Micro-Architectural Attacks on Cyber-Physical Systems

07/09/2019

Heechul Yun

Associate Professor*, EECS

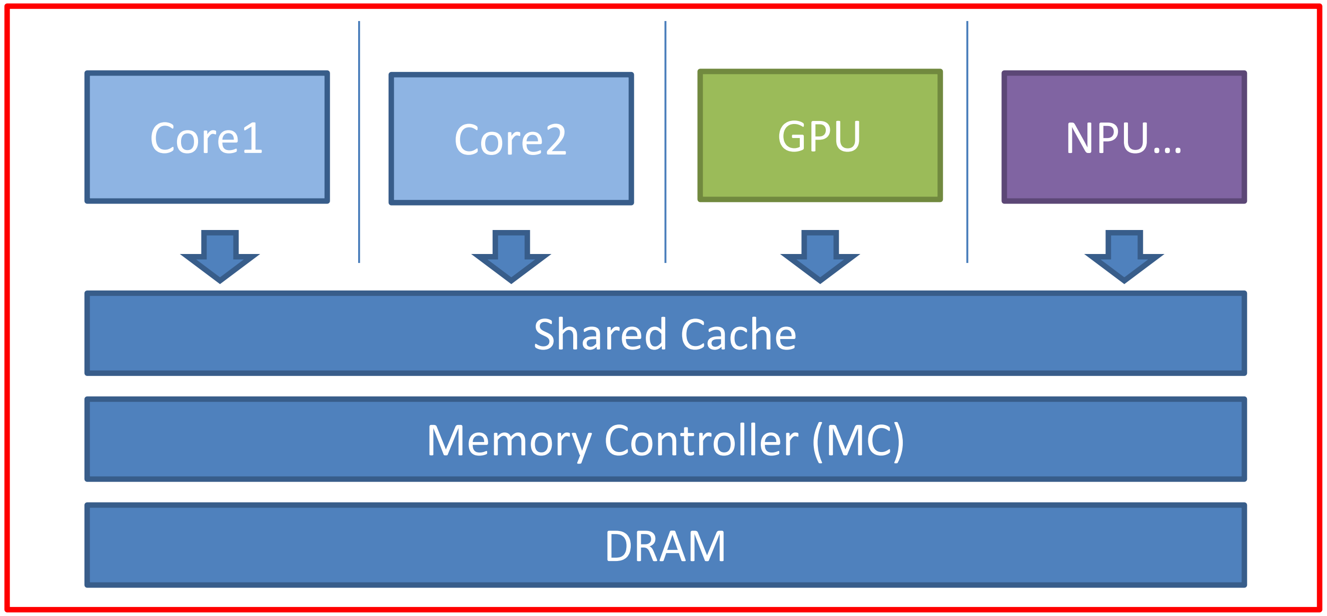
University of Kansas

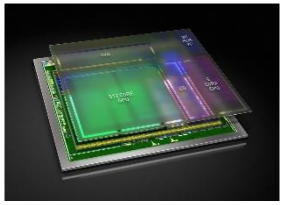
* Effective in Fall, 2019

Modern Cyber-Physical Systems

- Cyber Physical Systems (CPS)
  - Cyber (Computer) + Physical (Plant)
- **Real-time**
  - Control physical process in real-time
- **Safety-critical**
  - Can harm people/things
- **Intelligent**
  - Can function autonomously

Modern System-on-a-Chip (SoC)

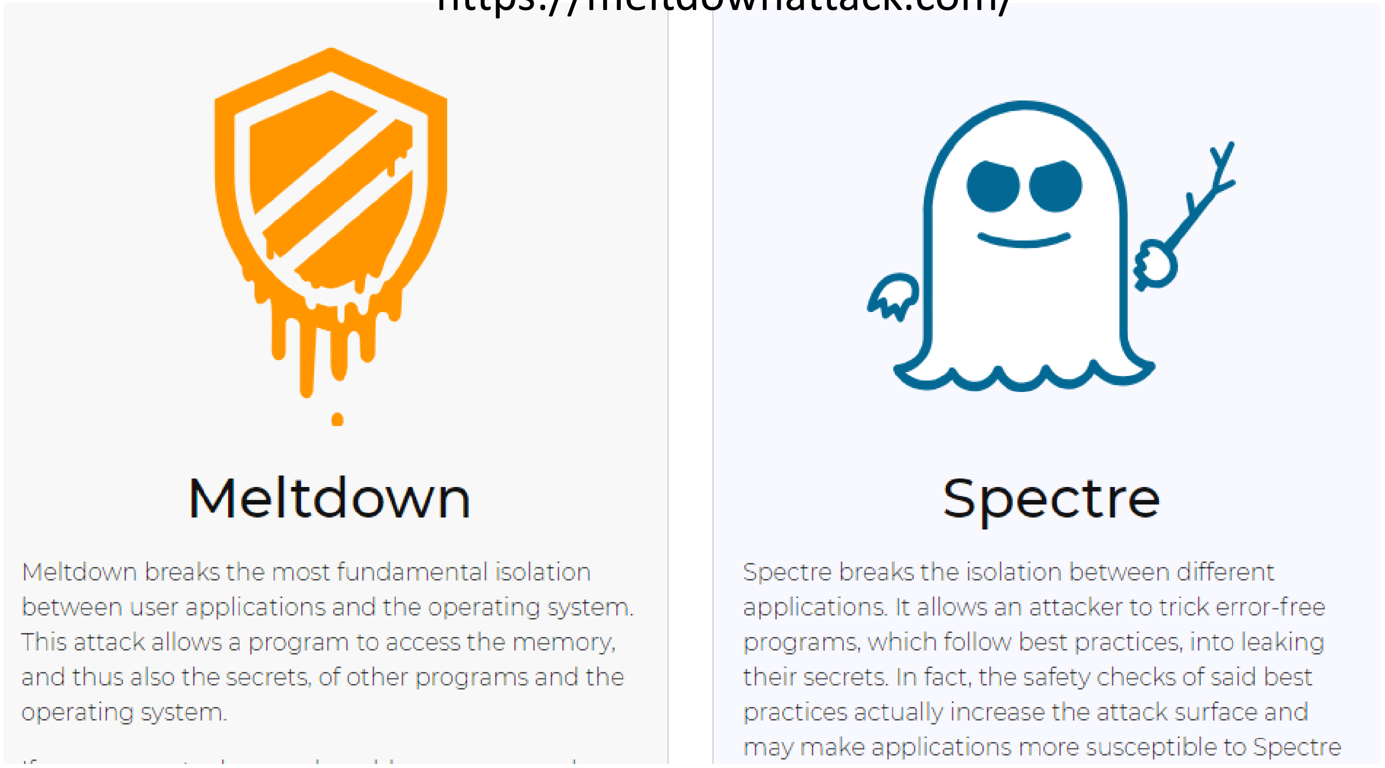




- Integrate multiple cores, GPU, accelerators
- Good performance, size, weight, power
- Challenges: safety, security

