VANYA YANEVA

vanya.yaneva@ed.ac.uk

- PhD Candidate in Software Testing at the University of Edinburgh.

- My background is in High Performance Computing.

- My PhD work is on **accelerating software testing** by executing tests *in parallel* on the GPU.

- I am interested in:
    - functional software testing
    - parallel architectures and programming models
    - compiler optimisations and compiler-based tools

THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

ACCELERATED FINITE STATE MACHINE TEST EXECUTION USING GPUS

Vanya Yaneva

Arnav Kapoor, Ajitha Rajan, Christophe Dubach

THE UNIVERSITY *of* EDINBURGH
informatics

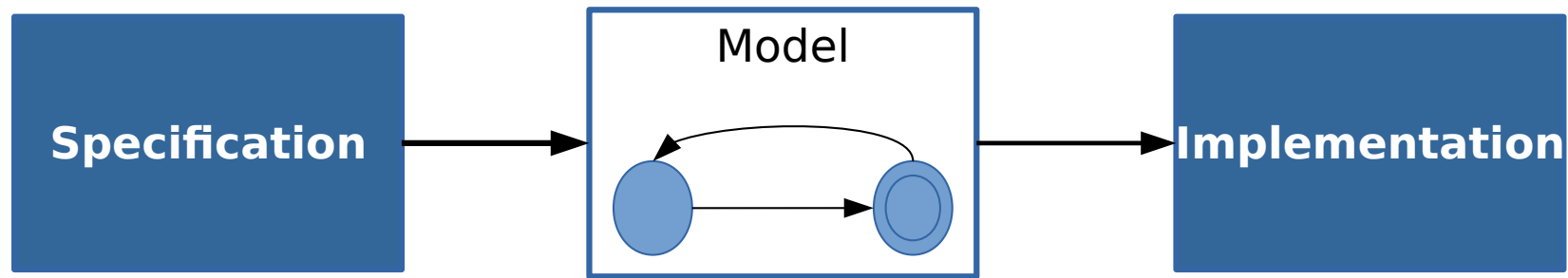INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
H Y D E R A B A D

EPSRC **Centre for Doctoral Training in**
Pervasive Parallelism

MODEL-BASED DEVELOPMENT

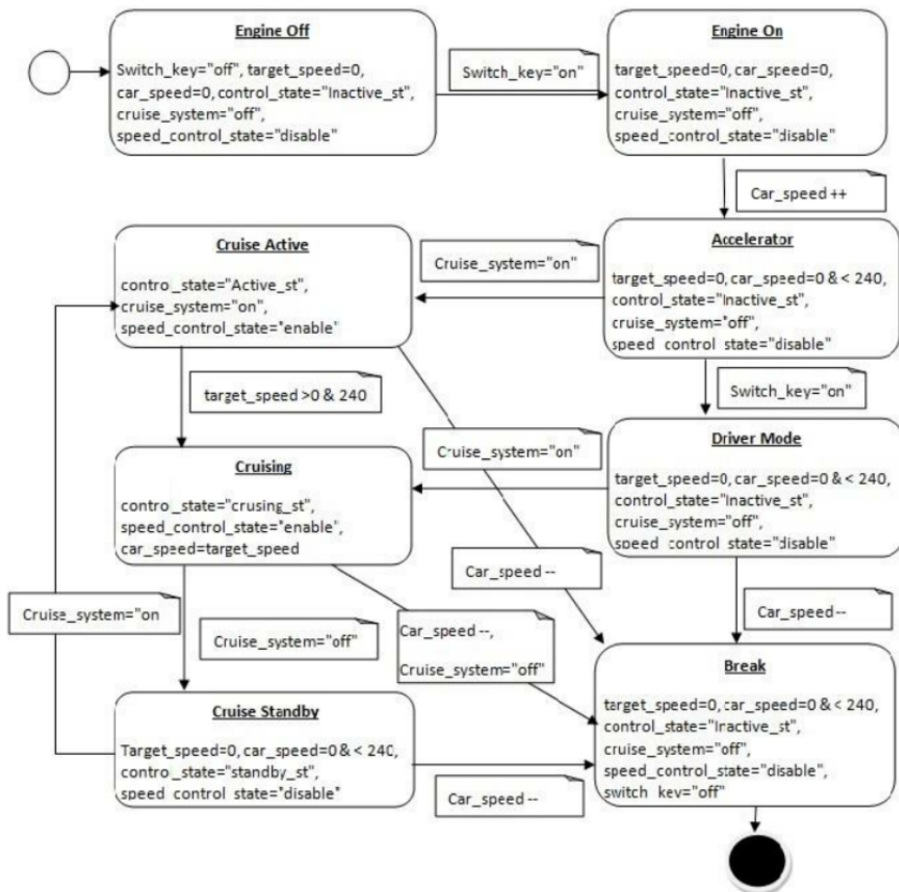*Software is **implemented** and **tested** based on a model.*

