Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs

Joseph Zuckerman
Davide Giri
Jihye Kwon
Paolo Mantovani
Luca P. Carloni

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

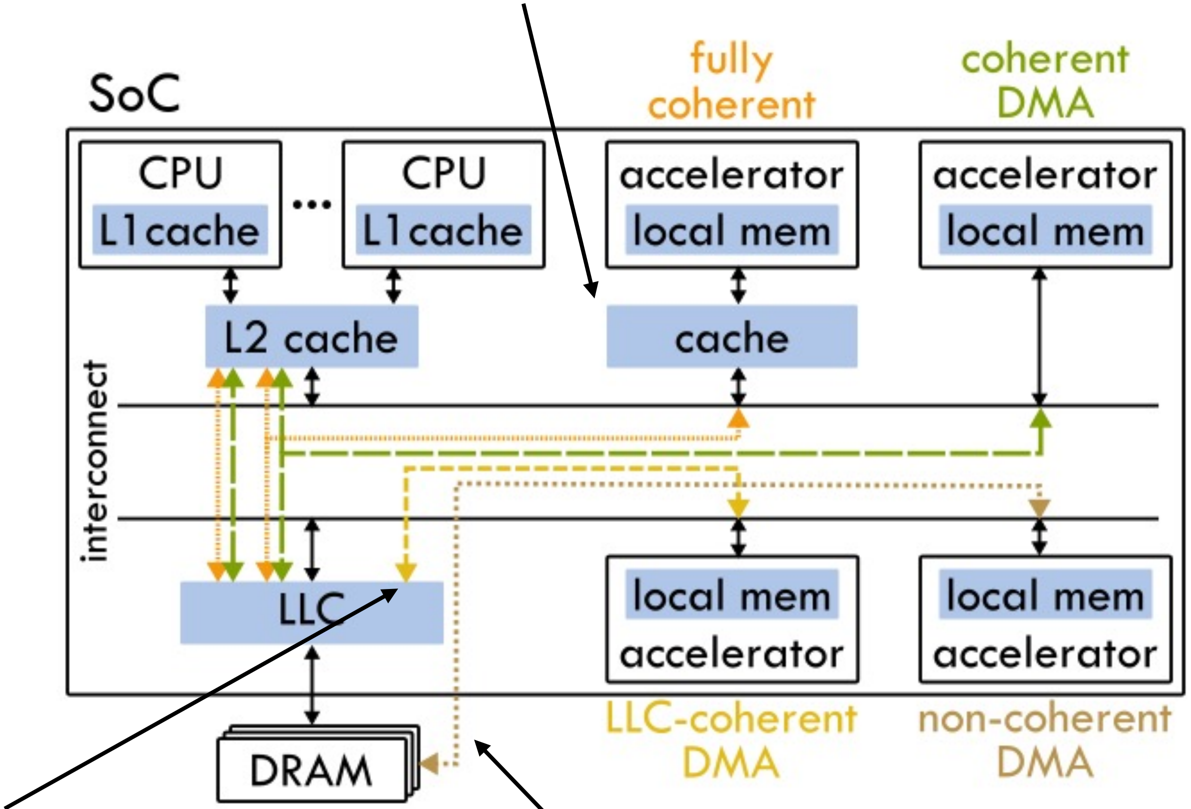
CS@CU

MICRO '21

Accelerators + Coherence

- This work focuses on fixed-function loosely-coupled accelerators (LCAs)
    - Sit on the system interconnect
    - Execute coarse-grain tasks
    - Invoked with a device driver
    - Configurable, but not programmable
- Several cache coherence modes for LCAs in literature
- Some enforced in HW, some in SW, some hybrid
- Each mode has a set of tradeoffs
    - Required flushes
    - Transaction overhead
    - Data transfer size
- Most solutions are fixed at design time

Accelerator has its own private cache



Accelerator performs DMA to LLC

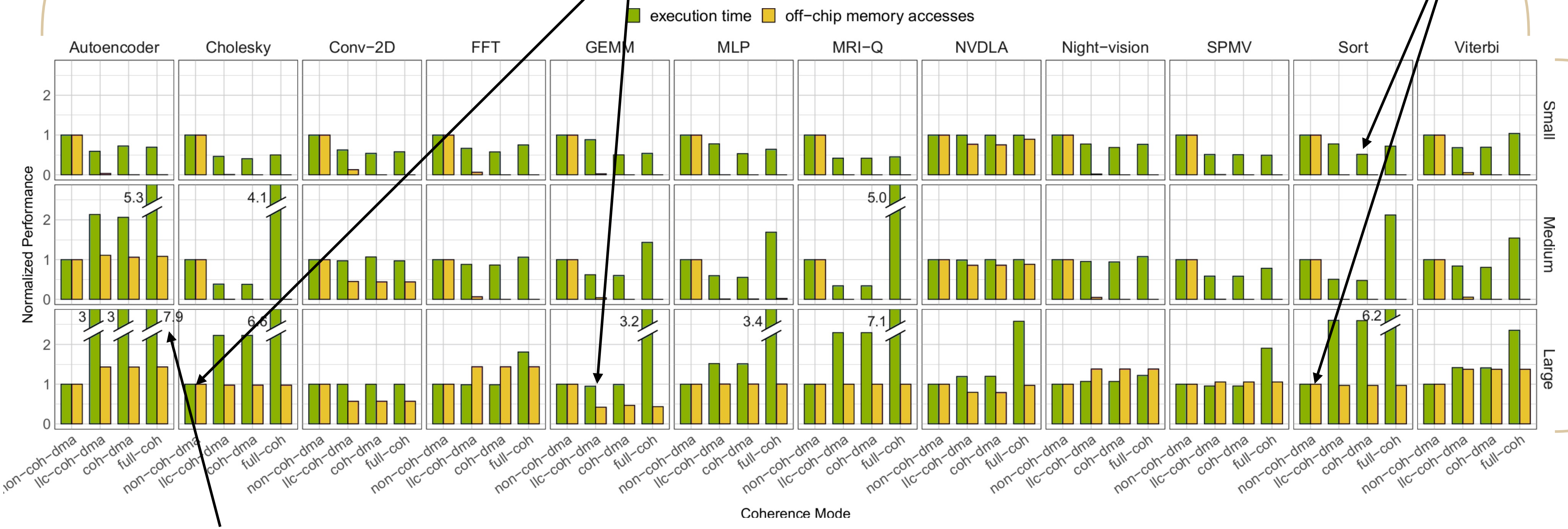
Accelerator bypasses entire cache hierarchy