Micro-Architectural Attacks

https://meltdownattack.com/



Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

- Software attacks on hardware are difficult to defend
- Complex hardware → **many attack vectors**

Micro-Architectural Attacks

- Micro-architectural hardware components
  - E.g., cache, tlb, DRAM, OoO engine, …
- Can leak secret
  - E.g., Meltdown, Spectre
- Can alter the content of the stored data
  - E.g., RowHammer
- Can affect execution timing
  - E.g., DoS attack on real-time tasks
- **Logically correct software is also vulnerable**

Project Goal

- Develop micro-architectural attack resistant computing infrastructure for secure cyber-physical systems (CPS)

Results So Far…

1. Jacob Michael Fustos, Farzad Farshchi, and Heechul Yun. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. *Design Automation Conference (**DAC**)*, 2019.
2. Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2019 **Outstanding Paper Award**
3. Waqar Ali and Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2019.
4. Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (**EMC^2**)*, 2019.

SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

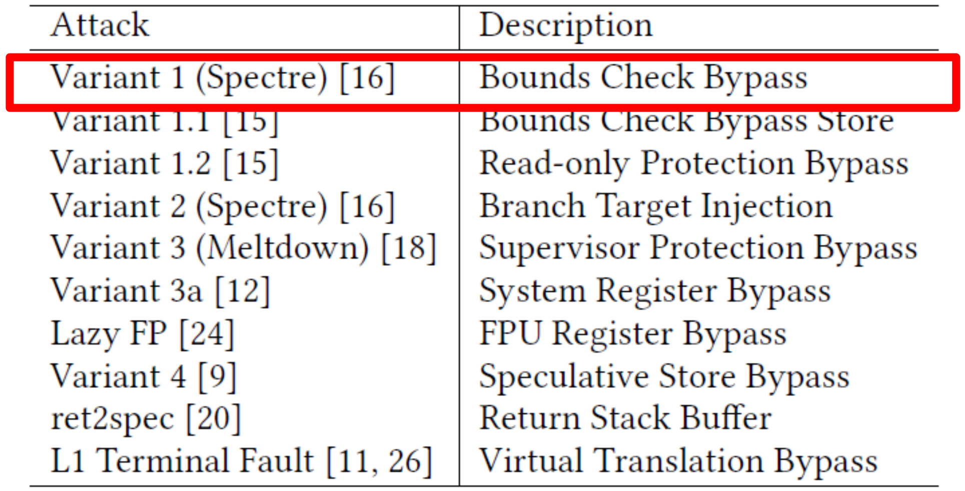
Jacob Fustos, Farzad Farshchi, and Heechul Yun

*ACM/IEEE Design Automation Conference (**DAC**)*

Las Vegas, Nevada, June, 2019.

Speculative Execution Attacks

- Attacks exploiting microarchitectural side-effects of executing speculative (transient) instructions
- Many variants



Attack	Description
Variant 1 (Spectre) [16]	Bounds Check Bypass
Variant 1.1 [15]	Bounds Check Bypass Store
Variant 1.2 [15]	Read-only Protection Bypass
Variant 2 (Spectre) [16]	Branch Target Injection
Variant 3 (Meltdown) [18]	Supervisor Protection Bypass
Variant 3a [12]	System Register Bypass
Lazy FP [24]	FPU Register Bypass
Variant 4 [9]	Speculative Store Bypass
ret2spec [20]	Return Stack Buffer
L1 Terminal Fault [11, 26]	Virtual Translation Bypass

No hardware support planned in near future

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
........
```

- Assume `x` is under the attacker's control
- Attacker trains the branch predictor to predict the branch is in-bound

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
........
```

1. [ACCESS]

- Speculative execution of the first line **accesses** the secret (`val`)