model = specification ?
Execute tests on the model

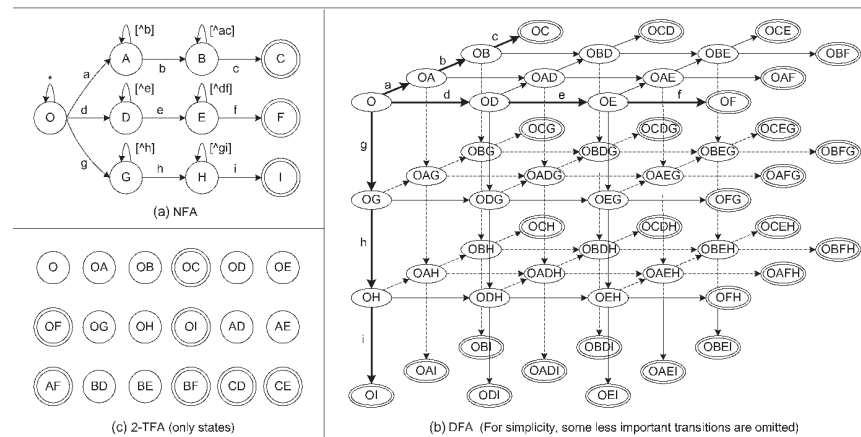There are many ways to define a model:
- UML • formal specification languages (Z, B, Alloy) • block/state charts
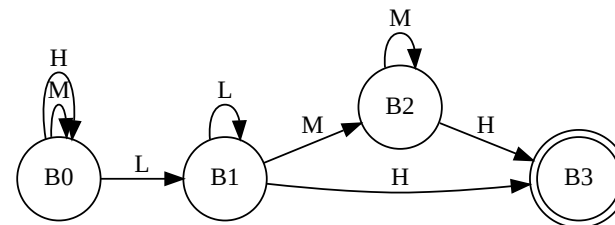
FINITE STATE MACHINES (FSMS)

A common model for a variety of systems.

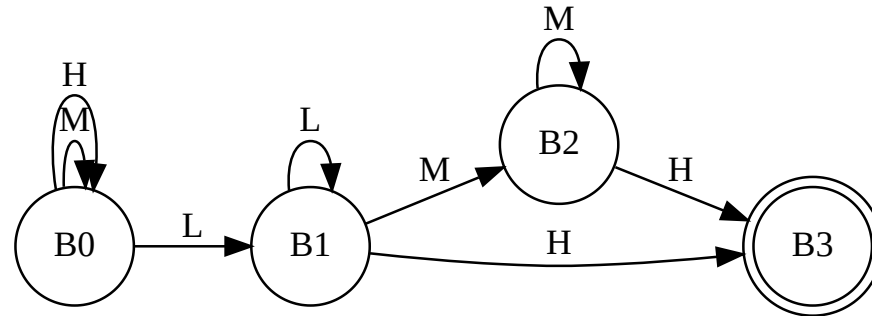control systems [Saifan&Mustafa 2014]

network intrusion detection [Xu et al. 2014]

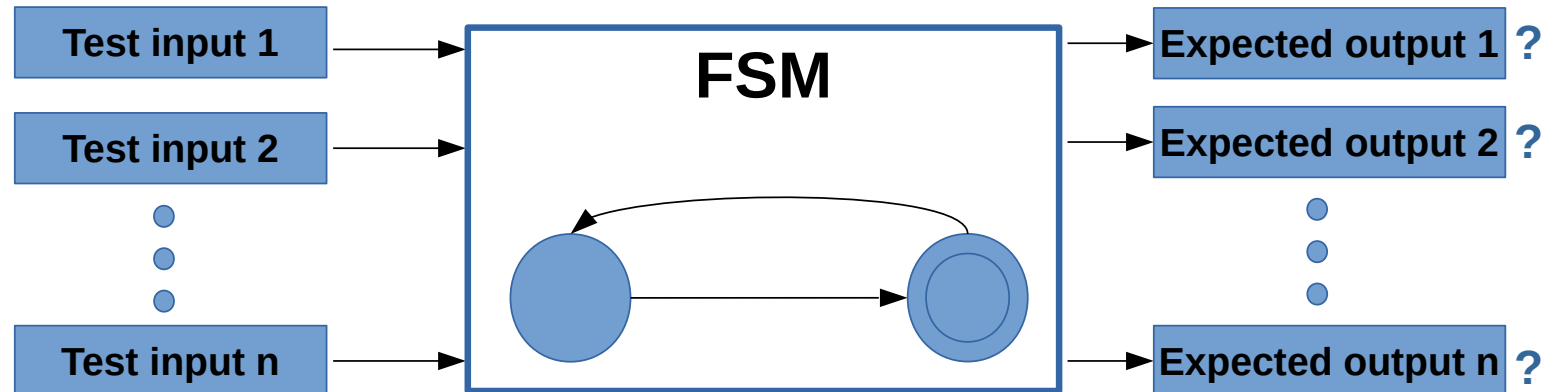signal processing tools [Lehane et al. 2016]