Motivation: Heterogeneous Coherence Needs

12 Accelerators

Different winners across accelerators

Different winners for S & L sizes



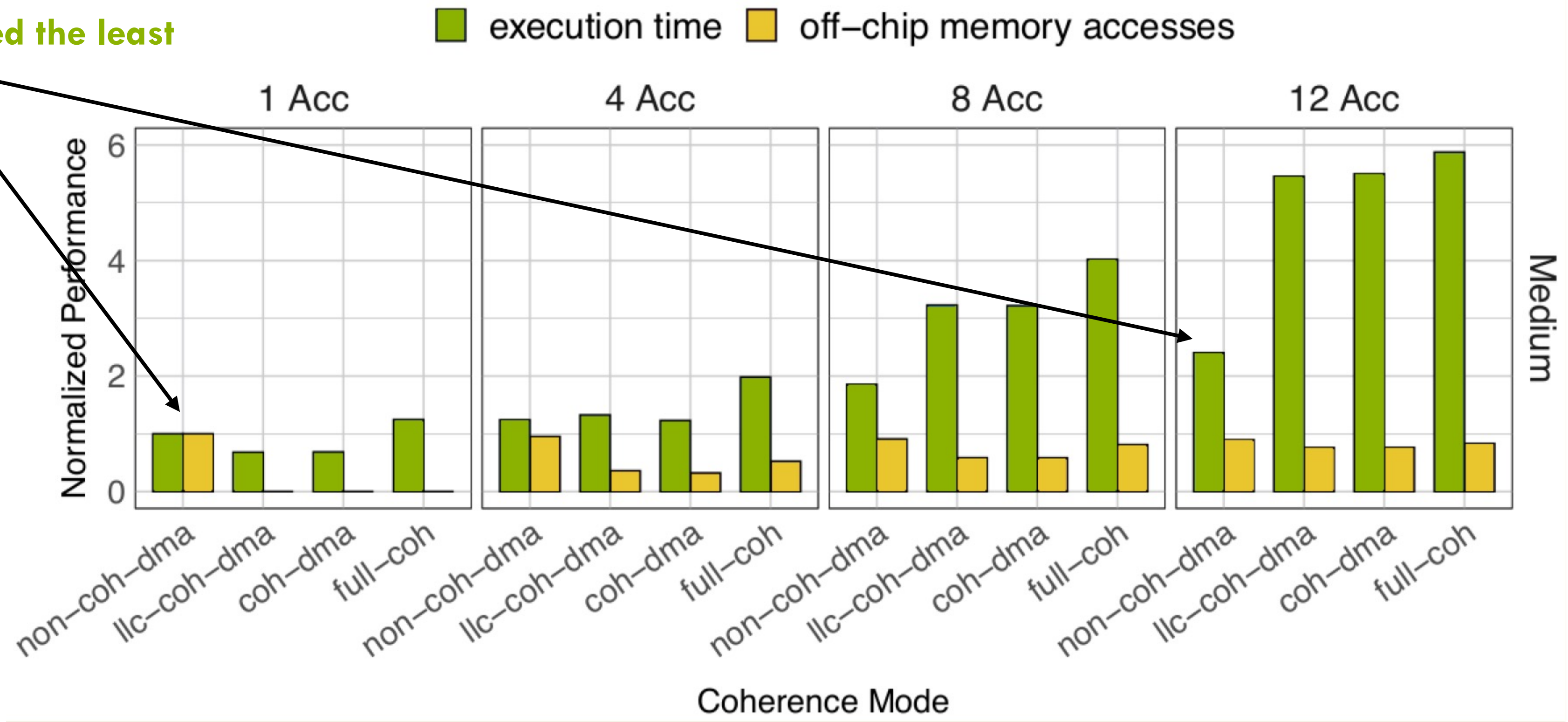
3 Workload Sizes

8x slowdown for full coherence

No best coherence mode!

Motivation: Impact of Contention

Non-coherent affected the least



- Performance also affected by contention on shared resources
- **Dynamic system status can change the best coherence choice**

Runtime Coherence Selection

- **We propose that SoCs should support multiple coherence modes for accelerators and the seamless selection from them at runtime.**

- Infeasible to expect application developers to manage this decision
  - Large state space
  - Varies based on architectural parameters of SoC
    - cache sizes, channels to memory, coherence protocol, type of interconnect, etc.
  - Workloads can change over time
  - Accelerators can be configured with different parameters

→ Use learning to enable the **automatic discovery of best coherence decisions**

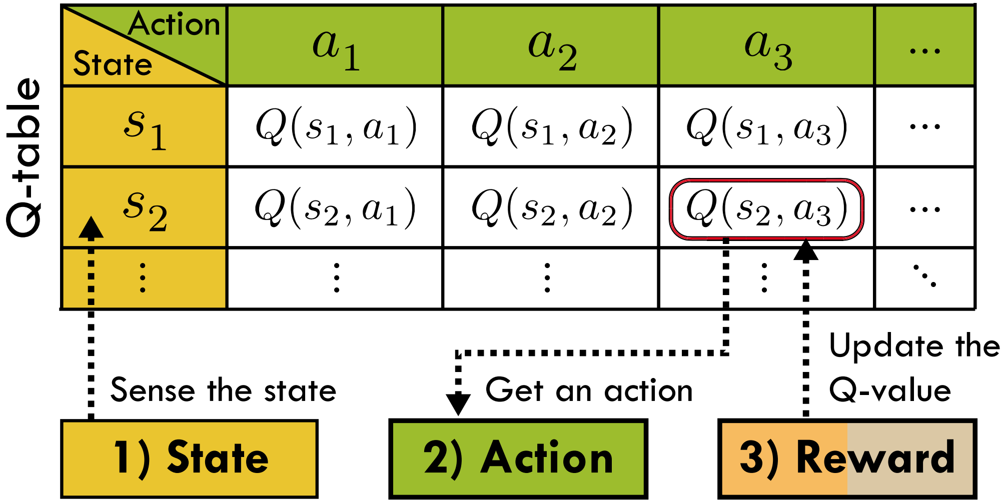
- An appropriate learning approach should:
  - Train online during normal SoC operation
  - Require no prior knowledge of the SoC architecture nor training data
  - Be able to continuously update as new workloads and states are encountered

Q-learning

- In *reinforcement learning (RL)*, an autonomous agent learns behaviors through trial-and-error interactions with the environment

- *Q-learning* is a type of RL that maintains a *Q-table* of *Q-values* for state-action pairs
  - Updated using the “reward” received by taking a particular action from that state
  - Can optimize for a desired objective by defining *a reward function*

- Cohmeleon’s Q-learning model
  - **States:** status of the SoC (number of accelerators running, size of workloads, coherence modes in use)
  - **Actions:** available accelerator coherence modes
  - **Rewards:** weighted combination of execution time and off-chip memory accesses



Q-table

Action State	a_1	a_2	a_3	$\cdots$
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	$Q(s_1, a_3)$	$\cdots$
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	$Q(s_2, a_3)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Sense the state

