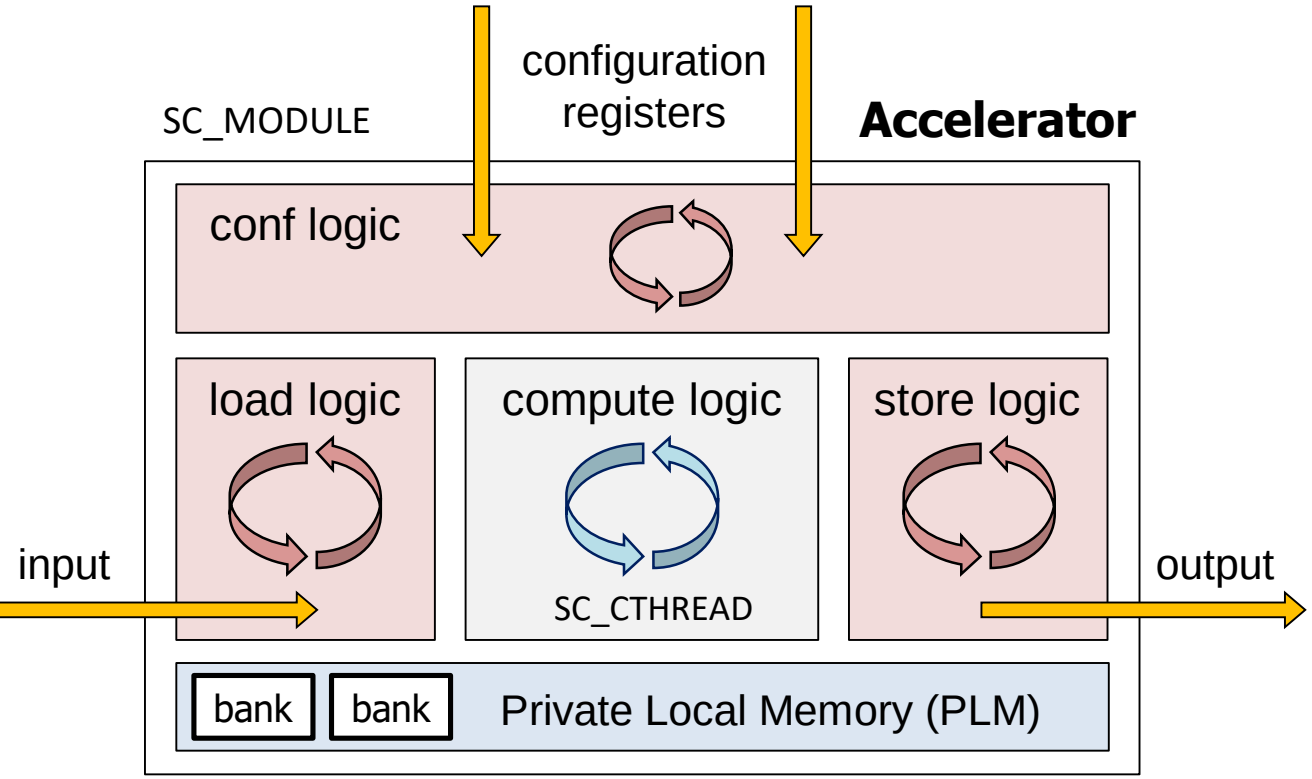
Tightly-Coupled Accelerators (TCA)



Loosely-Coupled Accelerators (LCA)



Modeling Loosely-Coupled Accelerators

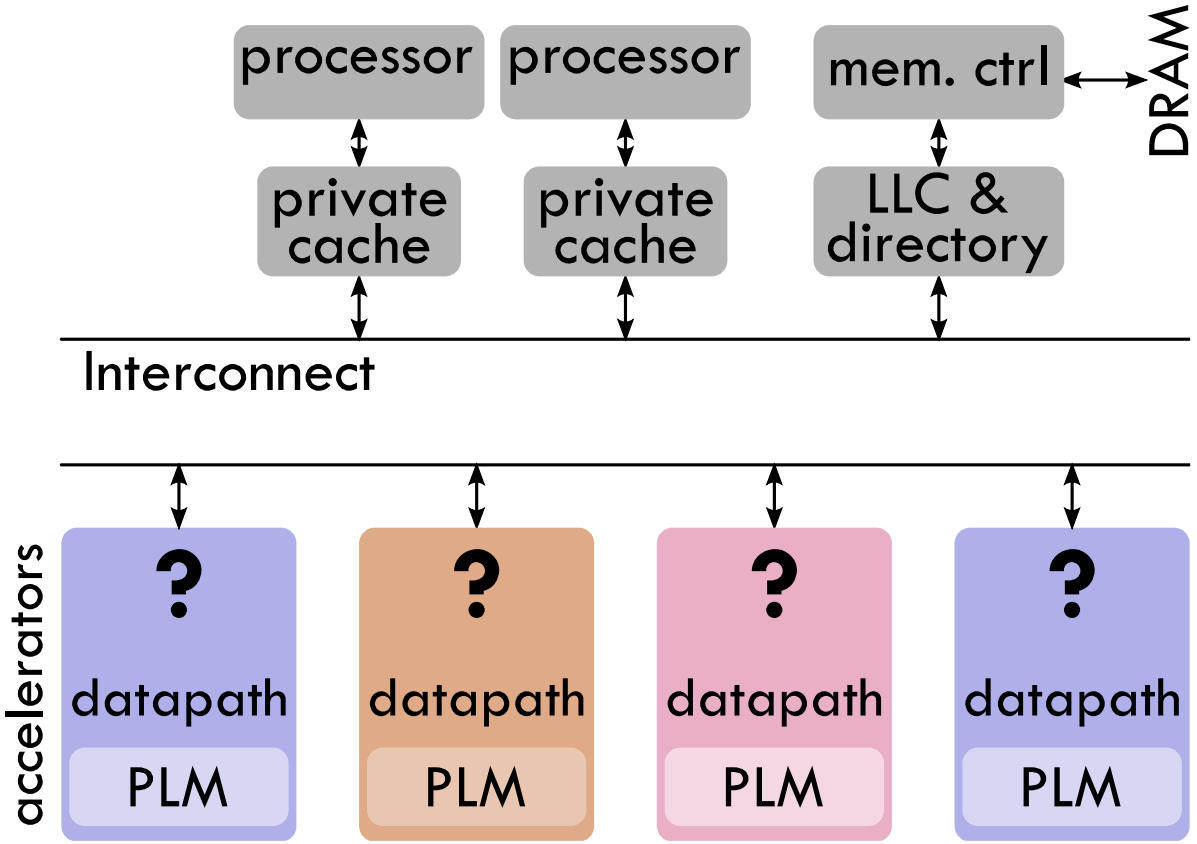


- **The behavior of loosely-coupled accelerators has 4 main phases**
  - configuration, input, compute, output
- **I/O phases transfer chunks of data from DRAM to the PLM**
  - these transfers are specified with TLM primitives, implemented with DMA mechanisms

- **The accelerator model enables the definition of a configurable interface that simplifies the integration of the accelerator within any ESP instance**
  - by decoupling the design of any accelerators from the design of the rest of the SoC