FINITE STATE MACHINES (FSMS)

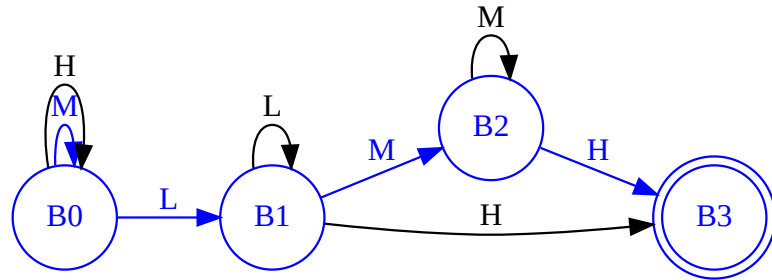Model for a digital oscilloscope

TESTING FSMS

Functional (black-box) testing

1. Generate the test inputs
2. **Execute the tests**
3. Check the results

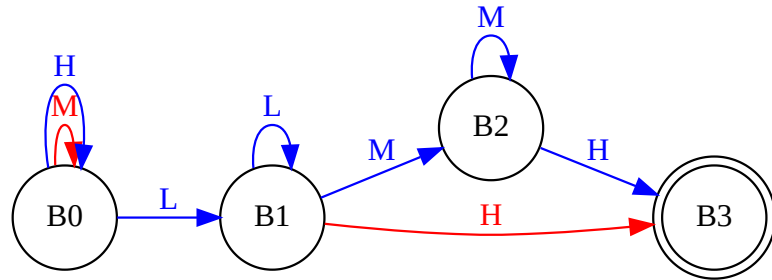# GENERATING TESTS

based on coverage criteria

All-state coverage:

- MLMH -> 0001

✘ **weak** fault-finding [Briand et al. 2004]

GENERATING TESTS

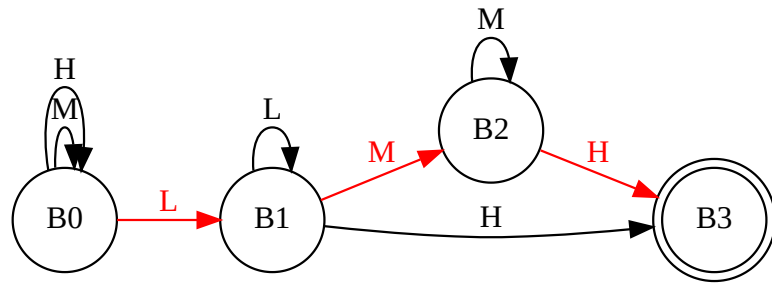*based on coverage criteria*

All-transition coverage:

- HLLMMH -> 000001
- MLH -> 001

✘ **weak** fault-finding [Briand et al. 2004]

GENERATING TESTS

based on coverage criteria

All-transition pair coverage:

- HLLMMH -> 000001
- MLH -> 001
- LMH -> 001

✔ **strong** fault-finding [Briand et al. 2004]

THE PROBLEM

Some finite state machines:

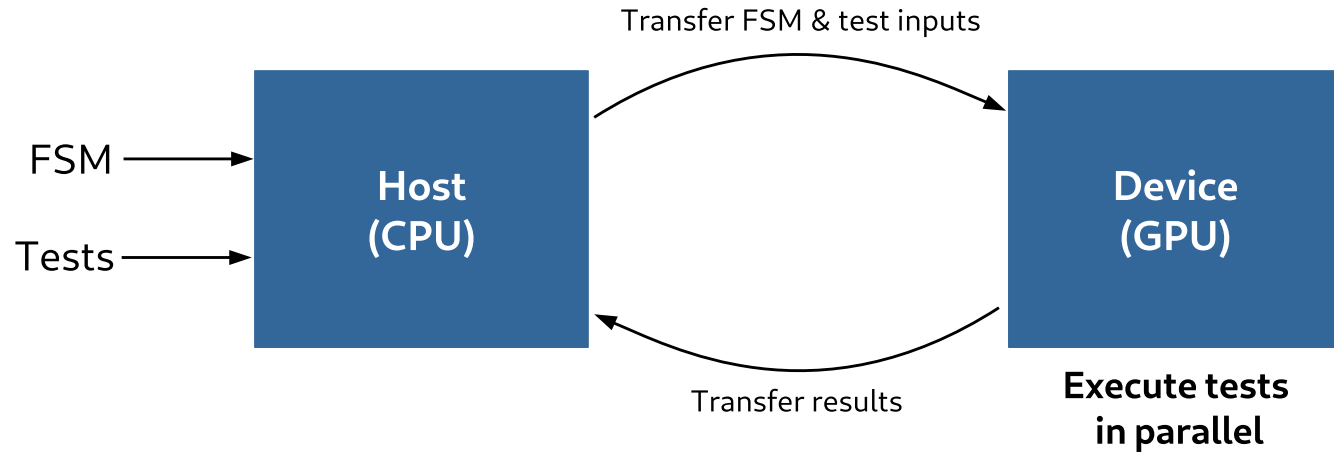**1K**   states and transitions

1M   test sequences

🕐   hours to execute

OUR APPROACH

*Execute the tests **in parallel** on the GPU.*

- Cheap and widely available
- Provide thousands of thread

Transfer FSM & test inputs

FSM →

Tests →

Host (CPU)

Device (GPU)

Transfer results

Execute tests in parallel

one GPU thread = one test

WE HAVE DONE IT BEFORE

*for testing **embedded** C programs.*

ISSTA'17

- Max. speedup **53x** (avg. 16x)
  - on 9 subjects from
    EEMBC embedded benchmark suite
  - when compared to a single CPU

- ***Completely automated*** parallel testing on the GPU
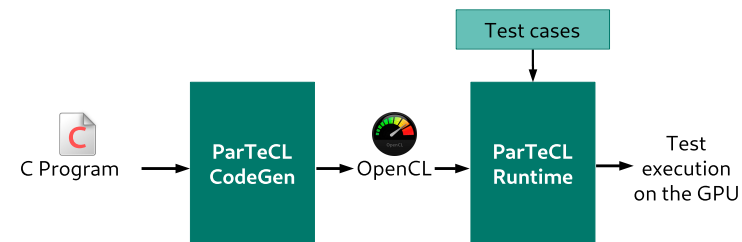
In our current work we *extend* ParTeCL Runtime,
providing a **completely automated** process.

FSMS ARE DIFFERENT TO C PROGRAMS

*FSM execution involves **a lot** of memory accesses.*
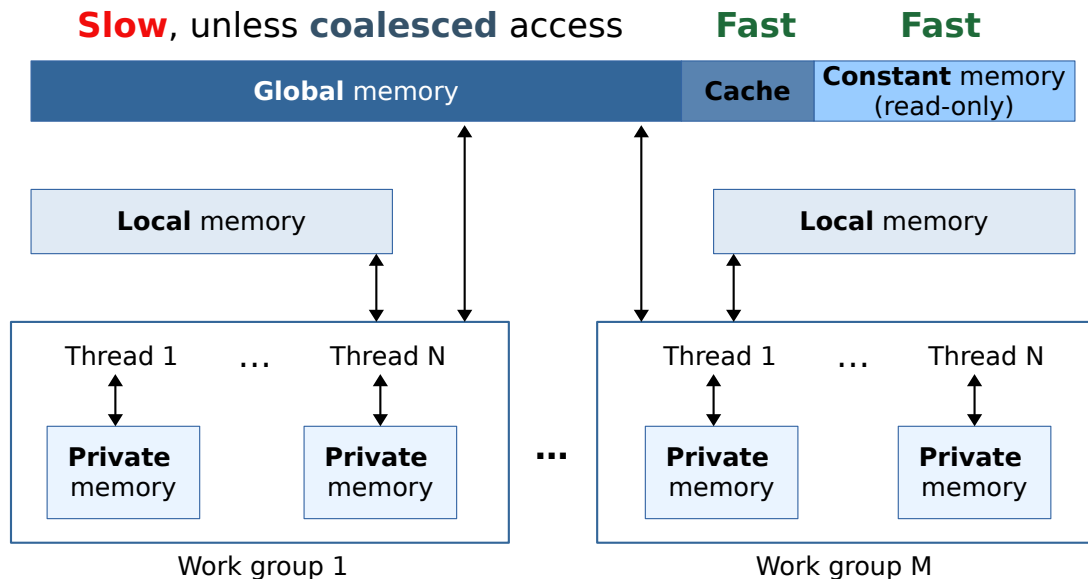
- **FSM** and **tests** are in *global* memory.
- **FSM** is in *constant* memory (if it fits).