Get an action

Update the Q-value

1) State

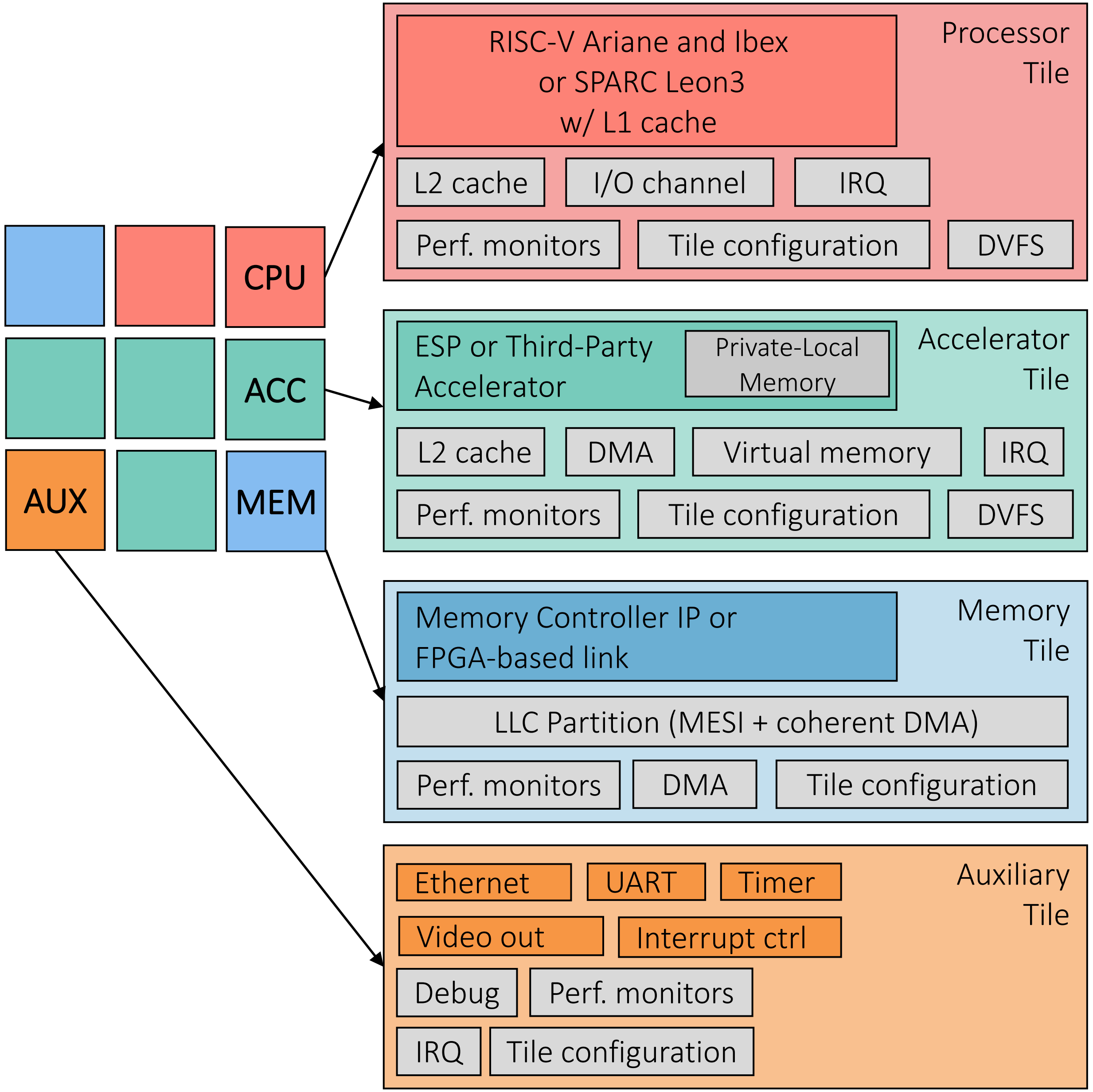
2) Action

3) Reward

Cohmeleon: Implementation

- Built on top of ESP (esp.cs.columbia.edu)
  - Open-source research platform for agile SoC design
  - Runtime selection of coherence modes for accelerators
  - “Push-button” generation of FPGA prototypes

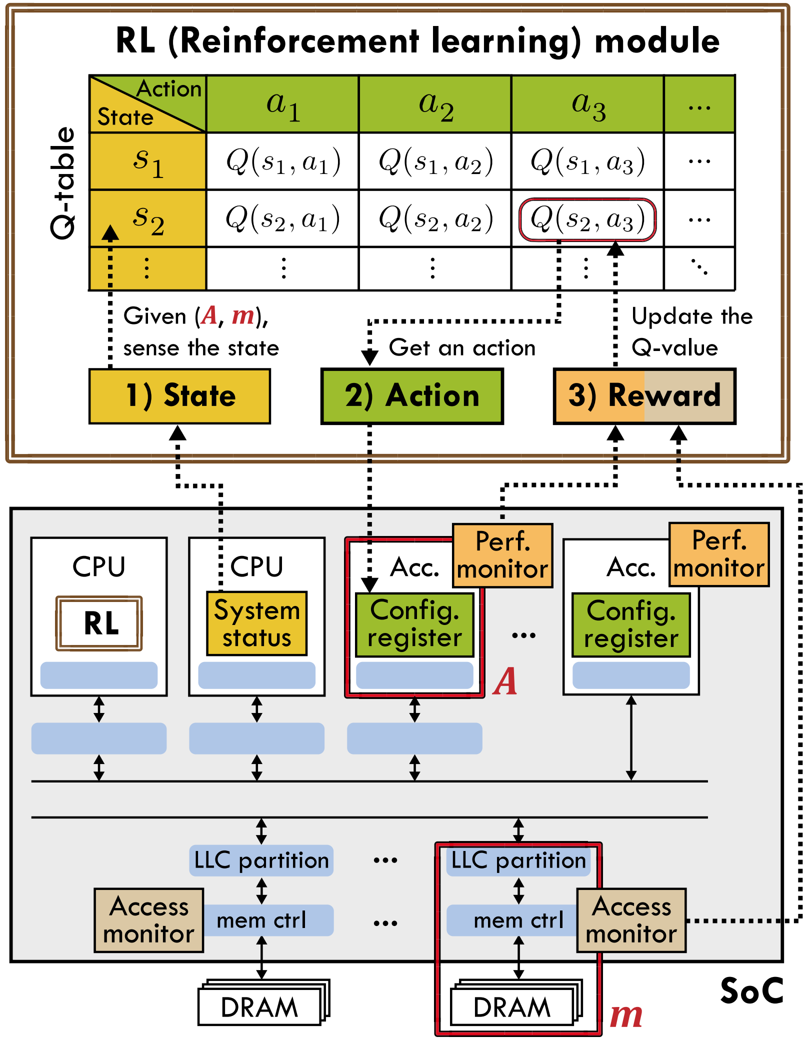
- ESP Architecture
  - Scalable 2D-mesh multi-plane NoC
  - Tile sockets provide platform services



