

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

Published in APSEC'18



THE UNIVERSITY of EDINBURGH
informatics



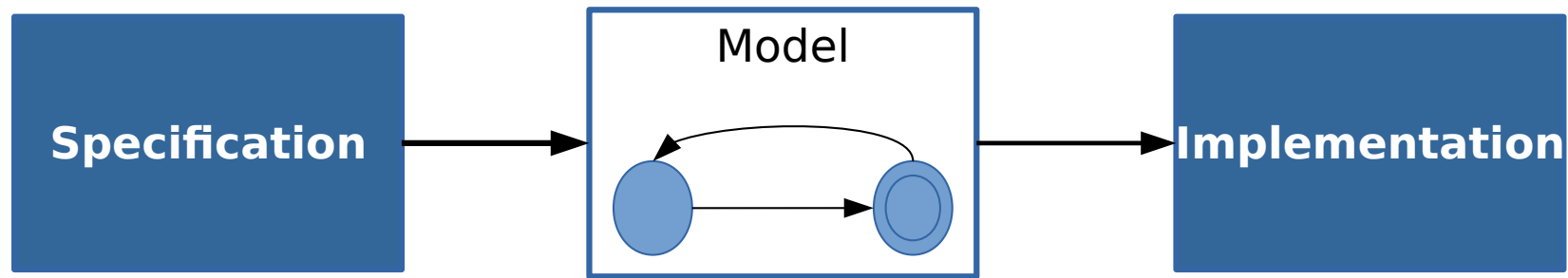
INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

HYDERABAD

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

MODEL-BASED DEVELOPMENT

Software is implemented and tested based on a model.



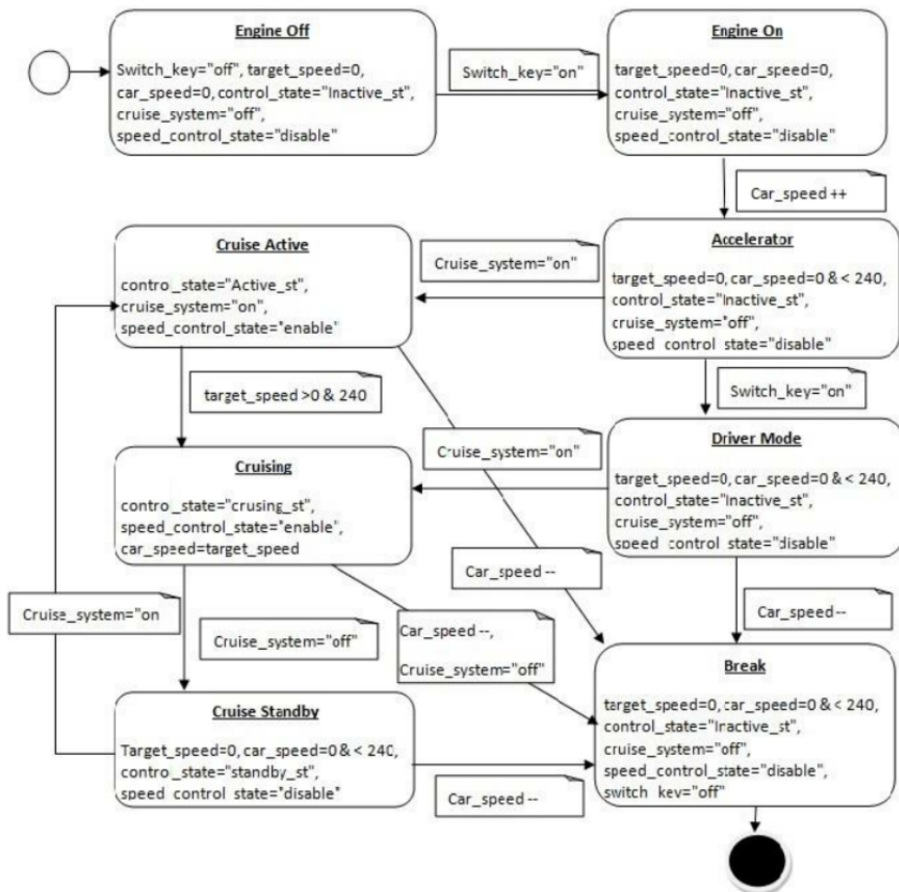
model = specification ?
Execute tests on the model

There are many ways to define a model:

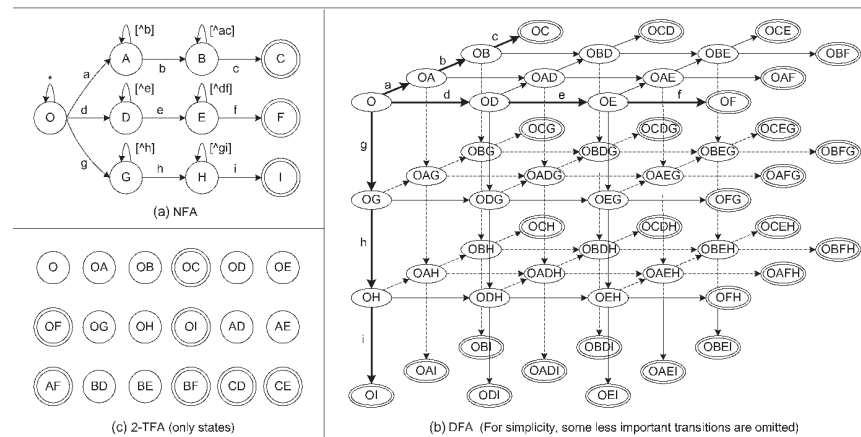
- UML
- formal specification languages (Z, B, Alloy)
- block/state charts

FINITE STATE MACHINES (FSMS)

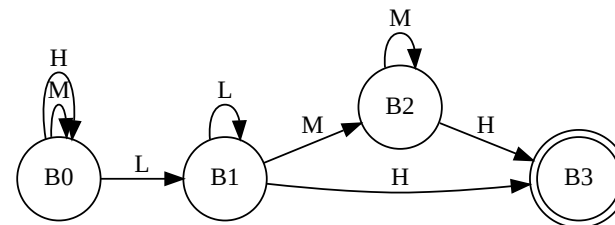
A common model for a variety of systems.



control systems [Saifan&Mustafa 2014]

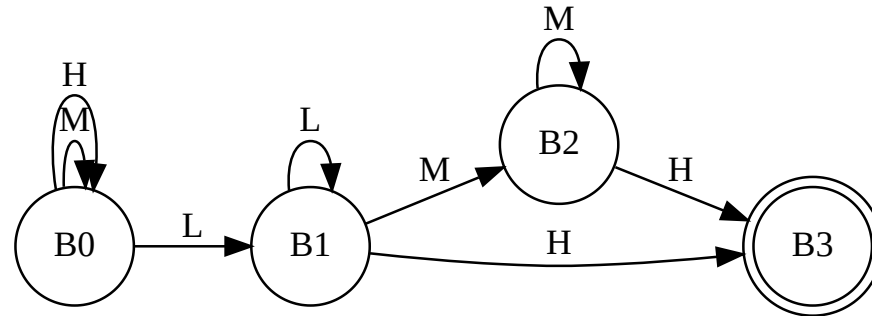


network intrusion detection [Xu et al. 2014]



signal processing tools [Lehane et al. 2016]

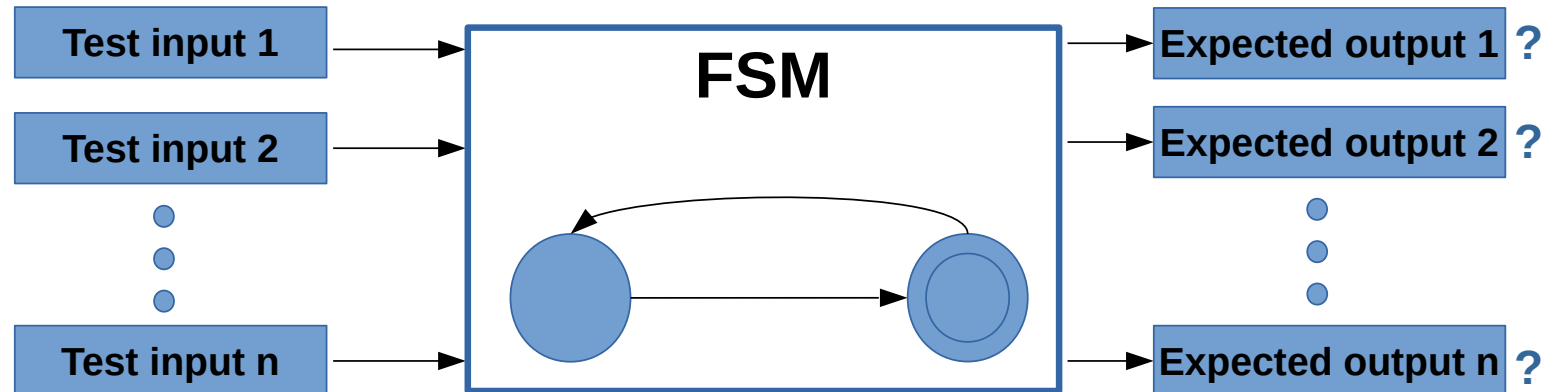
FINITE STATE MACHINES (FSMS)