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
........
```

2. [TRANSMIT]

- Speculative execution of the second, secret dependent load **transmits** the secret to a *microarchitectural state (e.g., cache)*

KU THE UNIVERSITY OF KANSAS

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

3. [RECEIVE]

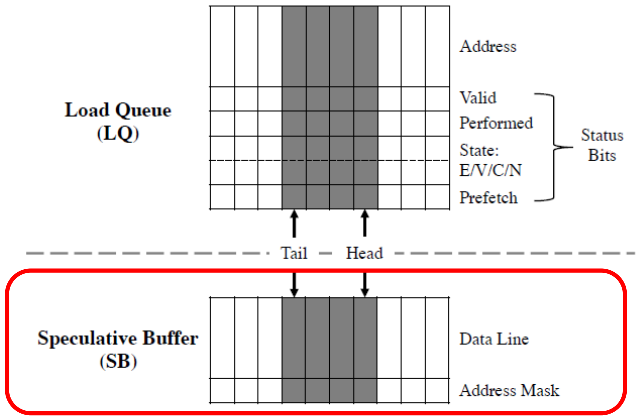
- Attacker **receives** the secret by timing access latency differences (cache hit vs. miss) among the elements in the probe array
  - Flush+reload, prime+probe, …

Existing Software Mitigation

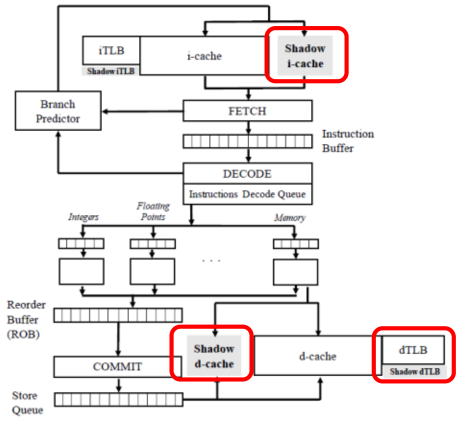
```
if(x < array1_length){
      _mm_lfence();
    val = array1[x];
    tmp = array2[val*512];
}
```

- Manually stop speculation
  - By inserting '`lfence`' instructions [Intel, 2018]
  - Or by introducing additional data dependencies [Carruth, 2018]
  - Error prone, high programming complexity, performance overhead

Existing Hardware Mitigation



InvisiSpec [Yan et al., MICRO'18]



SafeSpec [Khasawneh et al., DAC'19]

- Hide speculative execution
  – By buffering speculative results into additional "*shadow*" hardware structures
  – High complexity, high overhead (performance, space)

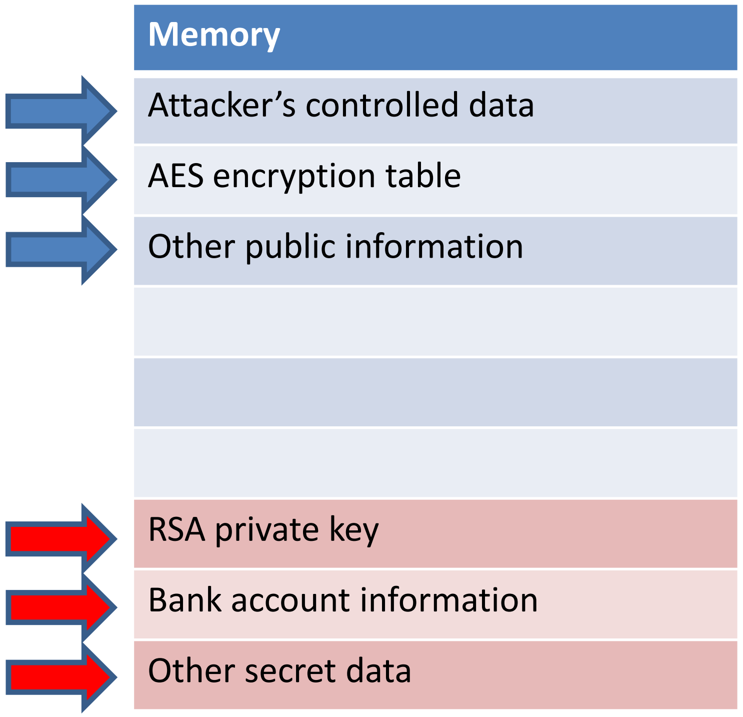
SpectreGuard

- Data-centric software/hardware collaborative approach
  - Software tells hardware what **data** (not code) needs protection
  - Hardware selectively protects the identified **data** from Spectre attacks

- Key observations
  - Not all data is secret
  - Not all speculative loads leak secret

Obs. 1: Not All Data Is Secret

- Non-sensitive data
  - Most program code, data
  - **Optimize for performance**
- Sensitive (secret) data
  - Cryptographic keys, passwords, …
  - **Optimize for security**



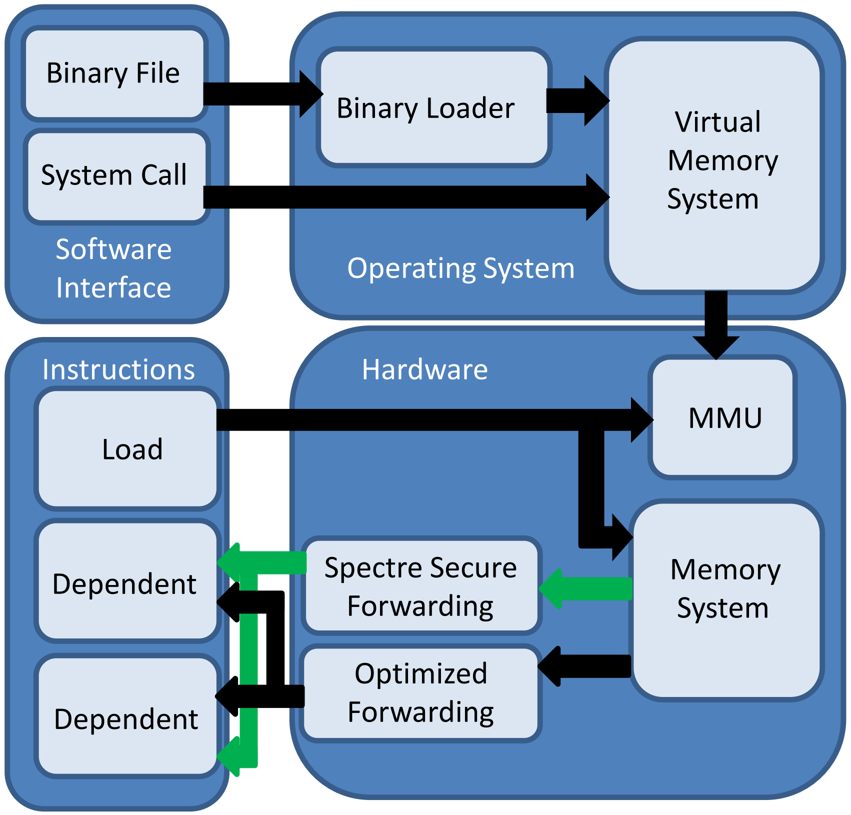
Obs. 2: Not All Speculative Loads Leak Secret

```
if(x < array1 length){
      val = array1[x];            1. [ACCESS]
      tmp = array2[val*512];      2. [TRANSMIT]
}
........
```

- The first load does **NOT** leak secret
- The second, **secret dependent load** leaks the secret
- Delay the secret dependent load until *after* the branch is resolved

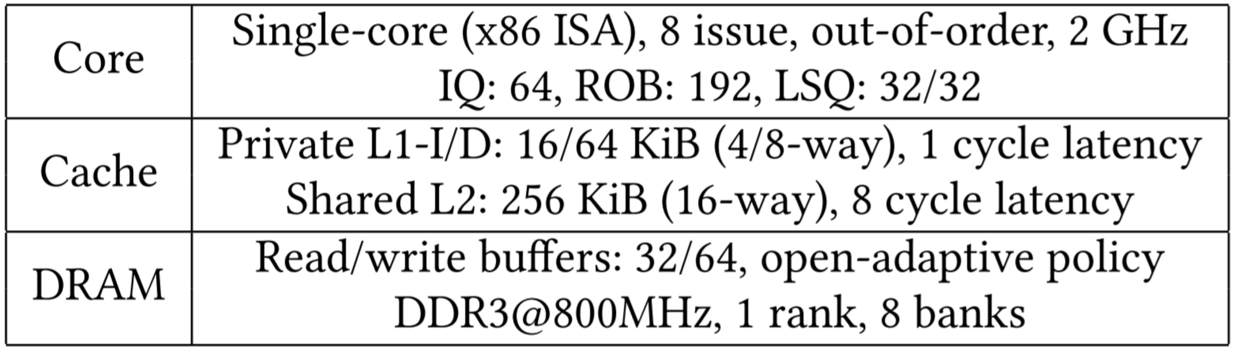
SpectreGuard Approach

- Step 1: Software tells OS what data is secret
- Step 2: OS updates the *page table* entries
- Step 3: Load of the secret data is identified by MMU
- Step 4: secret data forwarding is **delayed** until safe



Evaluation Setup

- Full system simulation using Gem5 (O3CPU model) and Linux kernel (4.18)



Core	Single-core (x86 ISA), 8 issue, out-of-order, 2 GHz IQ: 64, ROB: 192, LSQ: 32/32
Cache	Private L1-I/D: 16/64 KiB (4/8-way), 1 cycle latency Shared L2: 256 KiB (16-way), 8 cycle latency
DRAM	Read/write buffers: 32/64, open-adaptive policy DDR3@800MHz, 1 rank, 8 banks

- Comparison
  - *Native*: unmodified baseline system
  - *InvisiSpec*: a fully hardware solution [Yan et al., Micro'18]
  - *Fence*: a fully software solution (insert `lfence` after all branches)
  - *SG*: SpectreGuard

Synthetic Workloads

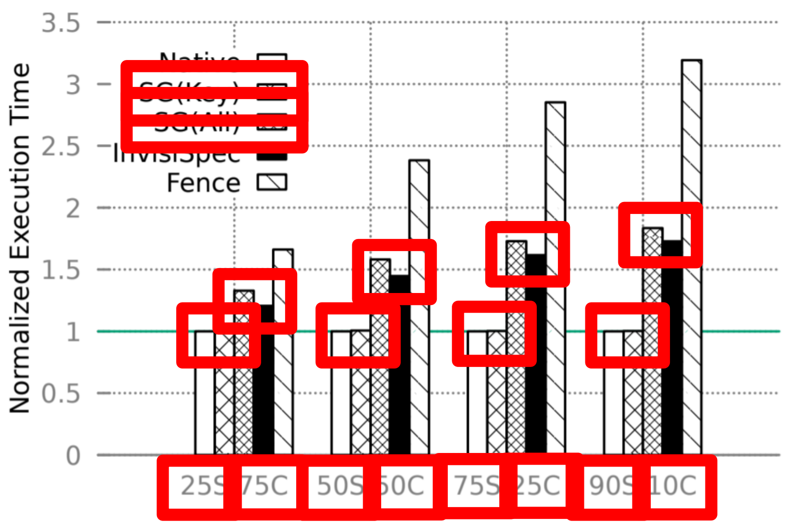
```
char *secret_key; // secret data