Cohmeleon: Implementation

- Status tracking and RL added to ESP accelerator invocation API
  - Runs on Leon3 processor cores

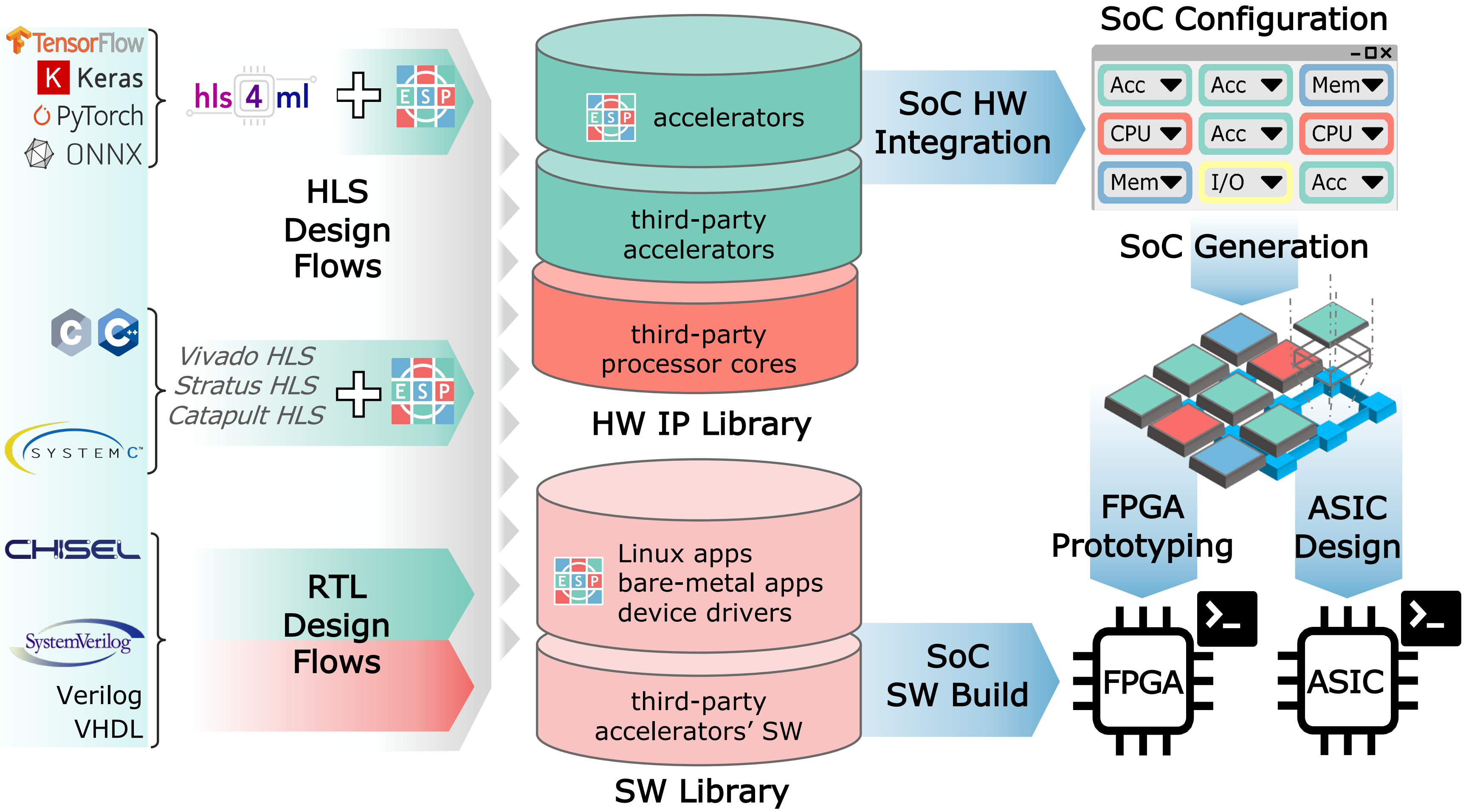
- Created a HW monitoring system for accessing performance counters from SW
  - Use accelerator cycles and memory accesses for evaluation
  - Negligible overhead



State \ Action	a_1	a_2	a_3	⋯
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	$Q(s_1, a_3)$	⋯
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	$Q(s_2, a_3)$	⋯
⋮	⋮	⋮	⋮	⋱

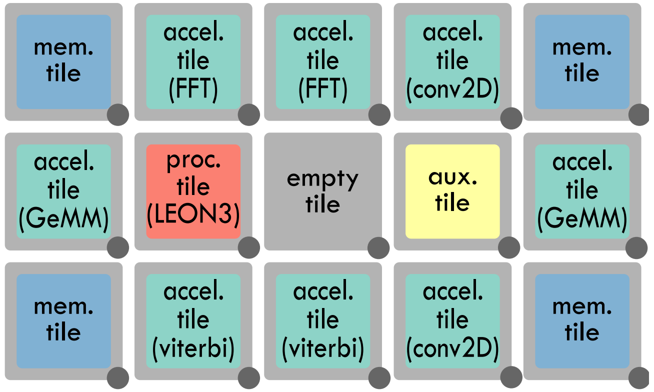
Evaluation: Setup

- ESP provides several design flows for accelerators with automatic SoC integration
- Designed 7 different many-accelerator SoCs for FPGA