Model for a digital oscilloscope  **KEYSIGHT**
TECHNOLOGIES

TESTING FSMS

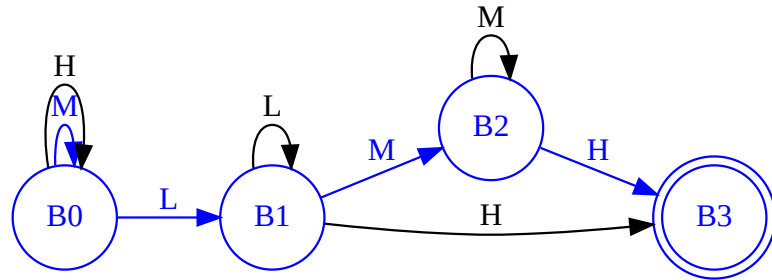
Functional (black-box) testing



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

GENERATING TESTS

based on coverage criteria



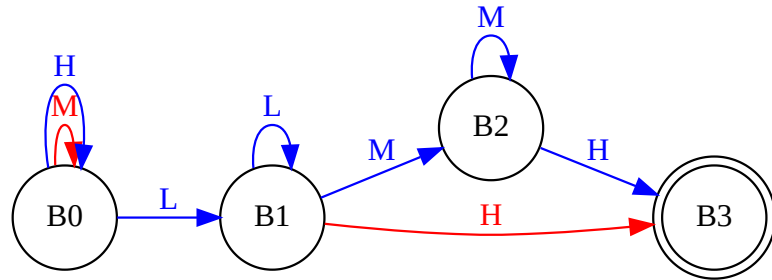
All-state coverage:

- MLMH -> 0001

X weak fault-finding [Briand et al. 2004]

GENERATING TESTS

based on coverage criteria



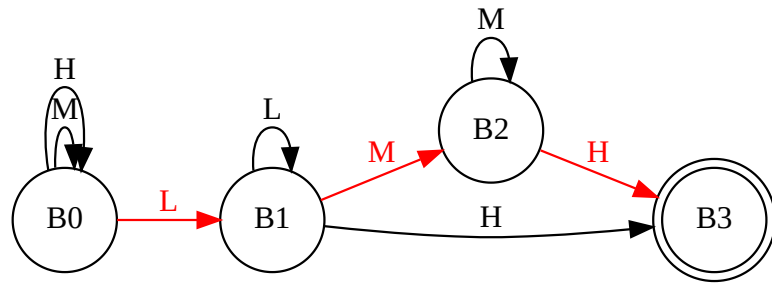
All-transition coverage:

- 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:



states and transitions



test sequences

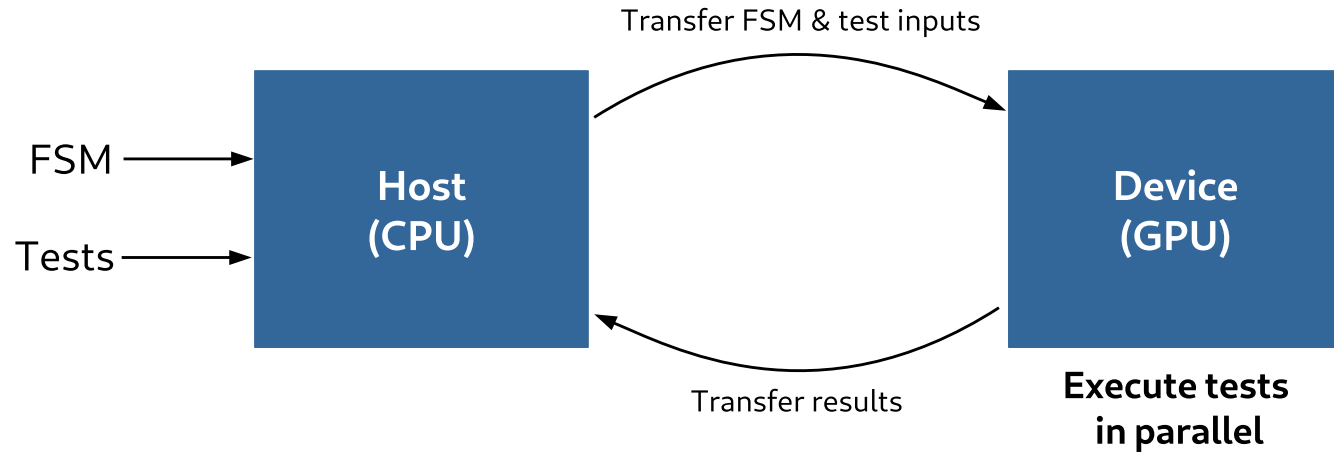


hours to execute

OUR APPROACH

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

- Cheap and widely available
- Provide thousands of thread



one GPU thread = one test

WE HAVE DONE IT BEFORE

for testing *embedded C programs*.