void benchmark(int S, int C)
{
    // (S)pectre gadget, unrelated to the secret
    for (i = 0; i < S; i++)
        do_work();

    // En(C)ryption task accessing the secret
    for (i = 0; i < C; i++)
        encrypt();
}
```

Secret data

- *(S)pectre*: contains Spectre gadget
  – does not access the secret key
- *En(C)ryption*: background communication
  – access the secret key

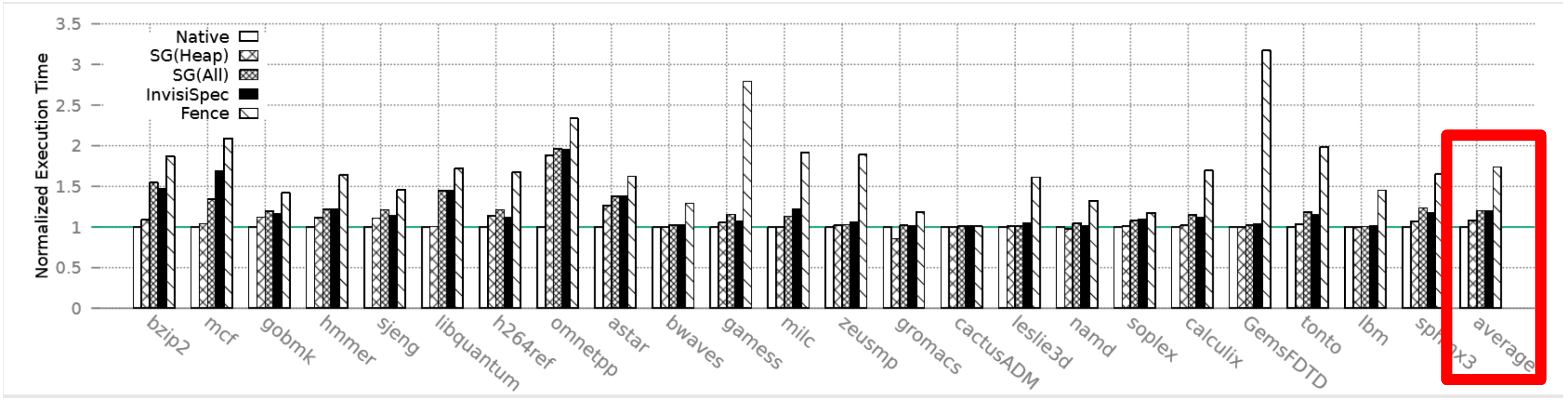
Results of Synthetic Workloads

```
char *secret_key; // secret data

void benchmark(int S, int C)
{
    // (S)pectre gadget, unrelated to the secret
    for (i = 0; i < S; i++)
        do_work();