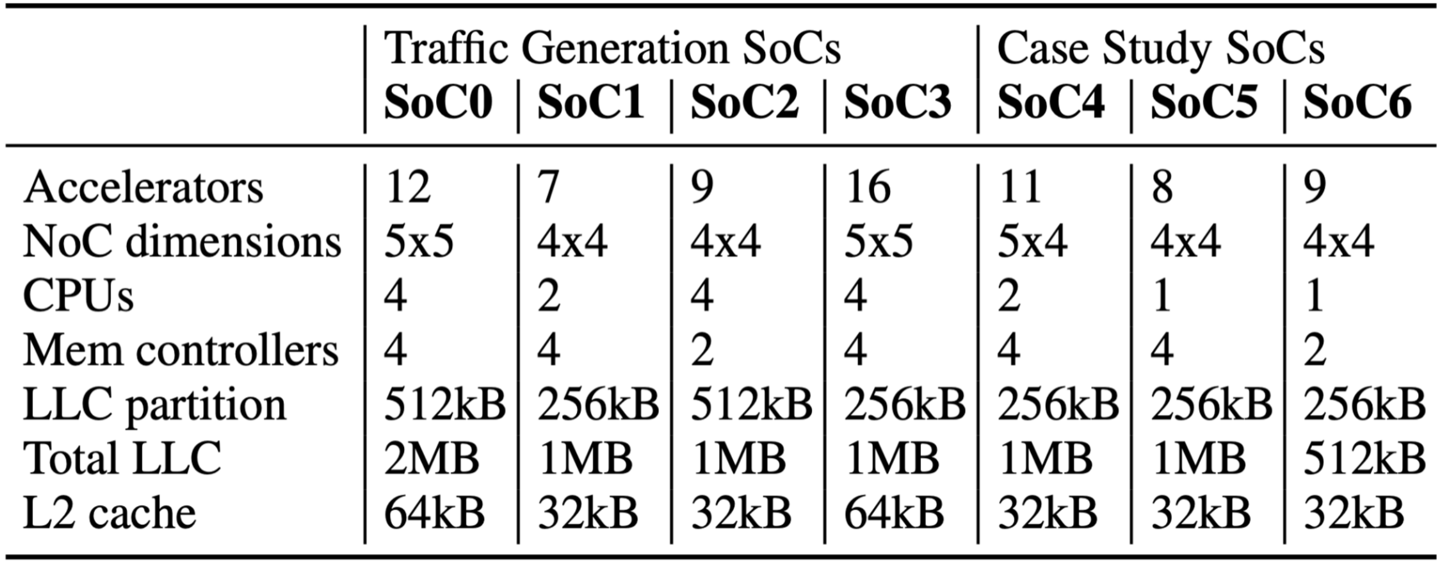


Evaluation: Setup

- 4 SoCs based on a highly-configurable traffic generator accelerator
- 3 SoCs using real accelerators
  - SoC with 10 of the accelerators from the motivation section
  - SoC for **autonomous driving** with FFT + Viterbi for V2V communication and GeMM+Conv2D for CNN execution
  - SoC for **computer vision** with Night-vision, MLP, and Autoencoder accelerators
  - All accelerators are available open source in ESP
- Developed a multithreaded application for each SoC to invoke accelerators in a diverse set of ways
  - Organized in "phases"



mem. tile	accel. tile (FFT)	accel. tile (FFT)	accel. tile (conv2D)	mem. tile
accel. tile (GeMM)	proc. tile (LEON3)	empty tile	aux. tile	accel. tile (GeMM)
mem. tile	accel. tile (viterbi)	accel. tile (viterbi)	accel. tile (conv2D)	mem. tile



	Traffic Generation SoCs				Case Study SoCs		
	SoC0	**SoC1**	**SoC2**	**SoC3**	**SoC4**	**SoC5**	**SoC6**
Accelerators	12	7	9	16	11	8	9
NoC dimensions	5x5	4x4	4x4	5x5	5x4	4x4	4x4
CPUs	4	2	4	4	2	1	1
Mem controllers	4	4	2	4	4	4	2
LLC partition	512kB	256kB	512kB	256kB	256kB	256kB	256kB
Total LLC	2MB	1MB	1MB	1MB	1MB	1MB	512kB
L2 cache	64kB	32kB	32kB	64kB	32kB	32kB	32kB

Evaluation: Baselines

Design time

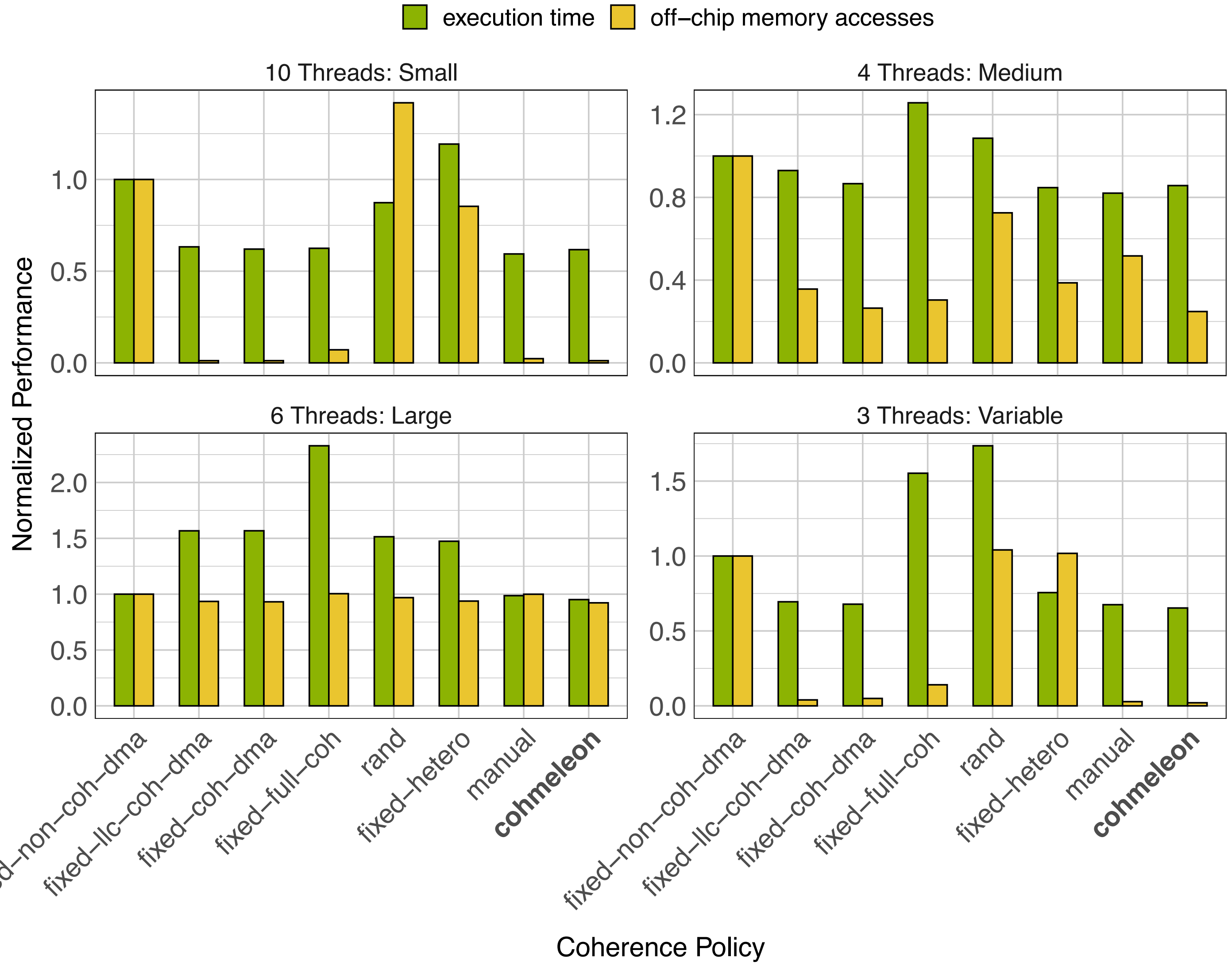
- *Fixed-homogeneous :* single coherence mode for the entire SoC
- *Fixed-heterogeneous*:* single coherence mode for each accelerator
  - Chosen by profiling at design time