Compiler-Assisted Test Acceleration on GPUs for Embedded Software

Vanya Yaneva
School of Informatics
University of Edinburgh, UK
vanya.yaneva@ed.ac.uk

Ajitha Rajan
School of Informatics
University of Edinburgh, UK
arajan@staffmail.ed.ac.uk

Christophe Dubach
School of Informatics
University of Edinburgh, UK
christophe.dubach@ed.ac.uk

ABSTRACT

Embedded software is found everywhere from our highly visible mobile devices to the confines of our car in the form of smart

sensors. Embedded software is critical to all safety-critical systems, and its development requires using increasingly powerful hardware. Speeding up embedded software testing by running it on GPUs. However, GPU maintenance requires developers have a deep understanding of GPU architecture.

We propose a framework for running embedded software on GPUs, making it available in the form of a library. In this paper, we describe the design and implementation of this framework, and show how it can be used to accelerate the execution of embedded software tests on GPUs. We show that this approach can achieve a speedup of up to 16x when compared to a single CPU for these benchmarks.

CCS CONCEPTS
• Software and its engineering
• Software testing and debugging
• Software testing and debugging
• Software testing and debugging

KEYWORDS
Functional testing
Automated testing

ACM Reference format:

Vanya Yaneva, Ajitha Rajan, and Christophe Dubach. 2017. Compiler-Assisted Test Acceleration on GPUs for Embedded Software. In *Proceedings of 26th International Symposium on Software Testing and Analysis*. Santa Barbara, CA, USA: ACM Press.

ParTeCL: Parallel Testing using OpenCL*

Vanya Yaneva
University of Edinburgh, UK
vanya.yaneva@ed.ac.uk

Ajitha Rajan
University of Edinburgh, UK
arajan@staffmail.ed.ac.uk

Christophe Dubach
University of Edinburgh, UK
christophe.dubach@ed.ac.uk

ABSTRACT

With the growing complexity of software, the number of test cases needed for effective validation is extremely large. Executing these large test suites is expensive and time consuming, putting an enormous pressure on the software development cycle. In previous work, we proposed using Graphics Processing Units (GPUs) to accelerate test execution by running test cases in parallel on the GPU threads. However, the complexity of GPU programming poses challenges to the usability and effectiveness of the proposed approach.

In this paper we present ParTeCL - a compiler-assisted framework to automatically generate GPU code from sequential programs and execute their tests in parallel on the GPU. We show feasibility and performance achieved when executing test suites for 9 programs from an industry standard benchmark suite on the GPU. ParTeCL achieves an average speedup of 16x when compared to a single CPU for these benchmarks.

CCS CONCEPTS

• Software and its engineering → Software testing and debugging; Source code generation; • Computer systems organization

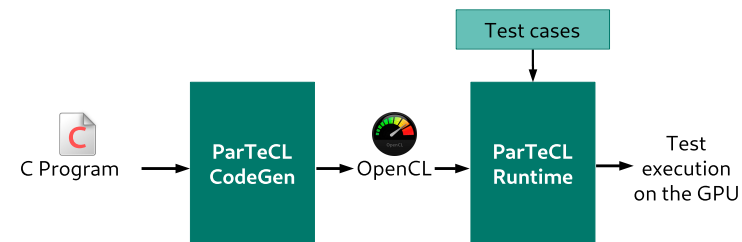
them concurrently to reduce execution time of the entire test suite. This approach, however, is costly in terms of resources, infrastructure, maintenance and energy consumed. Present day commodity parallel accelerators, such as Graphics Processing Units (GPUs), offer enormous computing power while also being cheap, easily available and energy efficient. A single GPU offers thousands of parallel threads with the potential to execute a large number of test cases concurrently. However, GPUs are notoriously hard to program and require significant expertise and a thorough understanding of the hardware and programming model to unlock their potential. We plan to address these problems in the context of test execution using our ParTeCL framework. ParTeCL has the following goals.

- (1) Increase the *usability and feasibility* of GPUs for test execution.
- (2) Increase the *performance and effectiveness* with compiler optimisations that analyse the tests and the program.