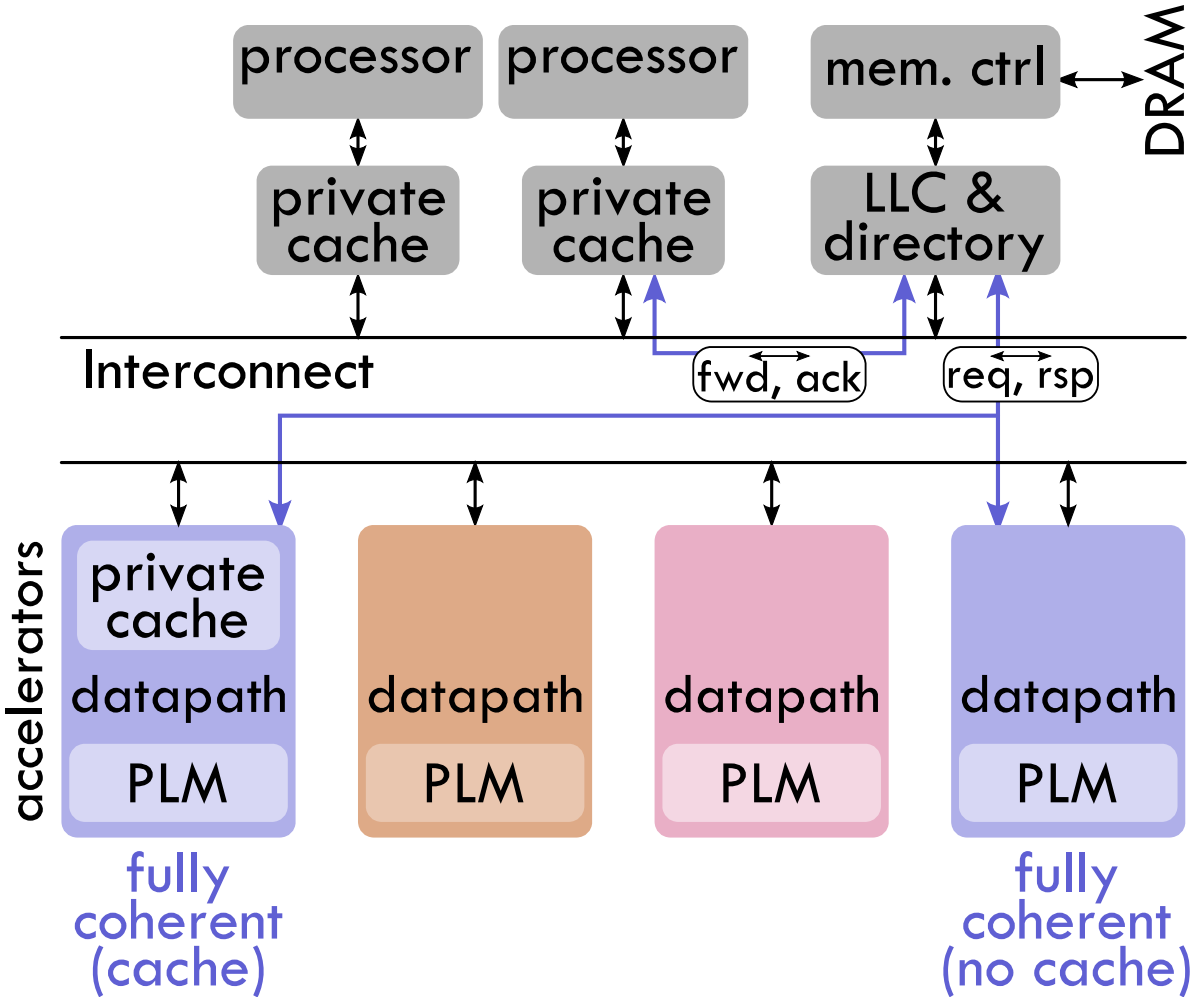
Loosely-Coupled Accelerators

- Major speedups and energy savings:
  - highly parallel and customized datapath
  - aggressively banked, multi-ported, *private local memory (PLM)*
- What should the cache coherence model for accelerators be?
  - 3 main models in literature
    [D. Giri et al., IEEE Micro '18]



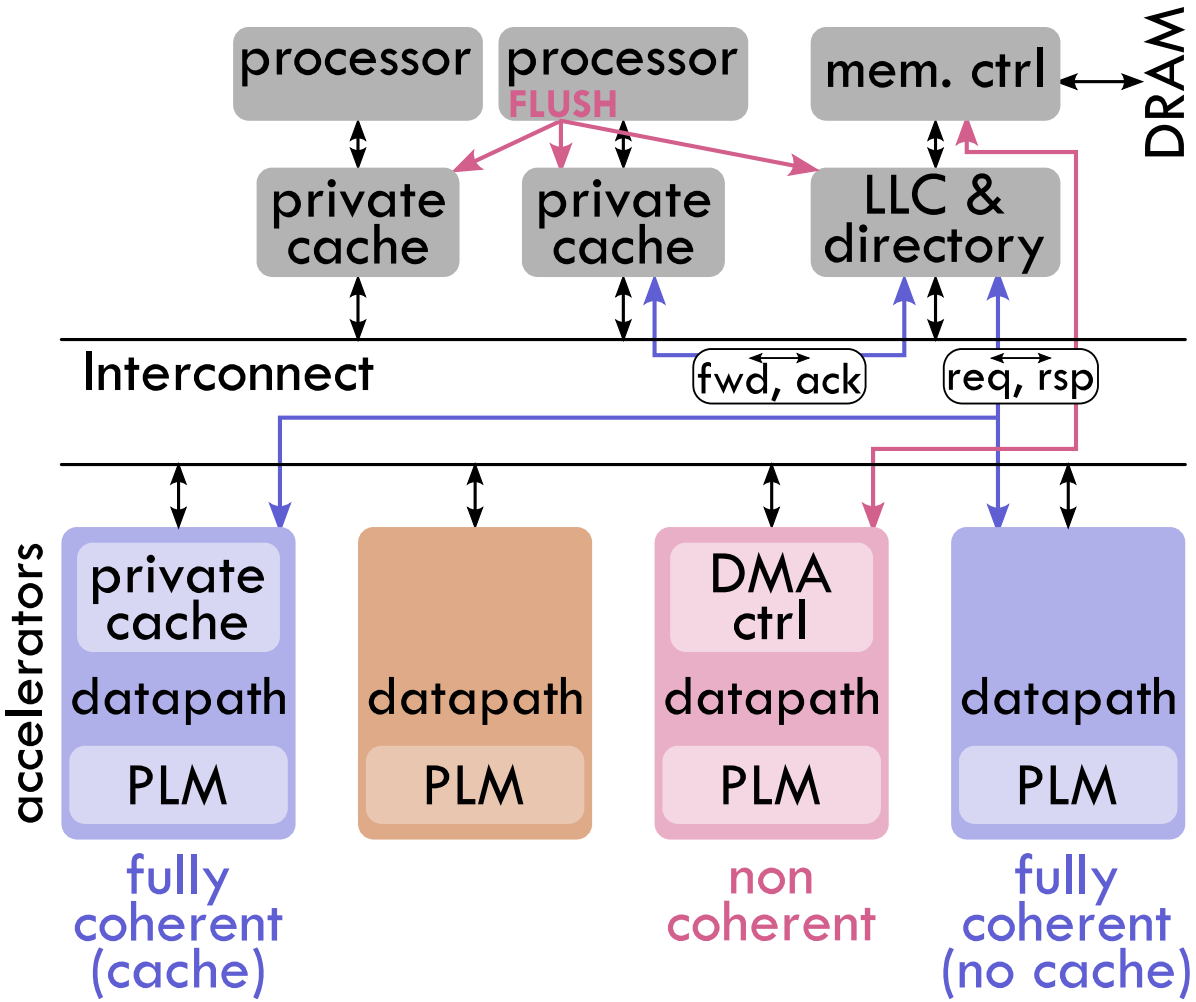
Accelerator Models: Fully Coherent

- Coherent with the 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
    - v IBM CAPI, *[Y. Shao et al., MICRO '16], [M. J. Lyons et al., TACO '12]*
    - x ARM ACE-lite