Run time

- *Random:* runtime selection without intelligence
- *Manually-tuned algorithm*:* heuristic incorporating information about SoC status
  - Developed by collecting significant data from tens of thousands of accelerator invocations

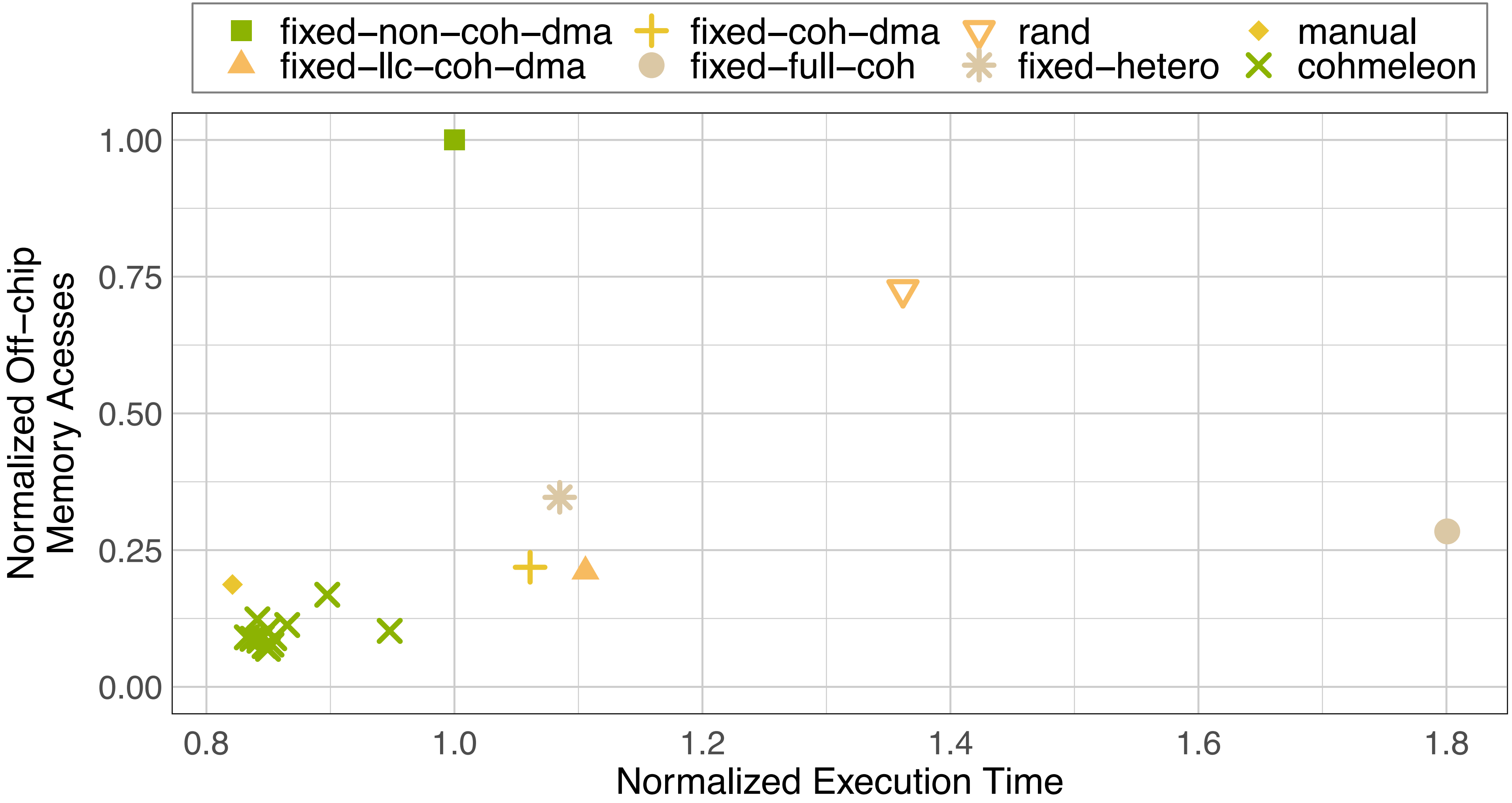
**state-of-the-art*

Results: Phases



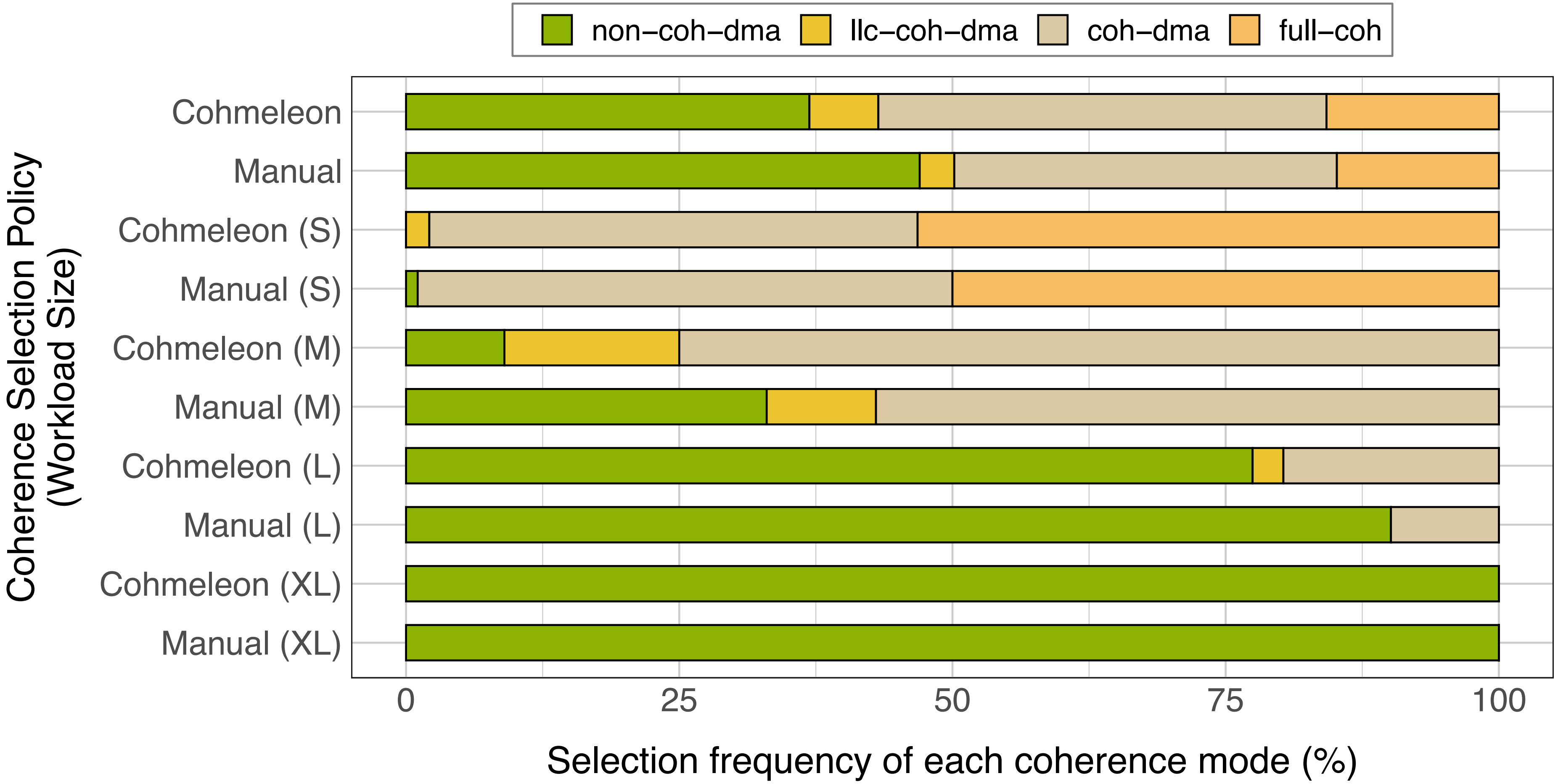
Different design-time winners for each phase, but Cohmeleon always matches best performance!