Our recently accepted paper [10] in the main research track of ISSTA 2017 presents empirical evaluations of our approach and discusses the performance and effectiveness of using

- 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

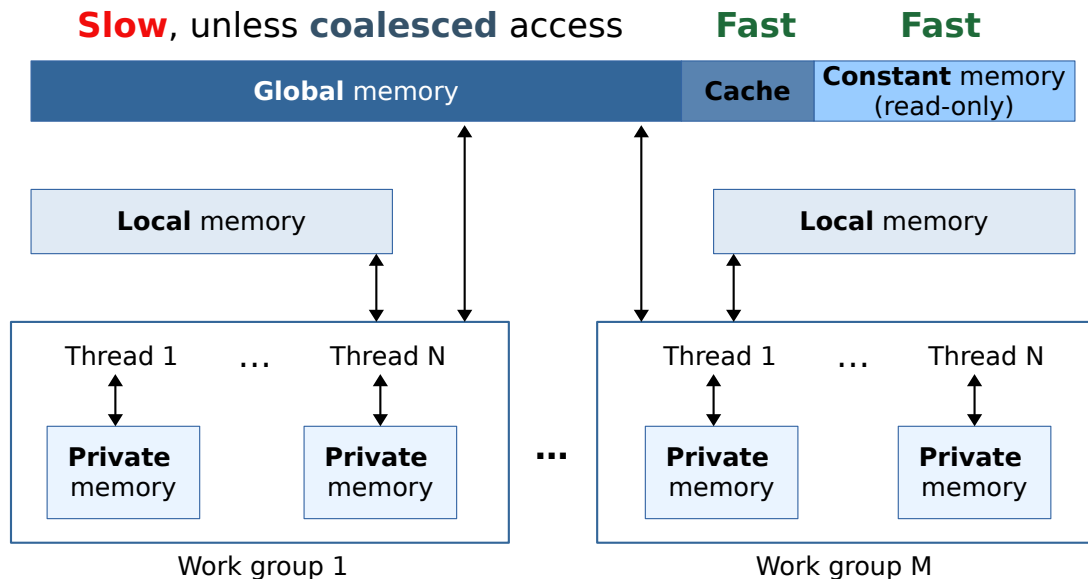


ISSTA'17

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?

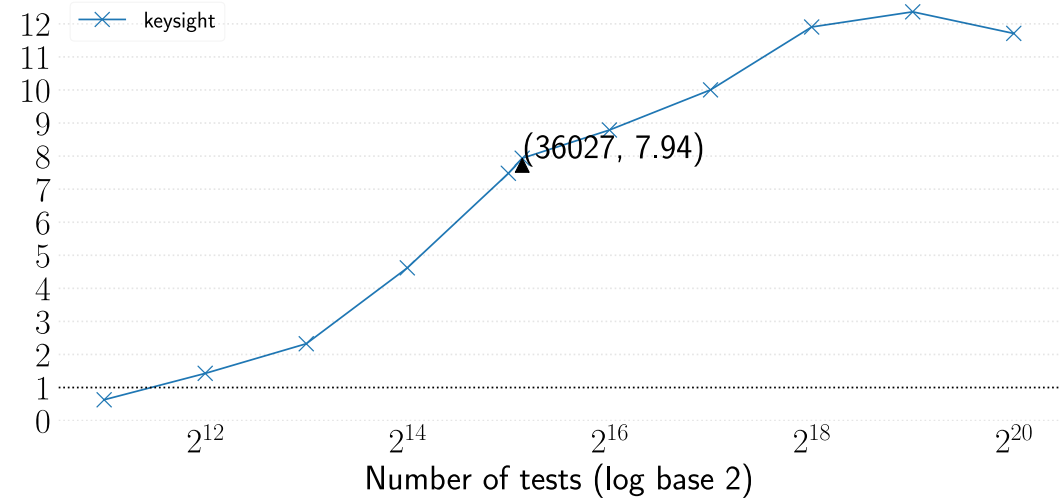
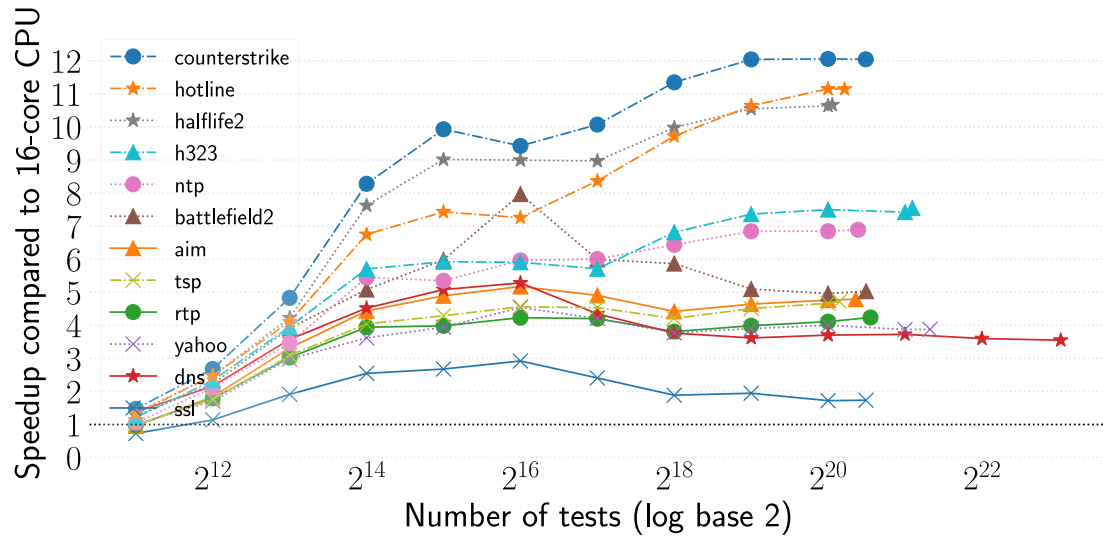
EXPERIMENT SUBJECTS