    // En(C)ryption task accessing the secret
    for (i = 0; i < C; i++)
        encrypt();
}
```



- Varies percent time spent in S and C
- *SG(Key)* achieves native performance
  - Only secret key is marked as secret
- *SG(All) a*chieves comparable performance with *InvisiSpec*
  - <u>All memory</u> (code, data, heap, stack) is marked as secret

Results of SPEC2006 Benchmarks



- *SG(All)* achieves comparable performance with *InvisiSpec*
- *SG(Heap)* achieves better performance than *InvisiSpec*
  - Only heap is marked as non-speculative (NS) pages
- SpectreGuard enables targeted security and performance trade-offs

Summary

- Speculative execution attacks
  - Affect all high-performance out-of-order processors
  - Existing software mitigation suffers high programming complexity/overhead
  - Hardware only mitigation is costly
- SpectreGuard
  - A data-centric software/hardware collaborative defense mechanism
  - Low programming effort (identifying secret **data**, not vulnerable code)
  - Low hardware cost (no additional "shadow" structure)
  - Effective, targeted defense against Spectre attacks

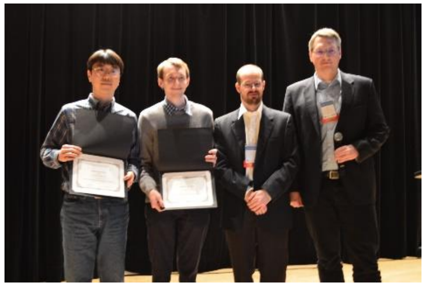
https://github.com/CSL-KU/SpectreGuard

Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention

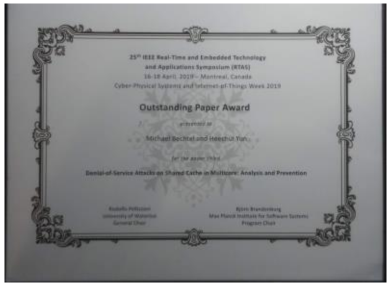
Michael Garrett Bechtel and Heechul Yun

*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*

Montreal, Canada, April, 2019



Outstanding Paper Award



25th IEEE Real-Time and Embedded Technology
and Applications Symposium (RTAS)
16-18 April, 2019 – Montreal, Canada
Cyber-Physical Systems and Internet-of-Things Week 2019

Outstanding Paper Award

presented to

Michael Bechtel and Heechul Yun

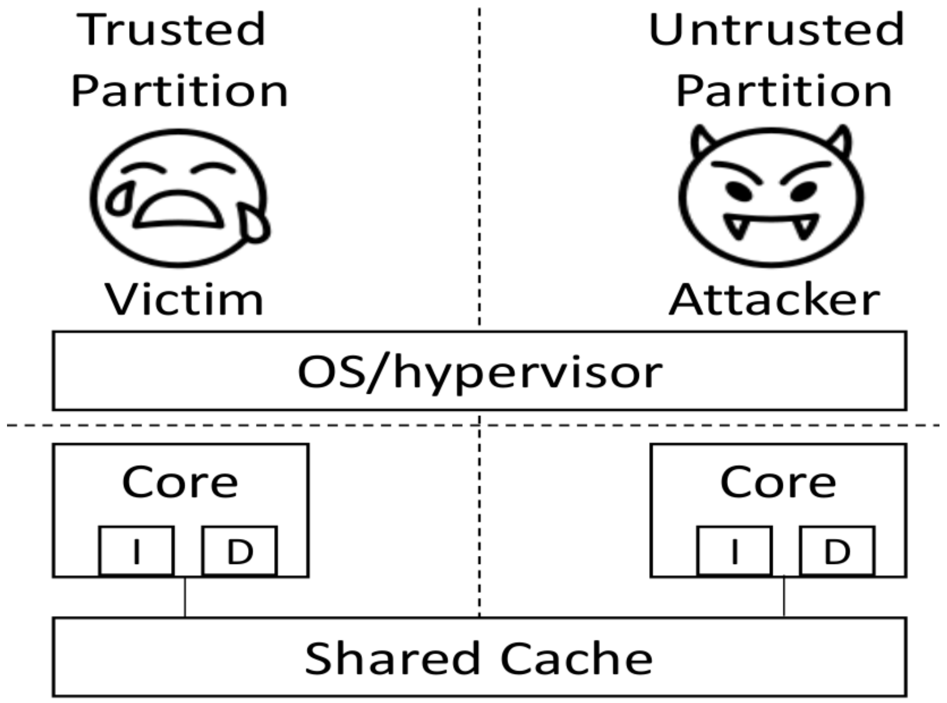
for the paper titled

Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention

Rodolfo Pellizzoni
University of Waterloo
General Chair

Björn Brandenburg
Max Planck Institute for Software Systems
Program Chair

Threat Model

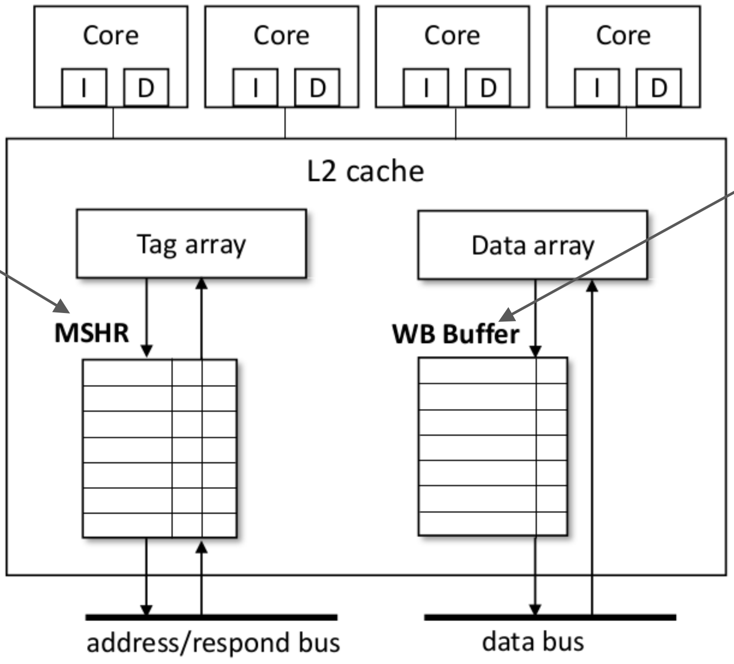


- Attacker's goal: increase the victim's task **execution time**
- The attacker is on different core/memory/cache partition
- The attacker can only execute non-privileged code.

Non-Blocking Cache

Miss Status Holding Registers[1]

- Track outstanding cache misses.



Writeback Buffer.

- Holds evicted dirty lines (writebacks).
- Prevents cache refills from waiting.

- We identified cache internal structures that can be potential DoS attack vectors

[1] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. "Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems." In *RTAS*, 2016 **(Best Paper Award)**

Cache DoS Attacks