Results: Exploration of Reward Function



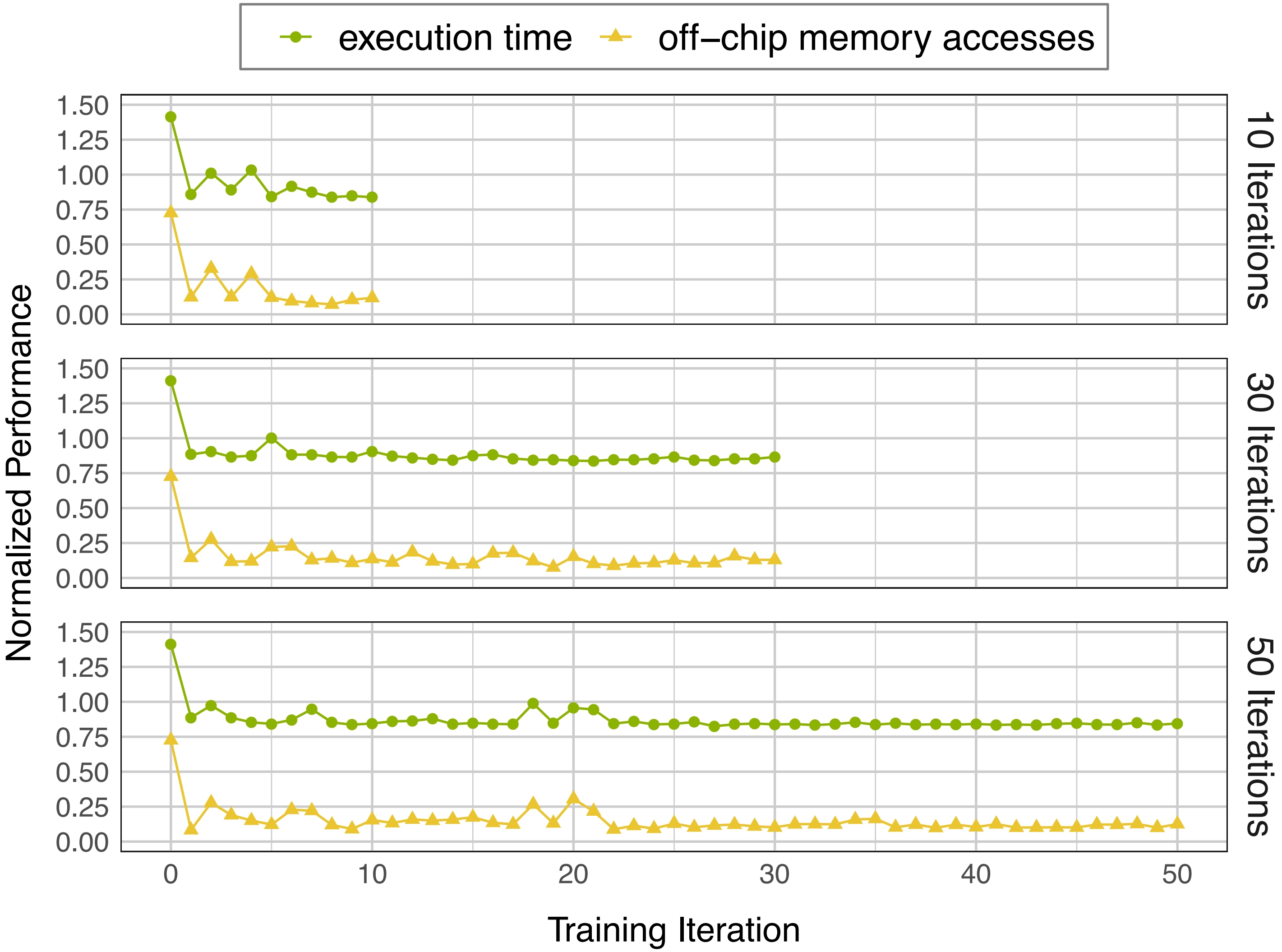
Changing reward function does not substantially impact performance → minimal tuning required