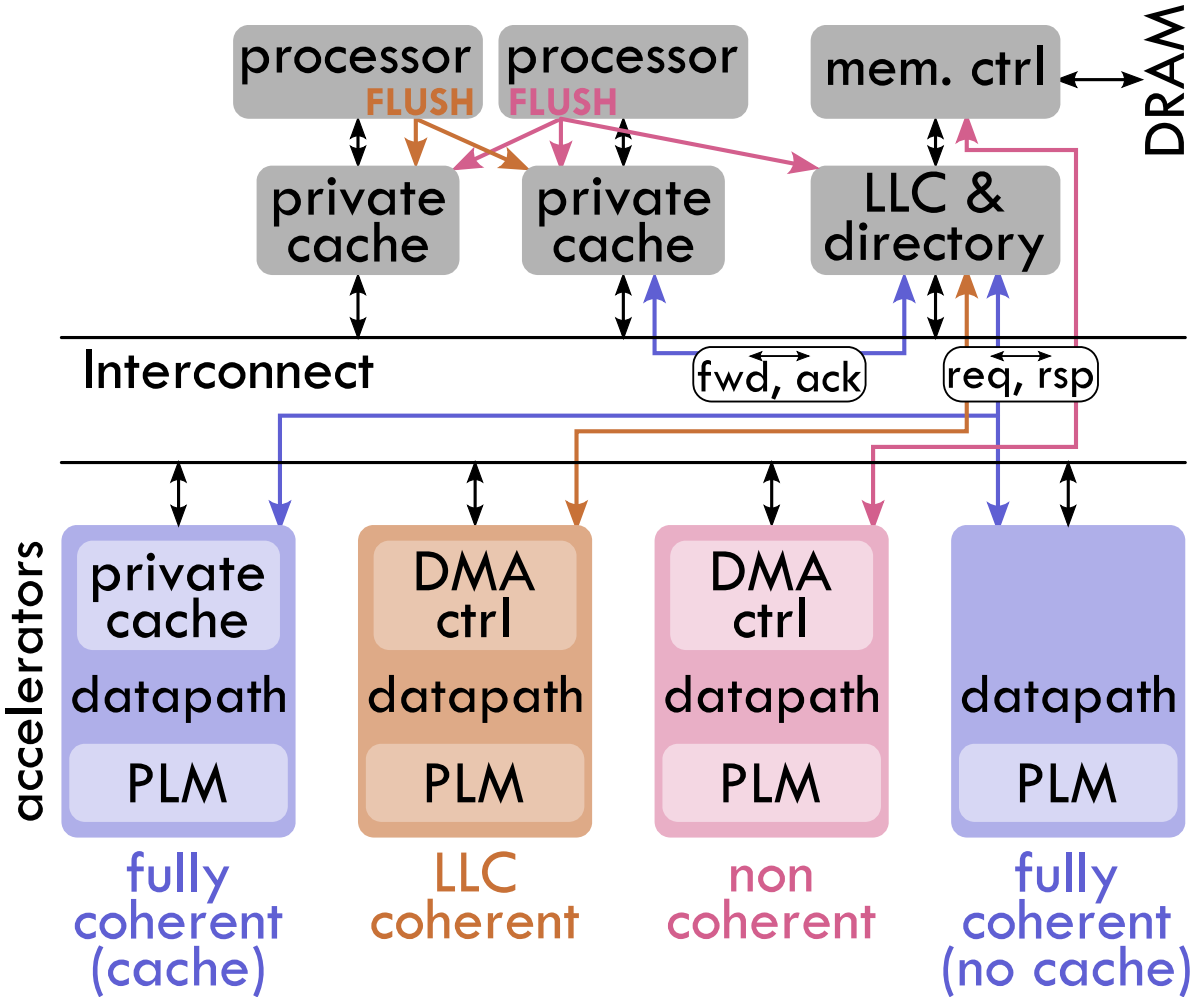
Accelerator Models: Non Coherent

- Not coherent with cache hierarchy
  - caches are by-passed while talking with DRAM
- Programming requirements
  - race-free accelerator execution
  - flush all caches prior to accelerator execution
- Implementation variants
  - generally NoC-based & DMA-based
    - *[Y. Chen et al., ICCD '13]*
      [E. Cota et al., DAC '15]
      [Y. Shao et al., MICRO '16]



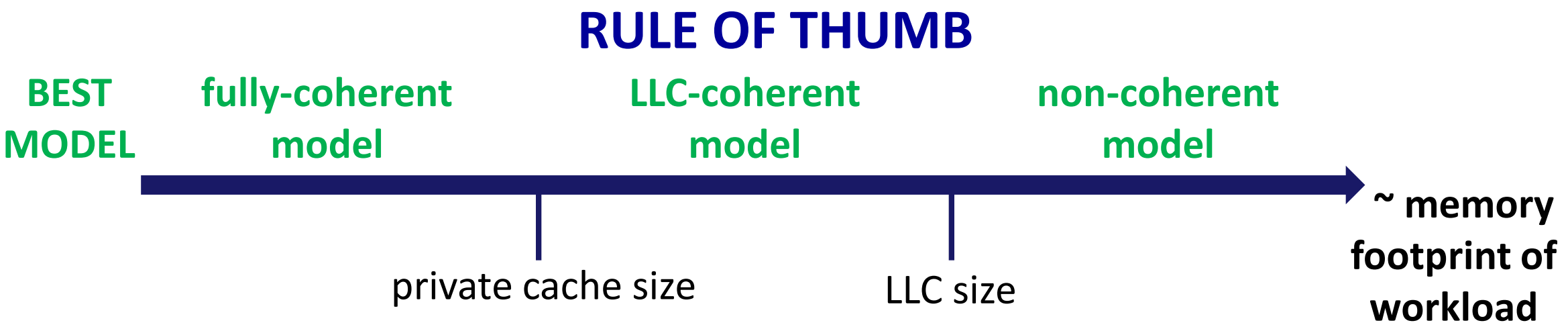
Accelerator Models: LLC Coherent

- Coherent with LLC only
  - processors' private caches are by-passed while talking with the LLC
- Programming requirements
  - race-free accelerator execution
  - flush processors' private caches prior to accelerator execution
- Implementation variants
  - first proposed by *[E. Cota et al., DAC '15]*
  - only 1 implementation in literature *[D. Giri et al., NOCS '18]*



Motivation: Why Different Coherence Models?