To improve performance we need:

compact representation AND/OR **coalesced access**

We investigate **2 FSM** memory layouts and **3 test** memory layouts.

RESEARCH QUESTIONS

- ## RQ1: Test execution speedup
  What is the speedup on the GPU, compared to a 16-core CPU?

- ## RQ2: FSM layout
  Does the choice of FSM layout influence the speedup?

- ## RQ3: Test layout
  Does the choice of test layout influence the speedup?

- ## RQ4: Sorting the tests based on length
  Does sorting the tests influence the speedup?

EXPERIMENT SUBJECTS

| FSM	Domain	#States	#Inputs	#Tests
ssl	intrusion detection (l7-filter)	34	256	1 475 251
battlefield2	intrusion detection (l7-filter)	71	256	1 476 796
dns	intrusion detection (l7-filter)	197	256	8 533 671
aim	intrusion detection (l7-filter)	41	256	1 344 963
rtp	intrusion detection (l7-filter)	28	256	1 536 723
tsp	intrusion detection (l7-filter)	27	256	1 162 511
yahoo	intrusion detection (l7-filter)	54	256	2 627 405
ntp	intrusion detection (l7-filter)	31	256	1 374 296
hotline	intrusion detection (l7-filter)	34	256	1 216 433
h323	intrusion detection (l7-filter)	46	256	2 241 832
halflife2	intrusion detection (l7-filter)	24	256	1 088 409
counterstrike-source	intrusion detection (l7-filter)	30	256	1 472 463
keysight	digital signal processing	4004	3	36 027

Tests generated based on *all-transition pair* criteria.