FSM	Domain	#States	#Inputs	#Tests
ssl	intrusion detection (17-filter)	34	256	1 475 251
battlefield2	intrusion detection (17-filter)	71	256	1 476 796
dns	intrusion detection (17-filter)	197	256	8 533 671
aim	intrusion detection (17-filter)	41	256	1 344 963
rtp	intrusion detection (17-filter)	28	256	1 536 723
tsp	intrusion detection (17-filter)	27	256	1 162 511
yahoo	intrusion detection (17-filter)	54	256	2 627 405
ntp	intrusion detection (17-filter)	31	256	1 374 296
hotline	intrusion detection (17-filter)	34	256	1 216 433
h323	intrusion detection (17-filter)	46	256	2 241 832
halflife2	intrusion detection (17-filter)	24	256	1 088 409
counterstrike-source	intrusion detection (17-filter)	30	256	1 472 463
keysight	digital signal processing	4004	3	36 027

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

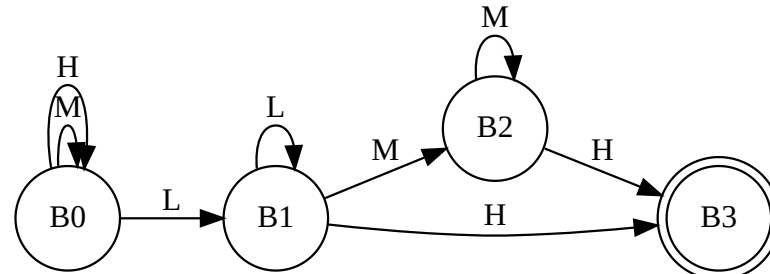
RQ1: TEST EXECUTION SPEEDUP

using the fastest configurations for GPU and 16-core CPU



Max. speedup **12x** (avg. 6.4x)