- The best choice of coherence model **varies at runtime** with the accelerator **workload size** and with the **number of active accelerators**
- LLC-coherent and fully-coherent models can significantly **reduce the number of off-chip memory accesses**



[D. Giri, P. Mantovani, and L. P. Carloni, *Accelerators & Coherence: An SoC Perspective*. IEEE MICRO, 2018.]

Heterogeneous Coherence: Experimental Setup

CHARACTERIZATION OF THE TARGET ACCELERATORS.

Accelerator	Memory Footprint	PLM (kB)	FPGA Resources LUT	FF	BRAM
FFT 1D	32kB - 256kB	40	7,537	4,310	10
Sort	128kB - 4MB	24	36,868	31,300	6
FFT 2D	256kB - 16MB	128	3,965	2,190	48
SPMV	25kB - 10MB	12	8,136	4,476	24

- **The ability to have perfectly balanced accelerator stages is highly dependent on the specific memory access patterns**
  - **as well as on the system interconnect and the memory hierarchy, including the selected cache-coherence model**

- **FFT1D**
  - streaming memory access
- **Sort**
  - no temporal locality, but in-place (i.e. in the PLM) data processing
- **FFT2D**
  - streaming memory access, but two phases with sequential dependency
- **SPMV**
  - asymmetric data reuse with irregular access pattern
  - very low compute-to-memory ratio

Results: Comparing the Speedup of Non-Coherent vs. LLC-Coherent Accelerators (Running *Standalone*)

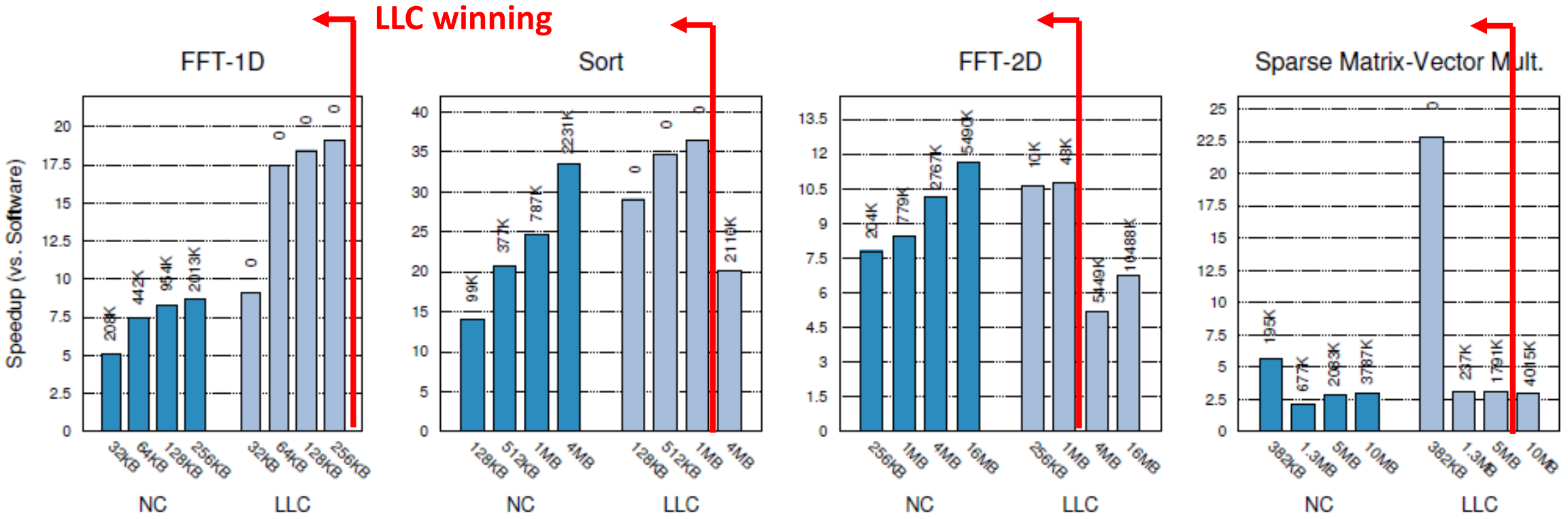


Fig. 4. Comparison of speedup w.r.t. software of non-coherent (NC) and LLC-coherent (LLC) accelerators. Bars are annotated with the memory access count.

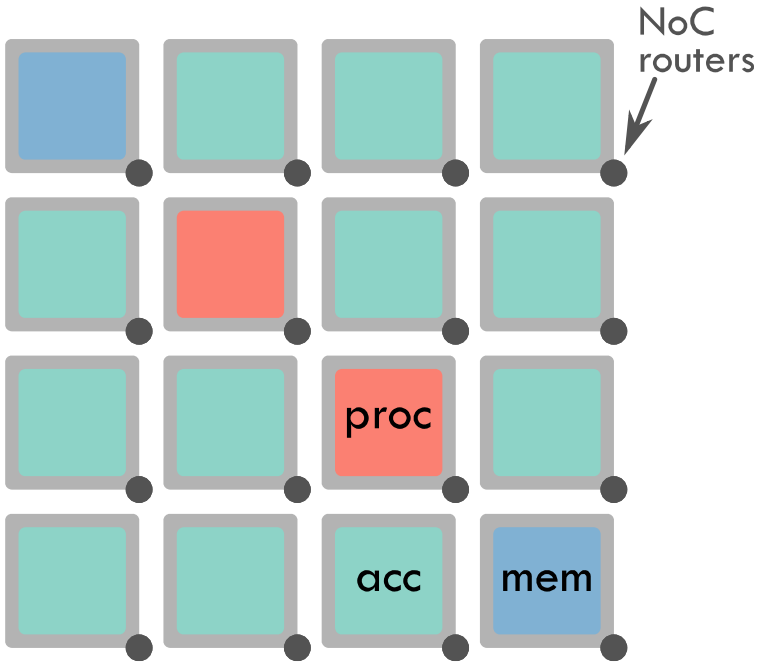
- **Compared to non-coherent accelerators, the relative speedup of LLC-coherent accelerators ranges between 0.5x and 4x**
  - the memory access count, instead, ranges from 0 to at most 2x (in worst-case scenario)
- **Confirmation of the benefits of runtime model selection based on footprint**

Contributions

- We propose a **runtime algorithm** to adaptively manage the cache coherence of accelerators
  - we show how to leverage the heterogeneity of cache-coherence models to improve the overall system performance.
- We evaluate the algorithm with:
  - our FPGA-based platform for rapid SoC prototyping, which is part of the Embedded Scalable Platform project
  - synthetic accelerators with a wide range of communication properties
  - synthetic application
    - varying number of concurrently active accelerators
    - variable memory footprint of the accelerators' workload