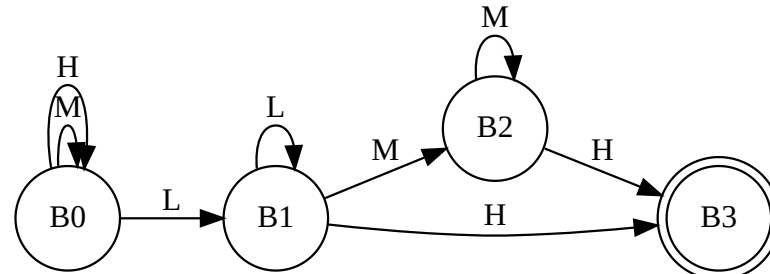# RQ1: TEST EXECUTION SPEEDUP

using the fastest configurations for GPU and 16-core CPU
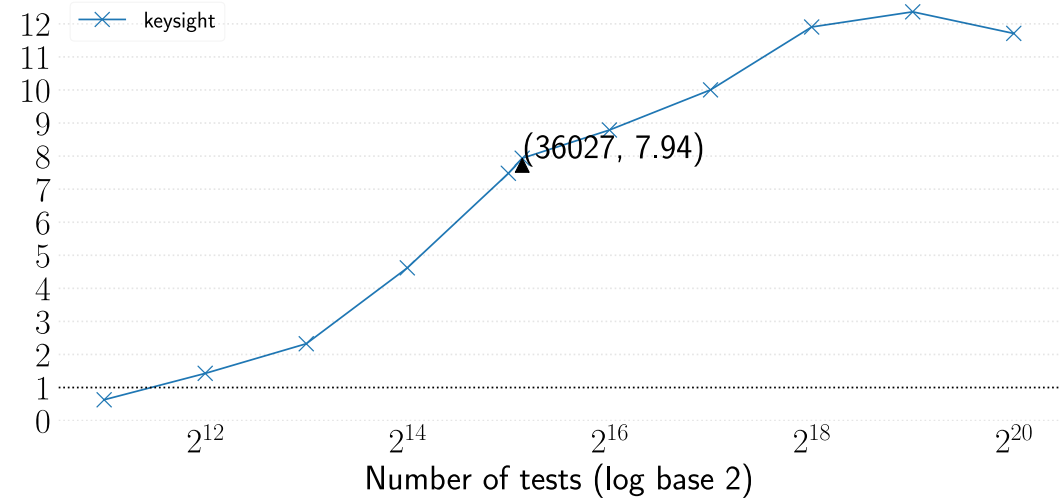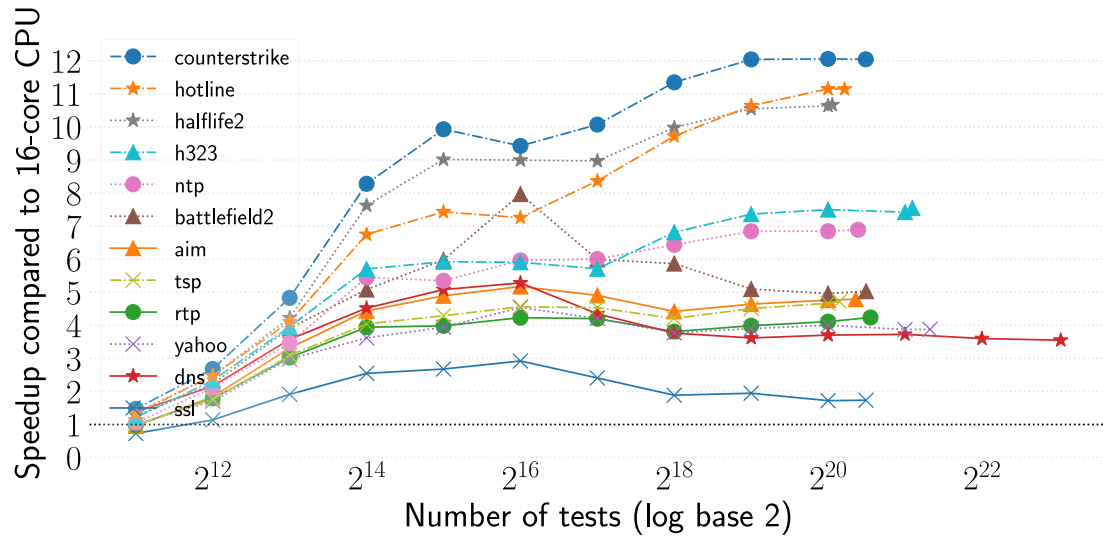
Max. speedup **12x** (avg. 6.4x)

RQ2: FSM LAYOUT IN MEMORY

Sparse
- ✔ compact encoding
- ✘ expensive search for each test input

Dense
- ✔ constant search for each test input
- ✘ padding adds memory overhead

RQ2: FSM LAYOUT IN MEMORY - RESULTS
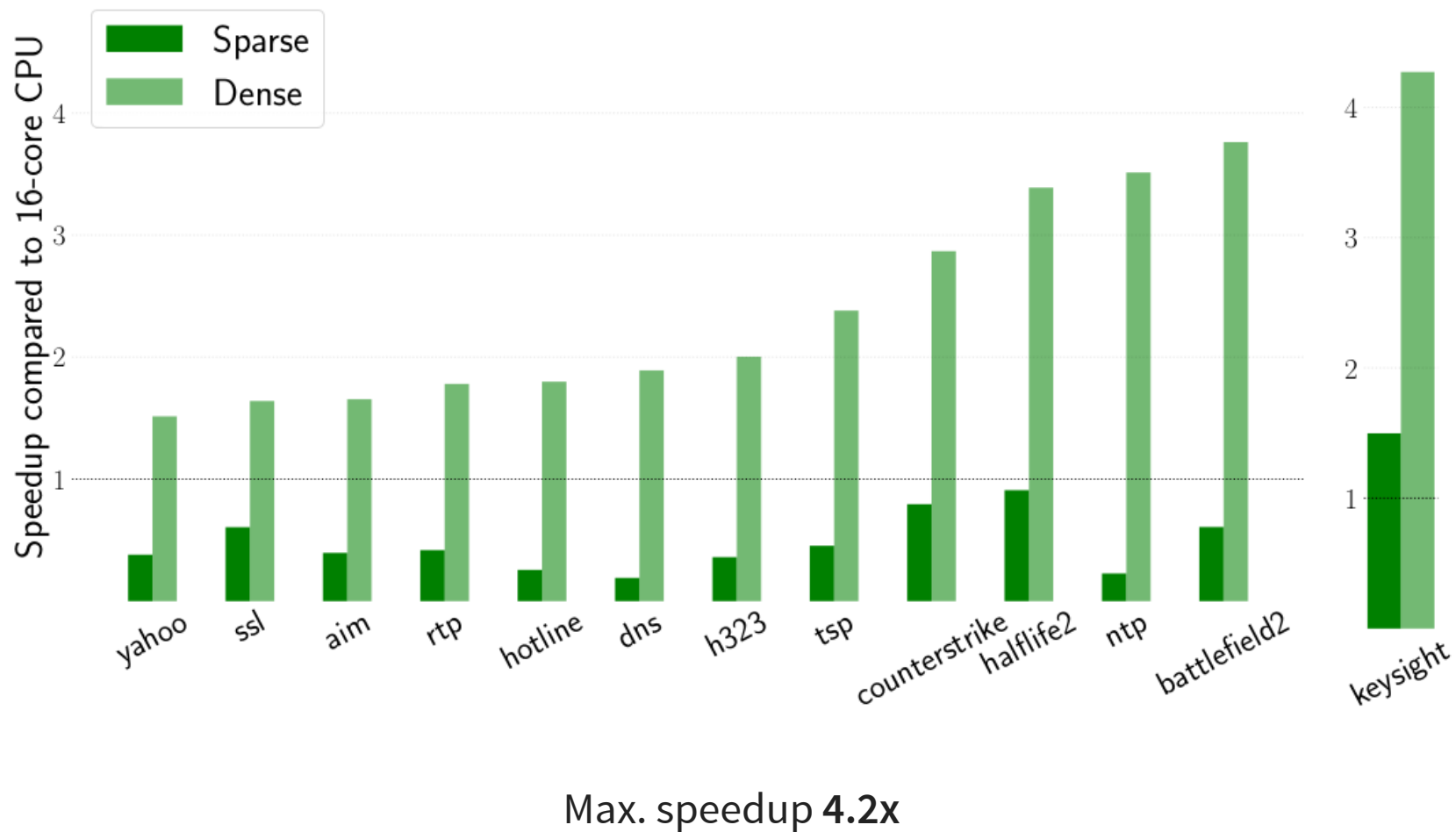
Max. speedup **4.2x**