```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        sum += ptr[i];
}
```

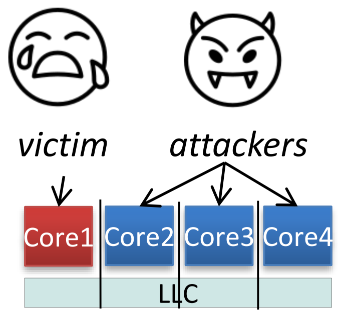
Read Attacker
(target **MSHRs**)

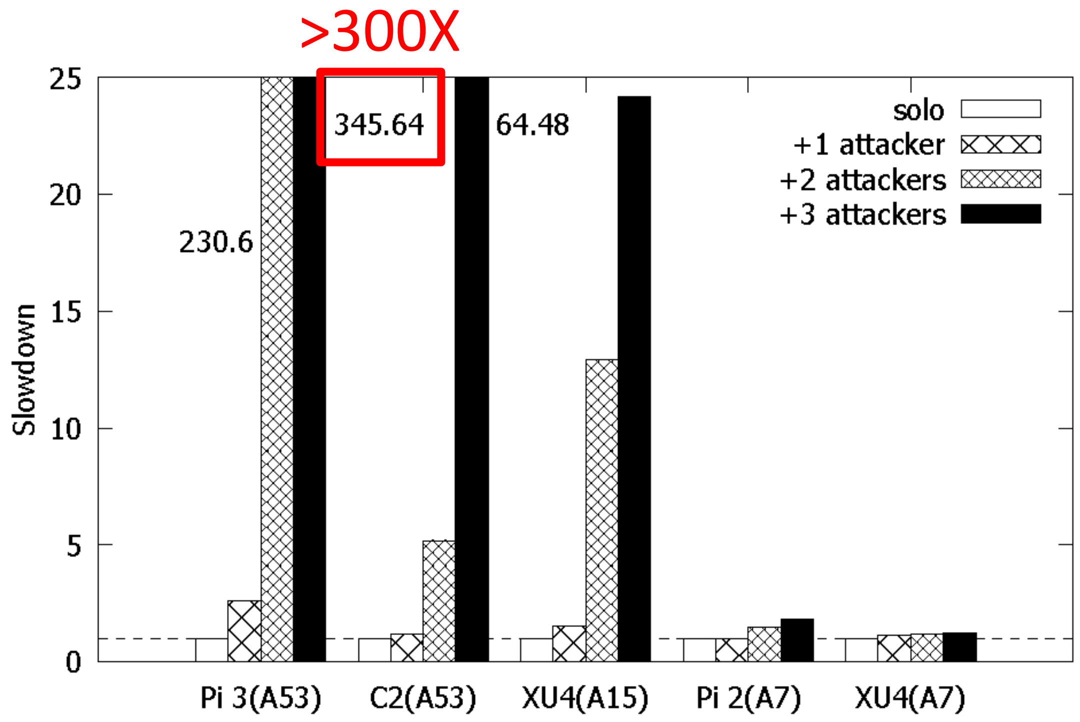
```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        ptr[i] = 0xff;
}
```

Write Attacker
(target **WBBuffer**)

- Denial-of-Service (DoS) attacks targeting internal hardware structures of a shared cache.
  - Block the cache → delay the victim's execution time

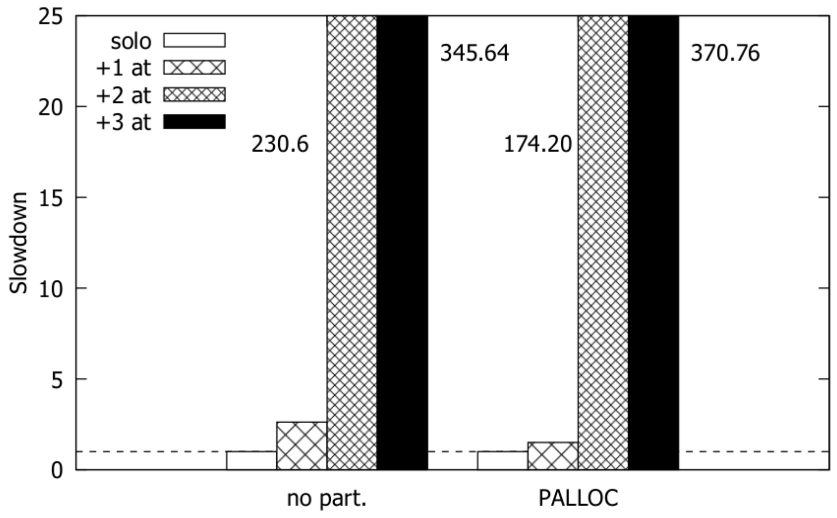
Effects of Cache DoS Attacks

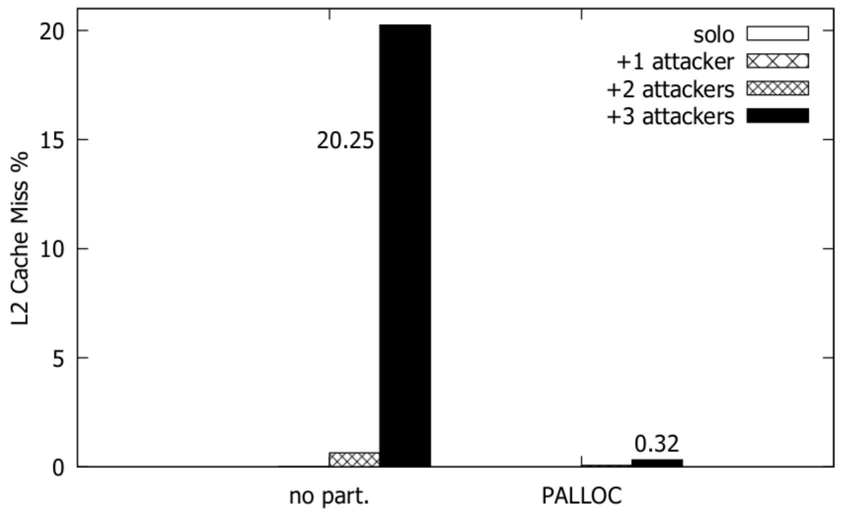




- Observed worst-case: >300X (times) slowdown
  - On popular in-order multicore processors
  - Due to contention in cache write-back buffer

Effect of Cache Partitioning





PALLOC[1] partitions the cache among the cores

- Partitioning doesn't protect against DoS attacks.
  - because cache internal structures are not partitioned.

[1] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni. PALLOC: DRAM Bank-Aware Memory Allocator for Performan ce Isolation on Multicore Platforms. In *RTAS*, 2014

Summary

- Cache internal hardware structures (MSHRs, WriteBack buffer) are viable DoS attack vectors in multicore platforms.
- Traditional cache partitioning is not effective for cache DoS attacks
- We proposed an OS solution to defense against cache DoS attacks.

RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems

Waqar Ali and Heechul Yun.

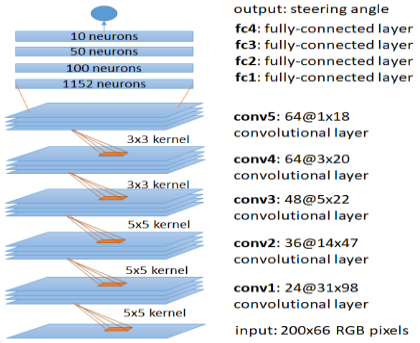
*IEEE Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*

Montreal, Canada, April, 2019

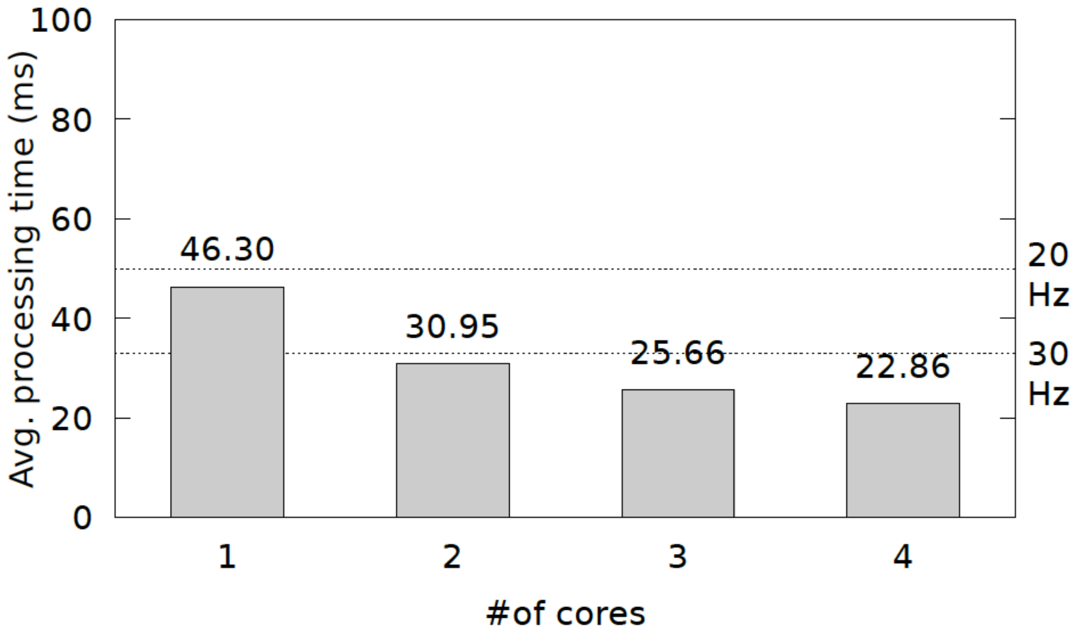
Parallel Real-Time Tasks