Our SoC Platform

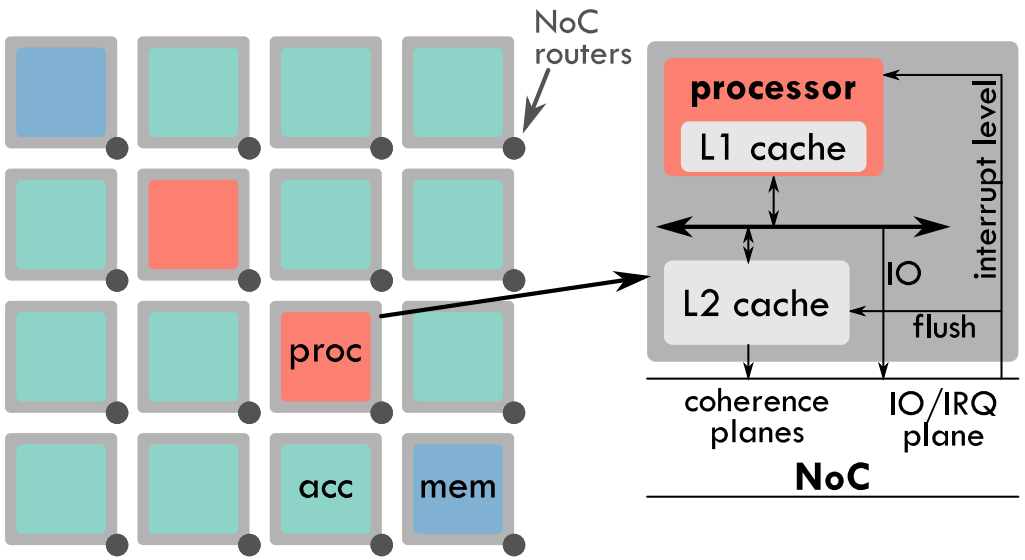
- Our design is based on an instance of **Embedded Scalable Platforms (ESP)**
  - socketed tiles
  - multi-plane NoC
  - easy integration and reuse of heterogeneous components
  - capable of running multi-processor and multi-accelerator applications on Linux SMP
  - support for all three cache-coherence models for accelerators



[Giri et al., NOCS '18]

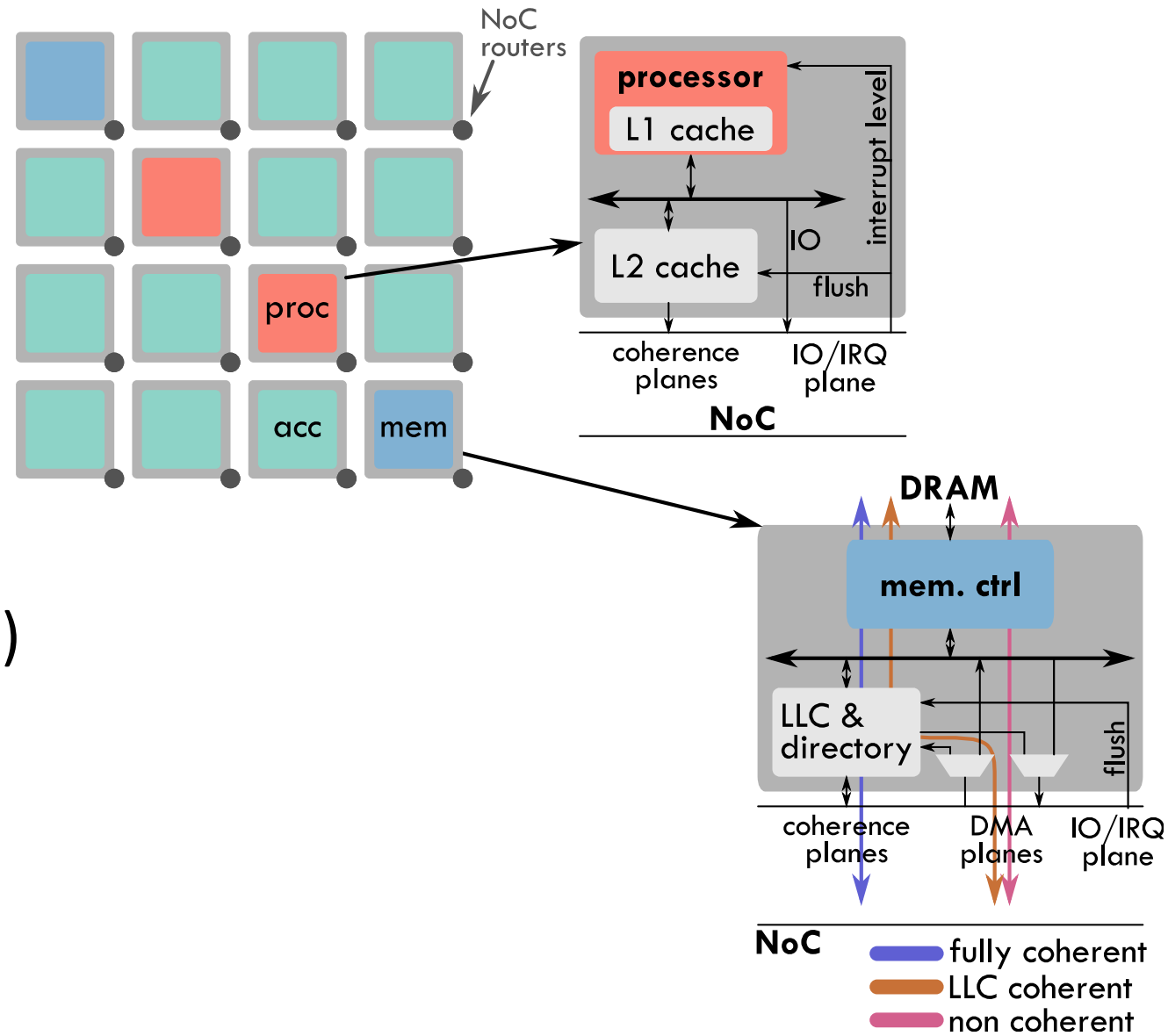
Processor Tile

- Main components
  - single-core processor tiles, with private L2 cache
- In this work
  - up to 2 processor tiles
  - 64KB private caches
  - off-the-shelf processor with L1 write-through caches: Leon3