Results: Coherence Selection Breakdown



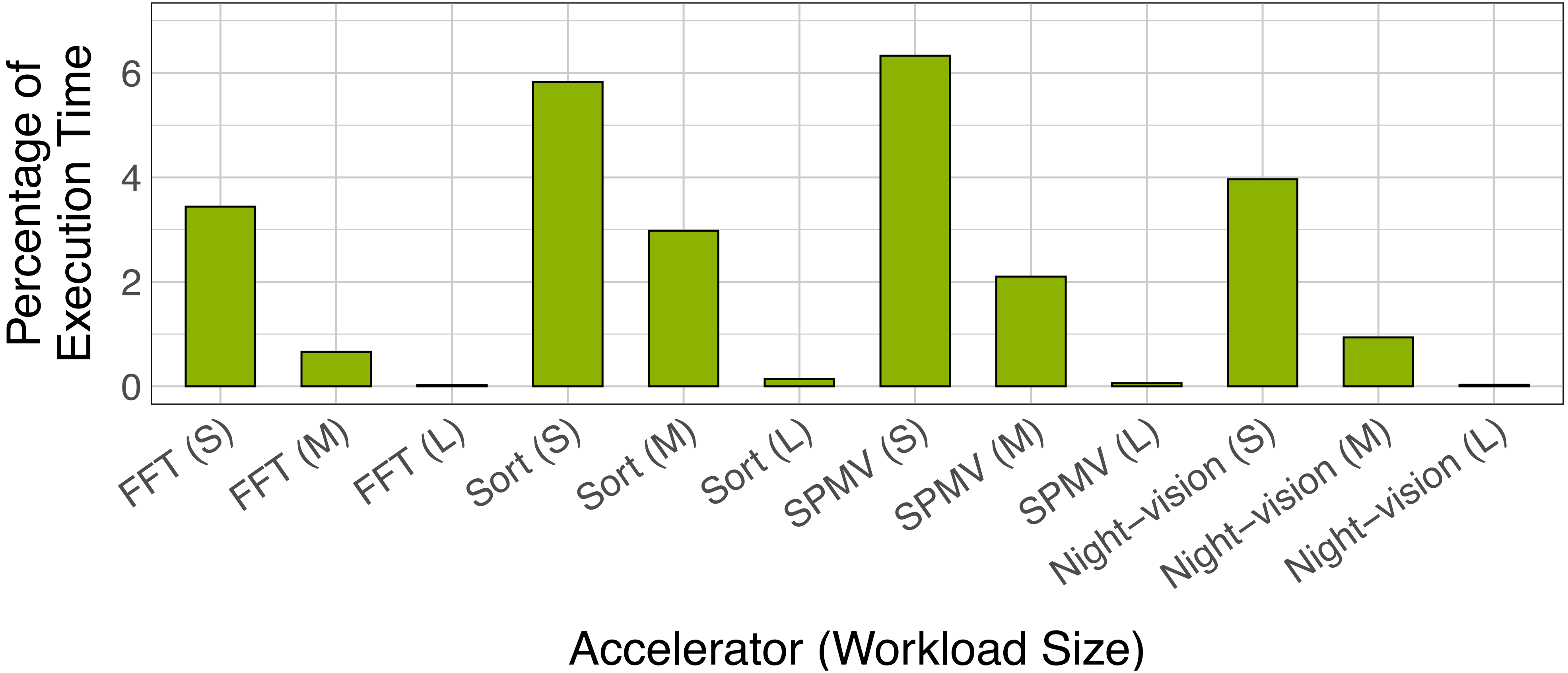
Cohmeleon learns a similar form to the manual algorithm, but relies on non-coherent less, while preserving performance

Results: Training Time



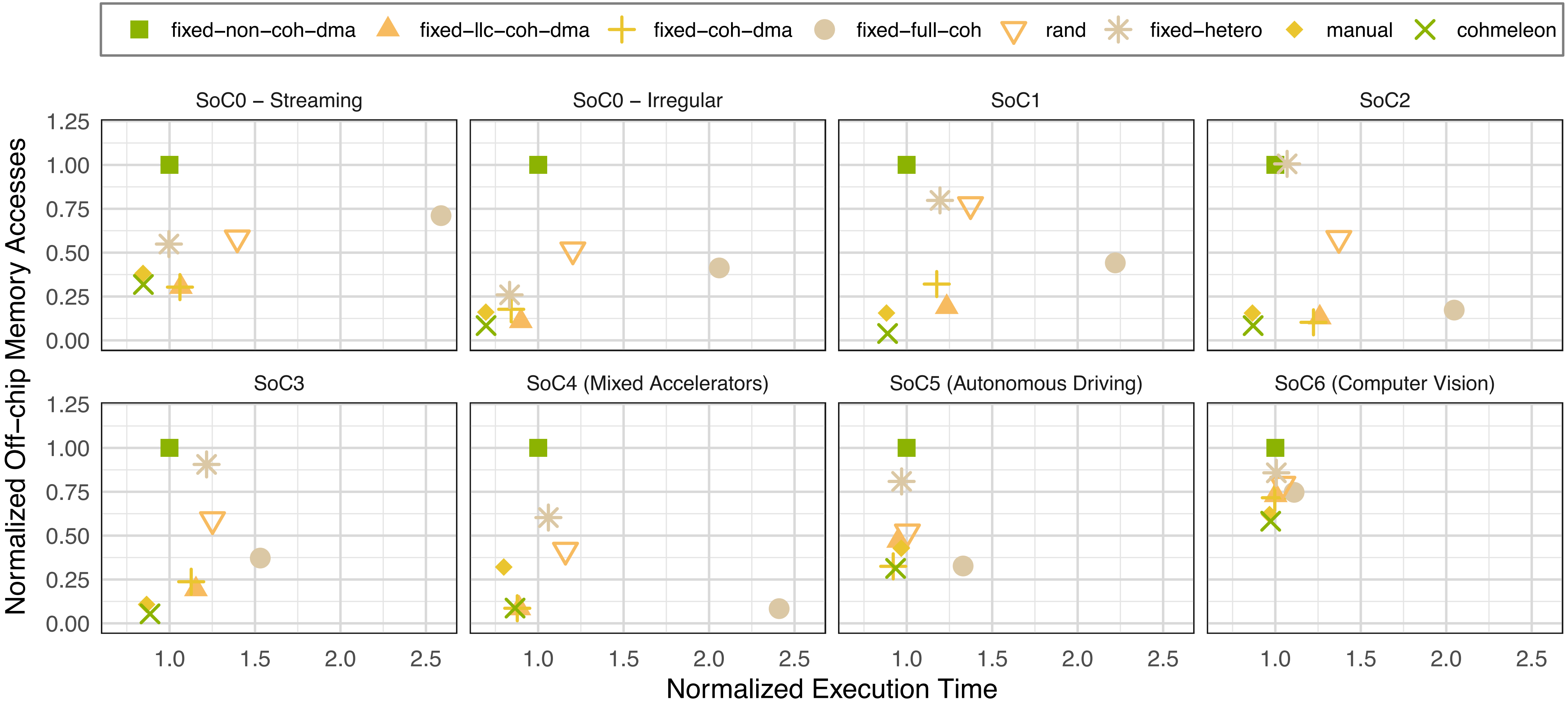
Quick improvements, some oscillation as exploration continues, but fast convergence

Results: Overhead



Execution time overhead is negligible for large workloads and small for others

Results: All SoCs



Avg speedup of 38% with a 66% reduction of off-chip memory accesses when compared to design-time solutions.

Conclusions

- Accelerator performance can vary greatly based on coherence modes
- SoCs should support multiple coherence modes for optimal performance
- Reinforcement learning can be used to automatically manage coherence mode decisions
- With little overhead, Cohmeleon provides significant performance benefits for multiple objectives
- We released Cohmeleon as a part of the open-source ESP project, and successfully completed MICRO artifact evaluation

Thank you!

Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs

Joseph Zuckerman, Davide Giri, Jihye Kwon, Paolo Mantovani, Luca P. Carloni

Session 4A: Parallelism
Tuesday, October 19
3:15-3:30pm EDT

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK
CS
@CU