- Many emerging workloads in AI, vision, robotics are parallel real-time tasks

DNN based real-time control[+]


output: steering angle
fc4: fully-connected layer
fc3: fully-connected layer
fc2: fully-connected layer
fc1: fully-connected layer
10 neurons
50 neurons
100 neurons
1152 neurons
conv5: 64@1x18 convolutional layer
3x3 kernel
conv4: 64@3x20 convolutional layer
3x3 kernel
conv3: 48@5x22 convolutional layer
5x5 kernel
conv2: 36@14x47 convolutional layer
5x5 kernel
conv1: 24@31x98 convolutional layer
5x5 kernel
input: 200x66 RGB pixels

Effect of parallelization on DNN control task



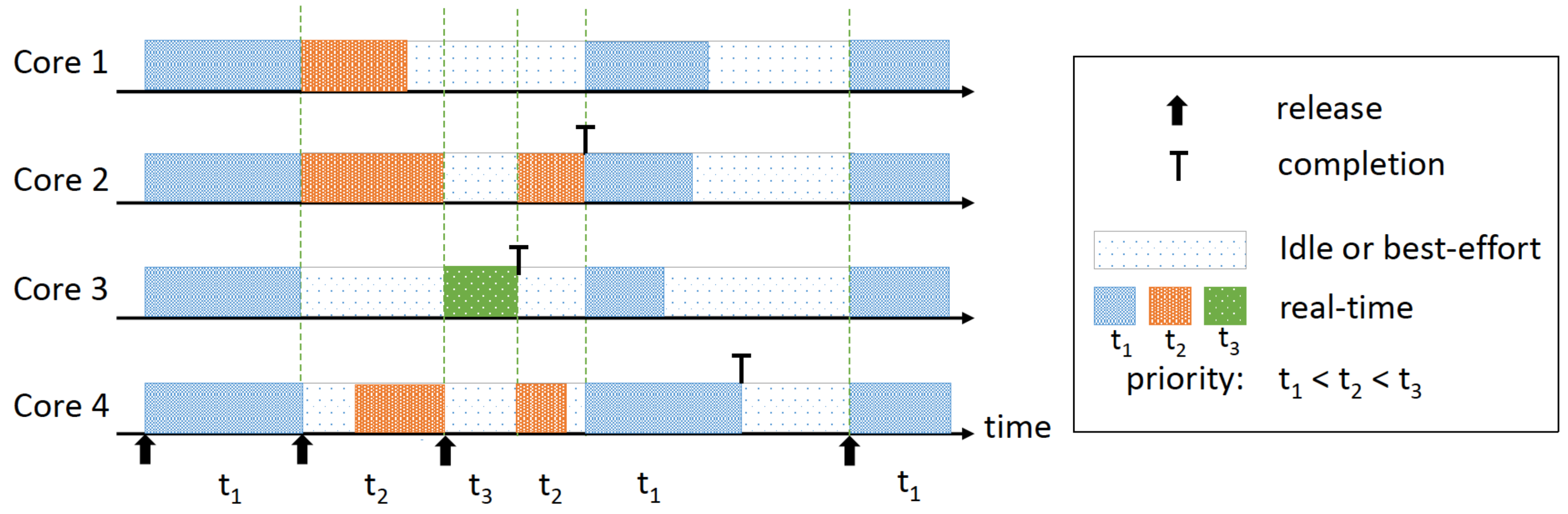
[+] M. Bojarski, "End to End Learning for Self-Driving Cars." arXiv:1604.07316, 2016

Observations

- Constructive sharing (Good)
  – Between threads of a single parallel task
- Destructive sharing (Bad)
  – Between threads of different tasks

- **Goal: analyzable and efficient parallel real-time task scheduling framework for multicore**
  - By avoiding destructive sharing

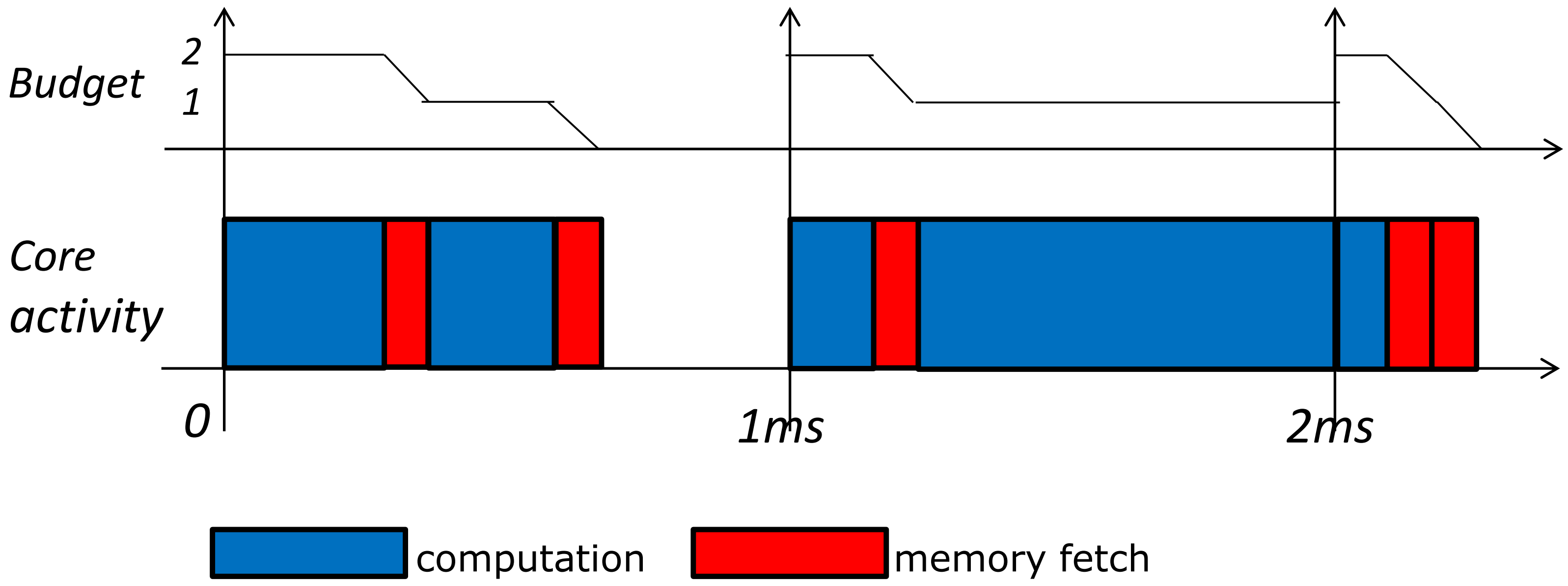
RT-Gang



- **One (parallel) real-time task---a gang---at a time**
  - Eliminate inter-task interference by construction
- Schedule best-effort tasks during slacks **w/ throttling**
  - Improve utilization with bounded impacts on the RT tasks

Safe Best-Effort Task Throttling

- Throttle the best-effort core(s) if it exceeds a given bandwidth budget **set by the RT task**



Basic throttling mechanism *

* Yun et al., “MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Pl atforms.” In *RTAS*, 2013

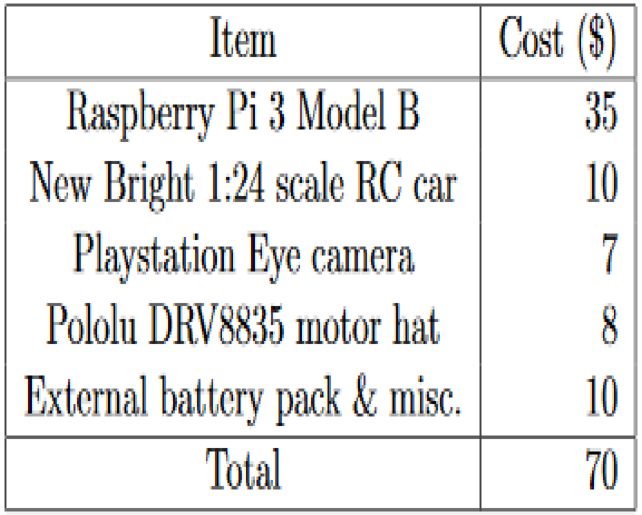
Implementation

- Modified Linux's RT scheduler
  - Implemented as a "feature" of SCHED_FIFO (sched/rt.c)