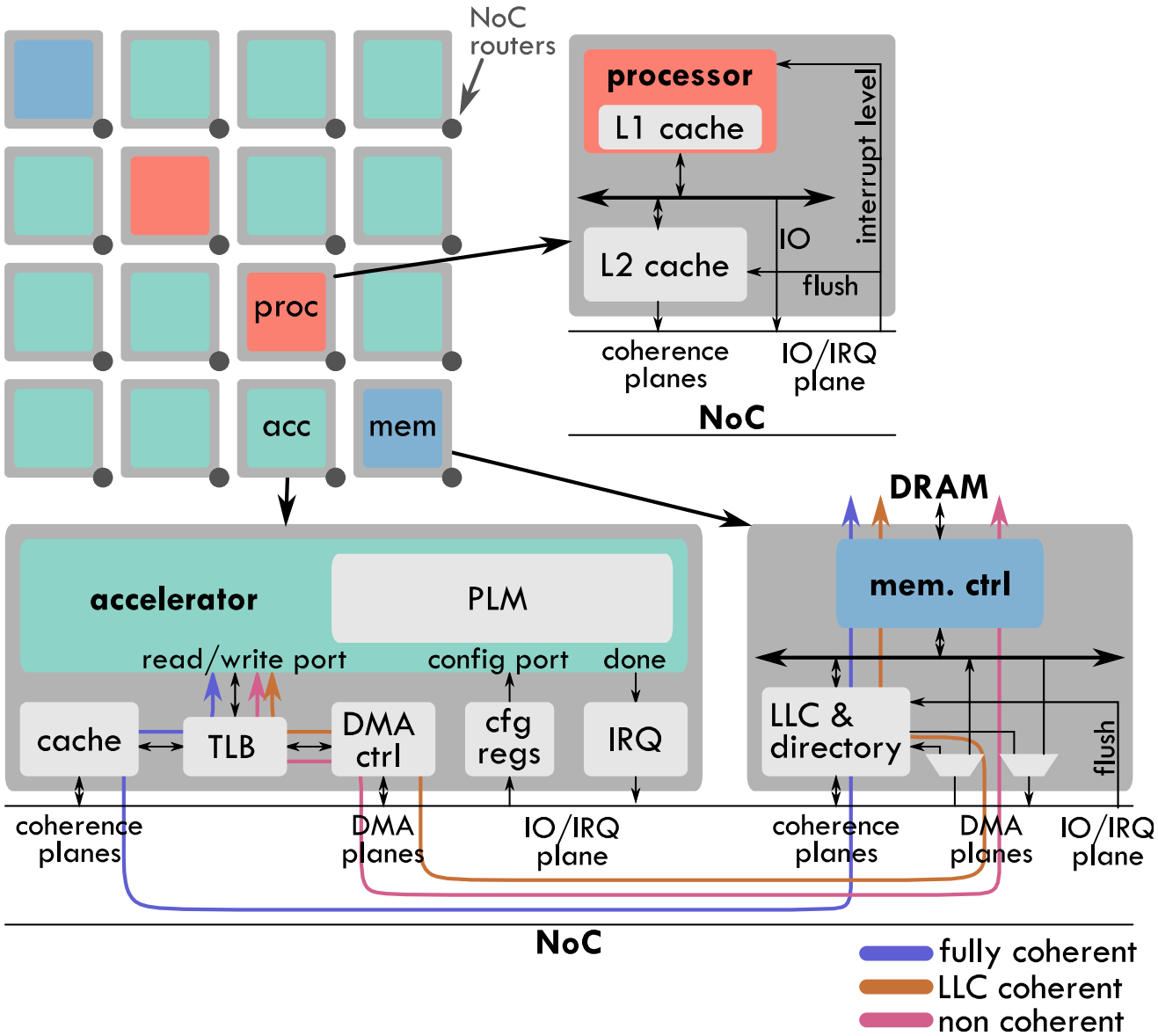
Memory Tile

- Main components
  - memory controller
  - LLC and directory
    - can be split over multiple tiles
- In this work
  - 2 memory tiles
  - 2MB aggregate LLC (1MB per tile)



Accelerator Tile

- Main components
  - any accelerator complying with a simple interface
  - a small TLB
  - a DMA controller and/or a private cache
- Support for *run-time selection* of coherence model
  - selection granularity: possible at each accelerator invocation
  - selection method: one I/O write to the configuration registers



The Proposed Algorithm for Adaptive Management of Accelerator Coherence

- Executed by the device driver at each accelerator's invocation
- Selects the cache-coherence model for the accelerator
- Static inputs: 4
- Dynamic inputs: 4

```
1  if (footprint < PRIVATE_CACHE_SIZE)
2    if (n_fully_coherent < MAX_FULLY_COHERENT)
3      coherence = FULLY_COHERENT;
4    else
5      coherence = LLC_COHERENT;
6
7  else if ((current_llc_footprint + footprint)
8             > LLC_SIZE)
9    coherence = NON_COHERENT;
10
11 else if (n_acc_on_llc_or_fully_coherent
             >= N_MEM_TILES * MAX_ACC_PER_LLC)
12   coherence = NON_COHERENT;
13
14 else
15   coherence = LLC_COHERENT;
```

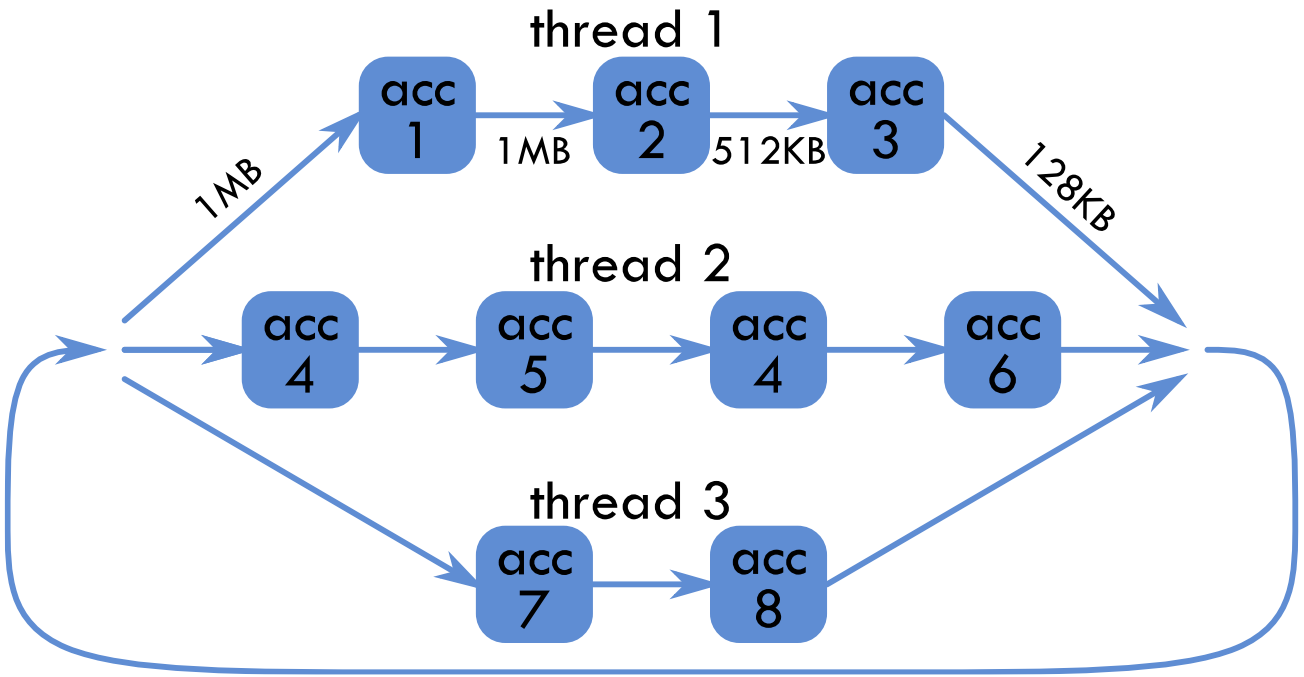
Synthetic Accelerators

- An accelerator is characterized by its communication properties
  - we defined 8 parameters to describe the communication properties
  - we designed a “master accelerator” with parametrizable communication properties
  - we generated 12 accelerators with a wide range of communication