RQ2: FSM LAYOUT IN MEMORY



B0	L, B1, 0	M, B0, 0	H, B0, 0
B1	L, B1, 0	M, B2, 0	H, B3, 1
B2	M, B2, 0	H, B3, 1	

Sparse

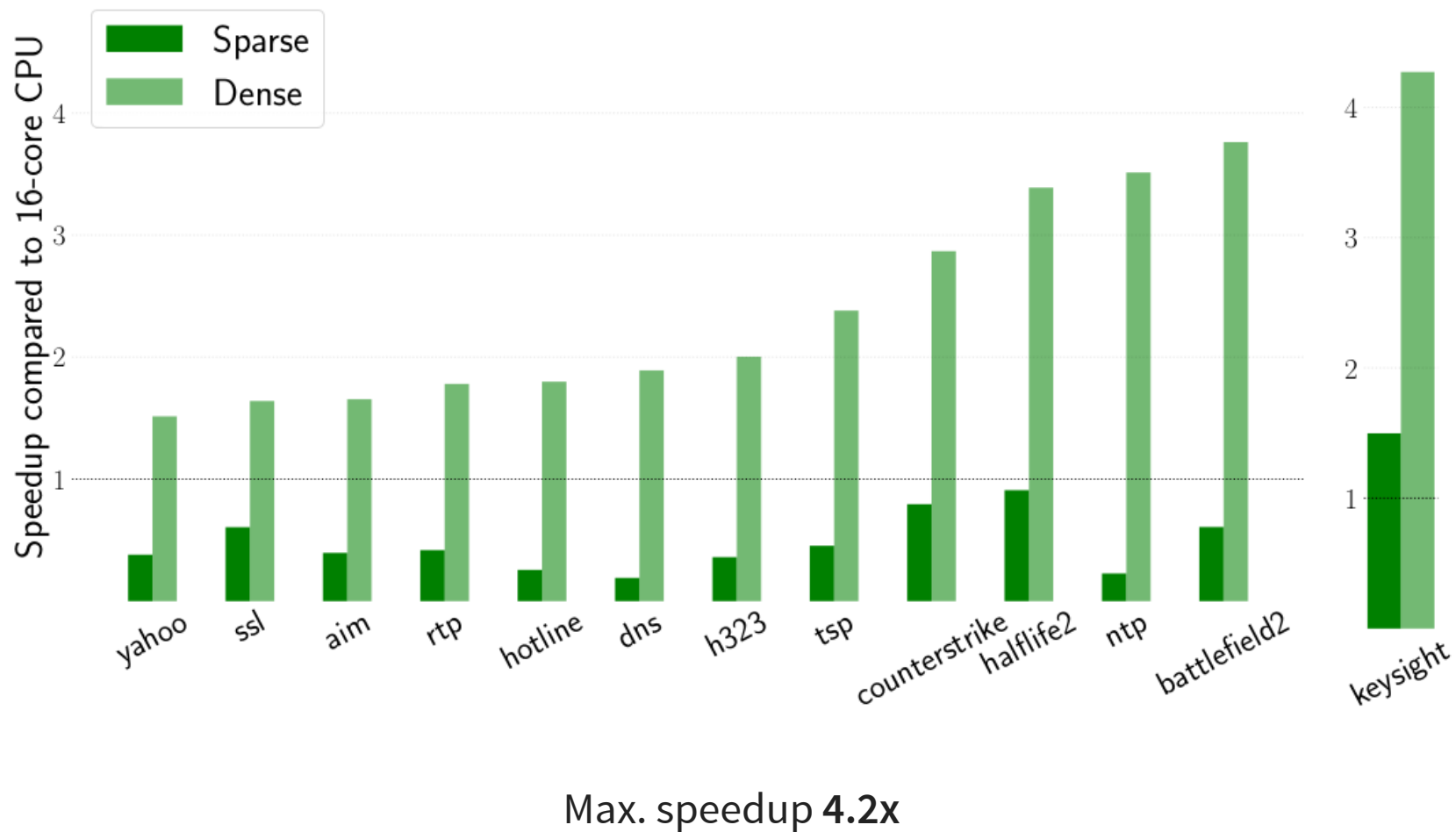
- ✓ compact encoding
- ✗ **expensive** search for each test input

	L	M	H
B0	B1, 0	B0, 0	B0, 0
B1	B1, 0	B2, 0	B3, 1
B2	padding	B2, 0	B3, 1
B3	padding	padding	padding