RQ3: TEST LAYOUT IN MEMORY
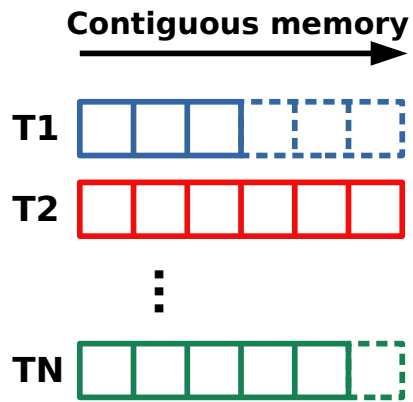
Padded

- ✔ easy to implement
- ✘ padding adds memory overhead
- ✘ no memory coalescing

Padded-transposed

- ✔ memory coalescing

With-offsets

- ✘ no memory coalescing
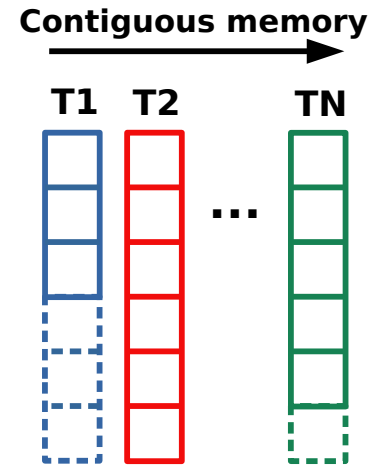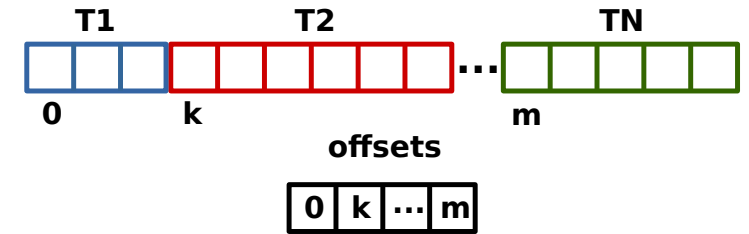- ✔ compact layout