Accelerator ID	1	2	3	4	5	6	7	8	9	10	11	12
Access pattern	stream	stride	stream	irreg	stream	stride	stream	irreg	stream	stride	stream	irreg
Access fraction	1	1	1	1	1	1	1	1/4	1	1	1	1/16
Burst length	64	4	32	4	128	8	64	4	16	4	32	4
Stride length	0	256	0	0	0	32	0	0	0	512	0	0
Compute-mem ratio	1	1	2	4	4	2	8	2	4	4	2	1
Reuse factor	2	4	1	1	4	1	1	4	1	2	4	1
In-place	no	no	yes	yes	no	yes	no	no	yes	no	no	yes
In-out ratio	1	2	4	1	2	4	1	2	4	1	2	4

Synthetic Application

- Application with multiple phases
  - variable memory footprints of the accelerators' workloads
  - variable number of concurrently active accelerators



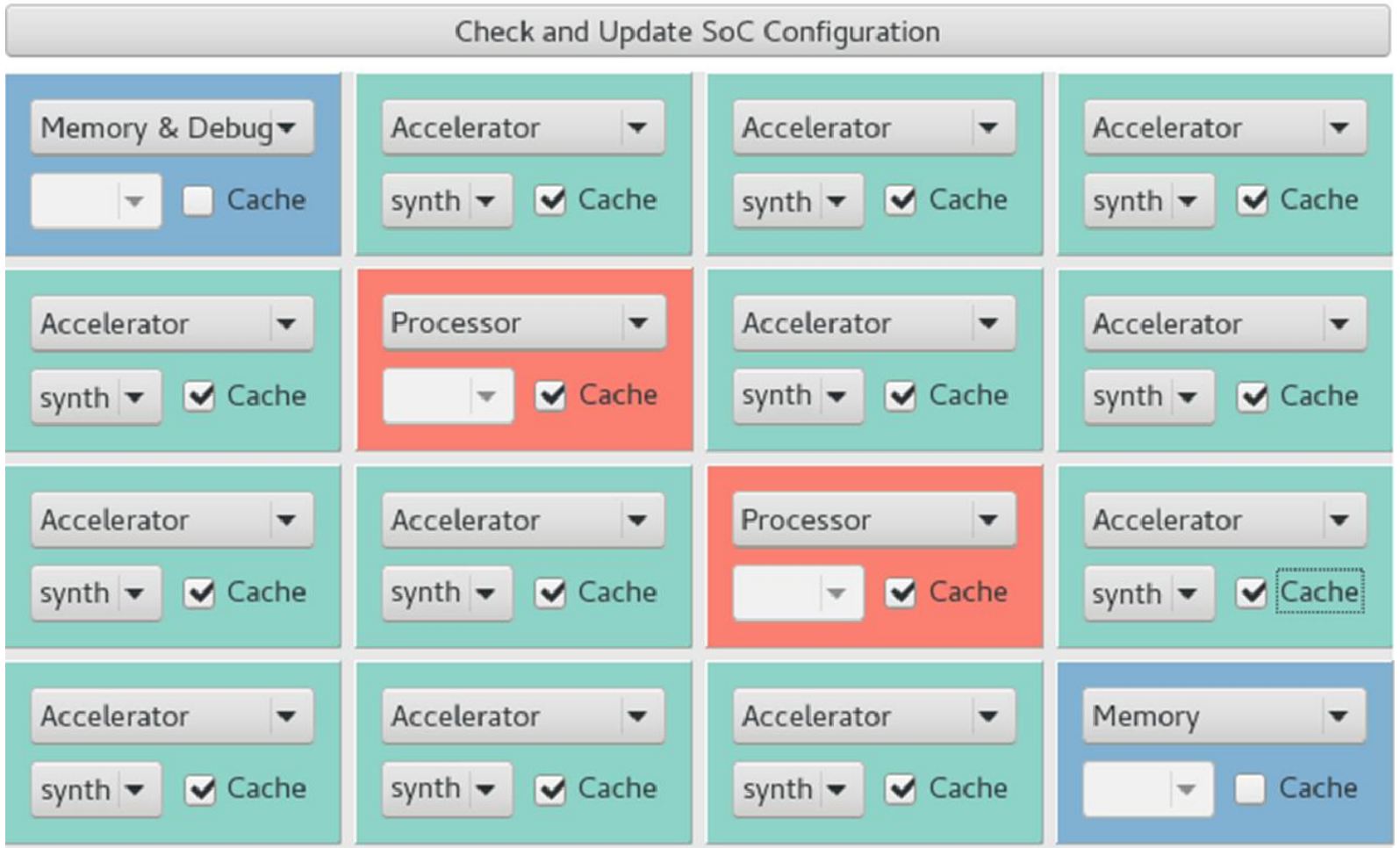
Sample of a possible app phase

App phases	Memory footprints sizes	Max active accelerators
1	variable	1
2	large	1
3	small	1
4	variable	6
5	large	6
6	small	6
7	variable	12
8	large	12
9	small	12