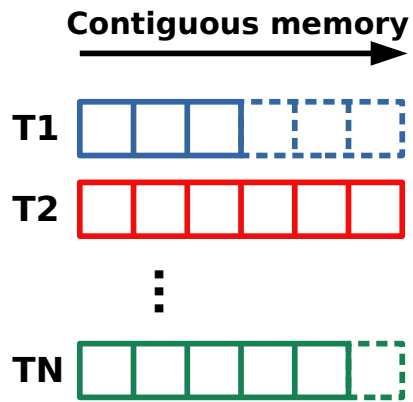
Dense

- ✓ **constant** search for each test input
- ✗ padding adds memory overhead

RQ2: FSM LAYOUT IN MEMORY - RESULTS

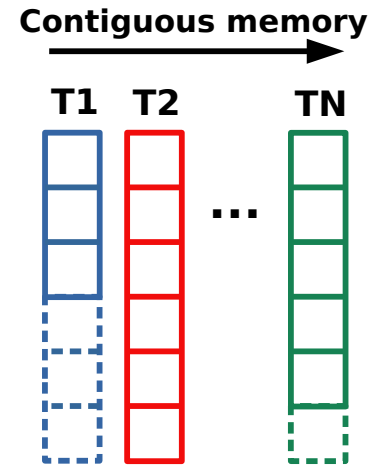


RQ3: TEST LAYOUT IN MEMORY



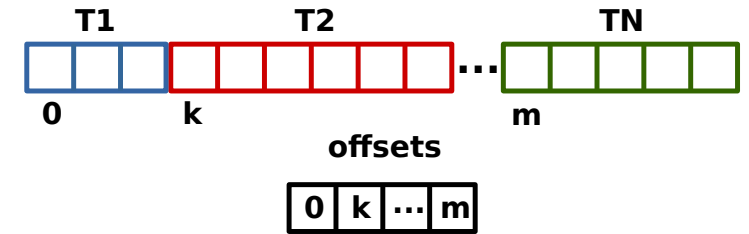
Padded

- ✓ easy to implement
- ✗ padding adds memory overhead
- ✗ no memory coalescing



Padded-transposed

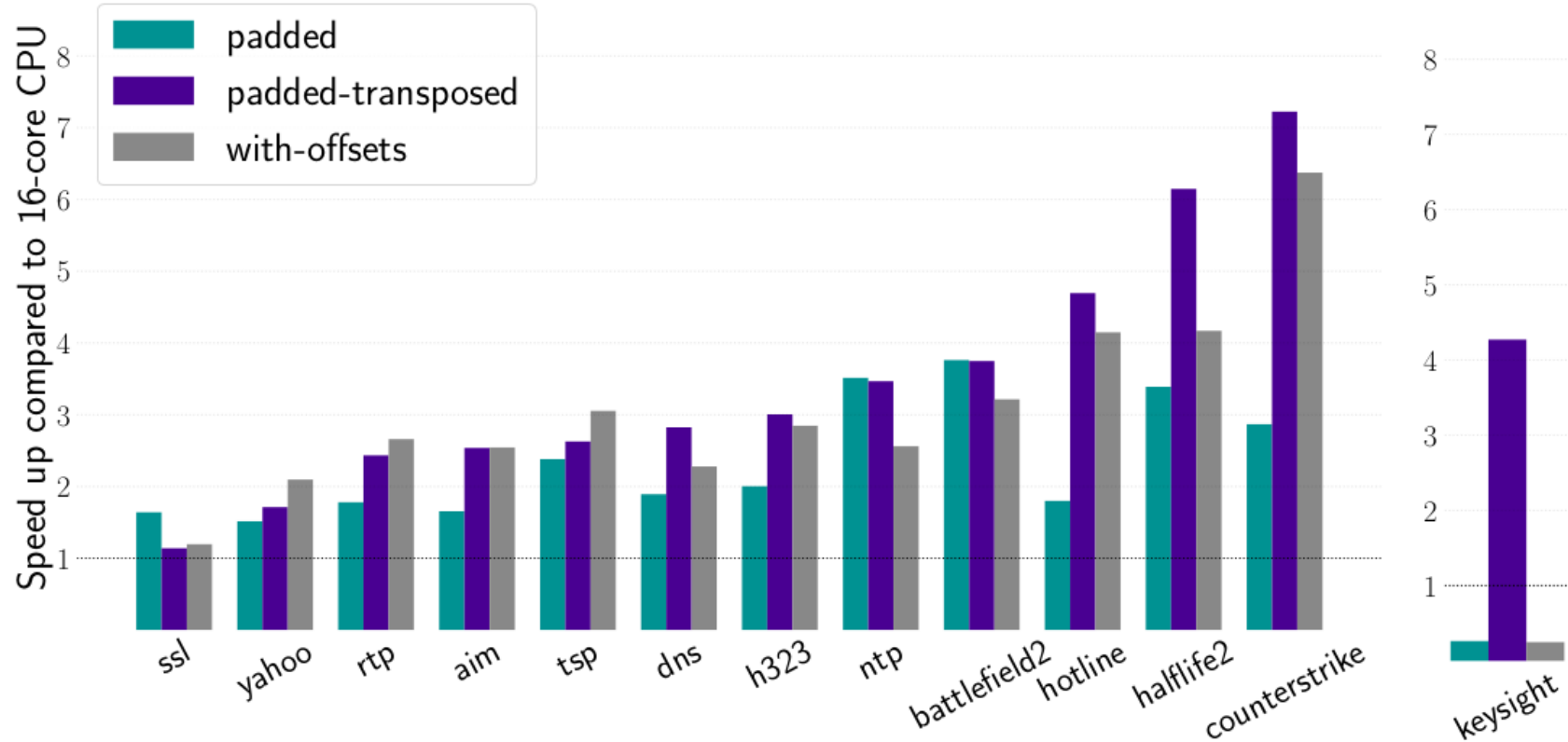
- ✓ memory coalescing



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/partectl-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*