RQ3: TEST LAYOUT IN MEMORY - RESULTS
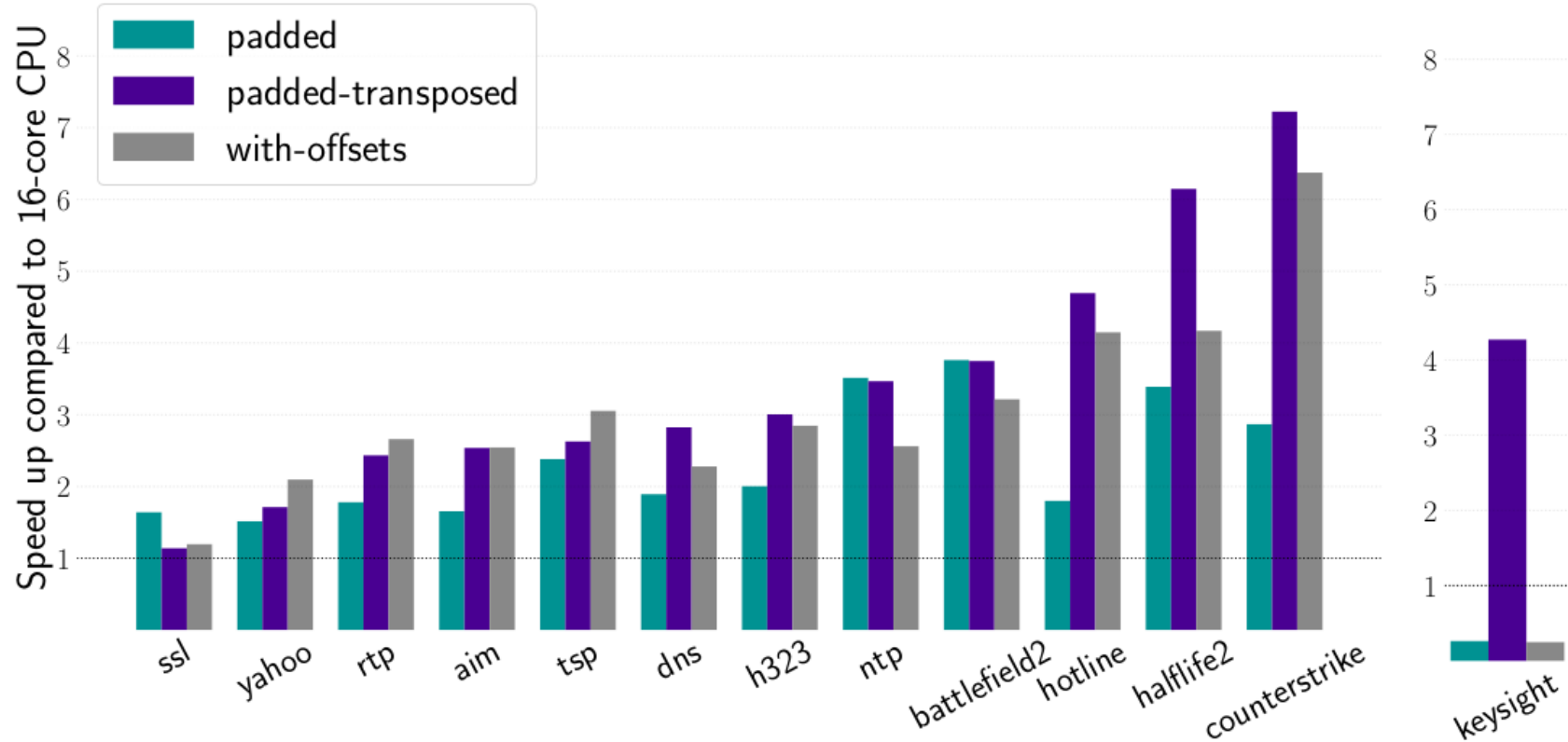
Max. speedup **7.8x**

RQ4: SORTING TESTS BASED ON LENGTH - RESULTS

Max. speedup **12x** (avg. 6x)

SUMMARY

GPUs can accelerate functional testing of FSMs.

Speedup compared to 16-core CPU is **max. 12x** (avg. 6x) with optimised FSM/test layout and load balancing.

We have automated the process.

github.com/wyaneva/partecl-runtime

REFERENCES

1. [Saifan&Mustafa 2014] Using Formal Methods for Test Case Generation According to Transition-Based Coverage Criteria - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Cruise-control-system-finite-state-machine-diagram_fig2_283902282 [accessed 4 Dec, 2018]

2. [Xu et al.2014] Xu, Yang et al. TFA: A Tunable Finite Automaton for Pattern Matching in Network Intrusion Detection Systems. IEEE Journal on Selected Areas in Communications 32 (2014): 1810-1821.

3. [Lehane et al. 2016] Lehane et al. Digital Triggering Using Finite State Machines. US Patent App. 14/957,491. March 2016

4. [Briand et al. 2004] Briand et al. Using Simulation to Empirically investigate Test Coverage Criteria Based on Statechart. *In ICSE 2004*