Phases in our app

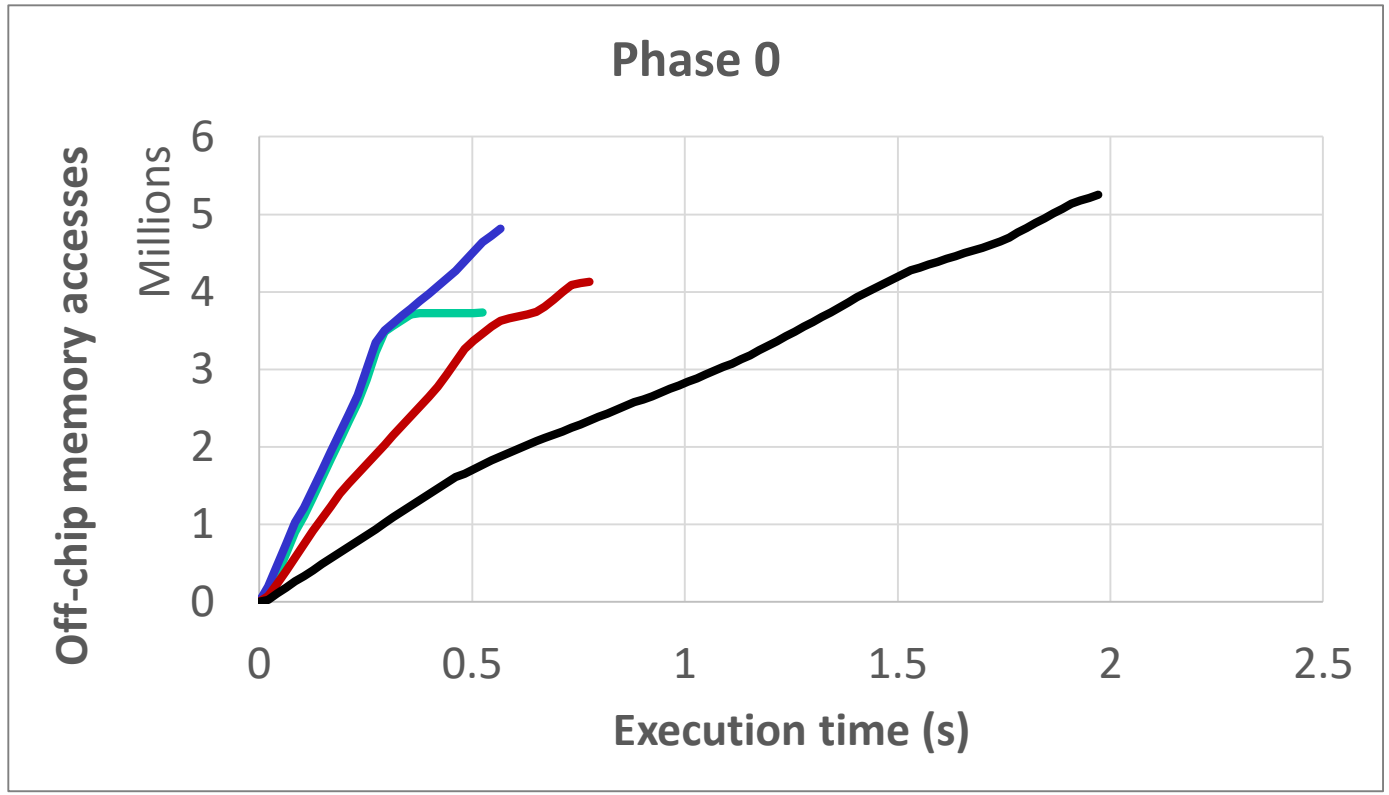
Evaluation SoC

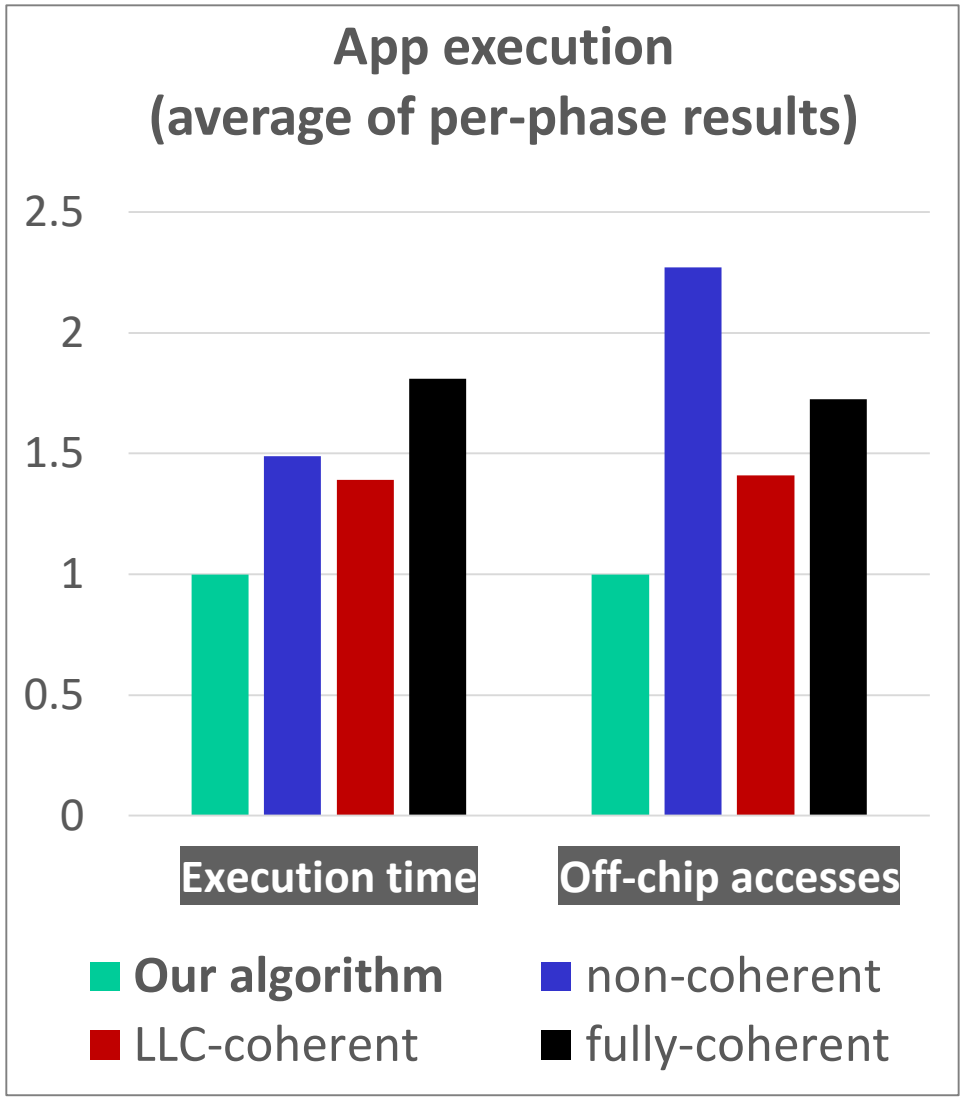
- ESP's GUI
  - the CAD flow from GUI to FPGA bitstream is fully automated
- We deployed this SoC on FPGA and we executed the synthetic application on Linux SMP



Results

- Our algorithm reduces:
  - the **execution time** by at least 40%
  - the **off-chip accesses** by at least 30%





Conclusions

- We showed how to exploit the heterogeneity of cache-coherence models
  - We proposed a runtime algorithm to select the proper cache-coherence model at each accelerator's invocation
- Heterogeneity of cache-coherence models for accelerators can:
  - lead to speedups of at least 40%
  - reduce the off-chip accesses by a minimum of 30%
- The algorithm is general enough to apply to any SoC
  - its inputs are: number of active accelerators, caches capacity, memory footprint of the accelerator workloads

Some Recent Publications

Available at www.cs.columbia.edu/~luca

1. L. P. Carloni. The Case for Embedded Scalable Platforms DAC 2016. (Invited Paper).
2. L. P. Carloni. From Latency-Insensitive Design to Communication-Based System-Level Design The Proceedings of the IEEE, Vol. 103, No. 11, November 2015.
3. E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. An Analysis of Accelerator Coupling in Heterogeneous Architectures. DAC 2015.
4. P. Mantovani, E. Cota, K. Tien, C. Pilato, G. Di Guglielmo, K. Shepard and L. P. Carloni. An FPGA-Based Infrastructure for Fine-Grained DVFS Analysis in High-Performance Embedded Systems. DAC 2016.
5. P. Mantovani, E. Cota, C. Pilato, G. Di Guglielmo and L. P. Carloni. Handling Large Data Sets for High-Performance Embedded Applications in Heterogeneous Systems-on-Chip. CASES 2016.
6. L. Piccolboni, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators. ACM Transactions on Embedded Computing Systems, 2017.
7. C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip. IEEE Trans. on CAD of Integrated Circuits and Systems, 2017.
8. D. Giri, P. Mantovani and L. P. Carloni. NoC-Based Support of Heterogeneous Cache-Coherence Models for Accelerators, NOCS, 2018.
9. D. Giri, P. Mantovani, and L. P. Carloni, Accelerators & Coherence: An SoC Perspective. IEEE MICRO, 2018.