- Best-effort task throttling
  - Based on BWLOCK++[*]

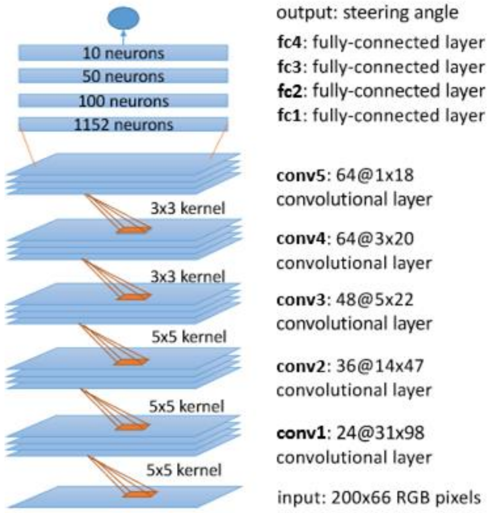
[*] W. Ali and H. Yun., "Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms." In *ECRTS*, 2018

DeepPicar*

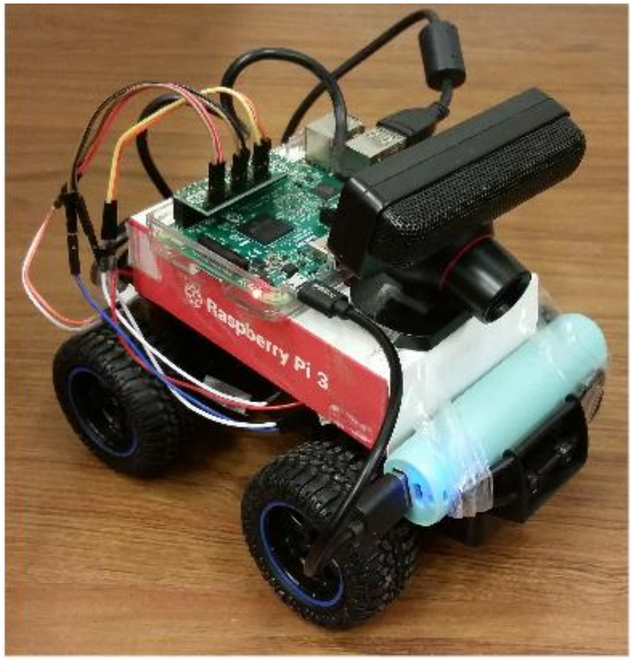
- A **low cost**, small scale replication of NVIDIA's DAVE-2
- **Uses the exact same DNN**
- Runs on a Raspberry Pi 3 in **real-time**



Item	Cost ($)
Raspberry Pi 3 Model B	35
New Bright 1:24 scale RC car	10
Playstation Eye camera	7
Pololu DRV8835 motor hat	8
External battery pack & misc.	10
Total	70



10 neurons
50 neurons
100 neurons
1152 neurons
3x3 kernel
3x3 kernel
5x5 kernel
5x5 kernel
5x5 kernel
output: steering angle
fc4: fully-connected layer
fc3: fully-connected layer
fc2: fully-connected layer
fc1: fully-connected layer
conv5: 64@1x18 convolutional layer
conv4: 64@3x20 convolutional layer
conv3: 48@5x22 convolutional layer
conv2: 36@14x47 convolutional layer
conv1: 24@31x98 convolutional layer
input: 200x66 RGB pixels



* Bechtel et al. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. In *RTCSA*, 2018
https://github.com/mbechtel2/DeepPicar-v2

DNN based Real-Time Control

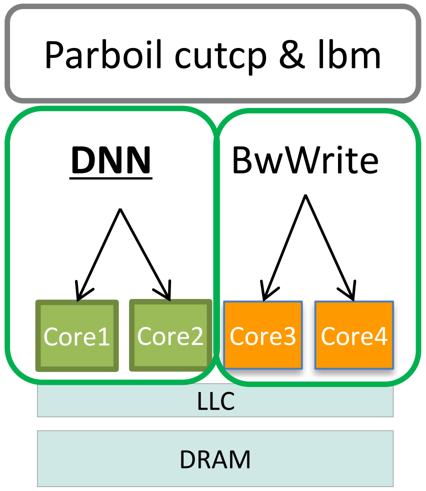
```
while True:
  # 1. read from the forward camera
  frame = camera.read()
  # 2. convert to 200x66 rgb pixels
  frame = preprocess(frame)
  # 3. perform inferencing operation
  angle = DNN_inferencing(frame)
  # 4. motor control
  steering_motor_control(angle)
  # 5. wait till next period begins
  wait_till_next_period()
```

- DNN Inferencing is the most compute intensive part.
- Parallelized by TensorFlow to utilize multiple cores.

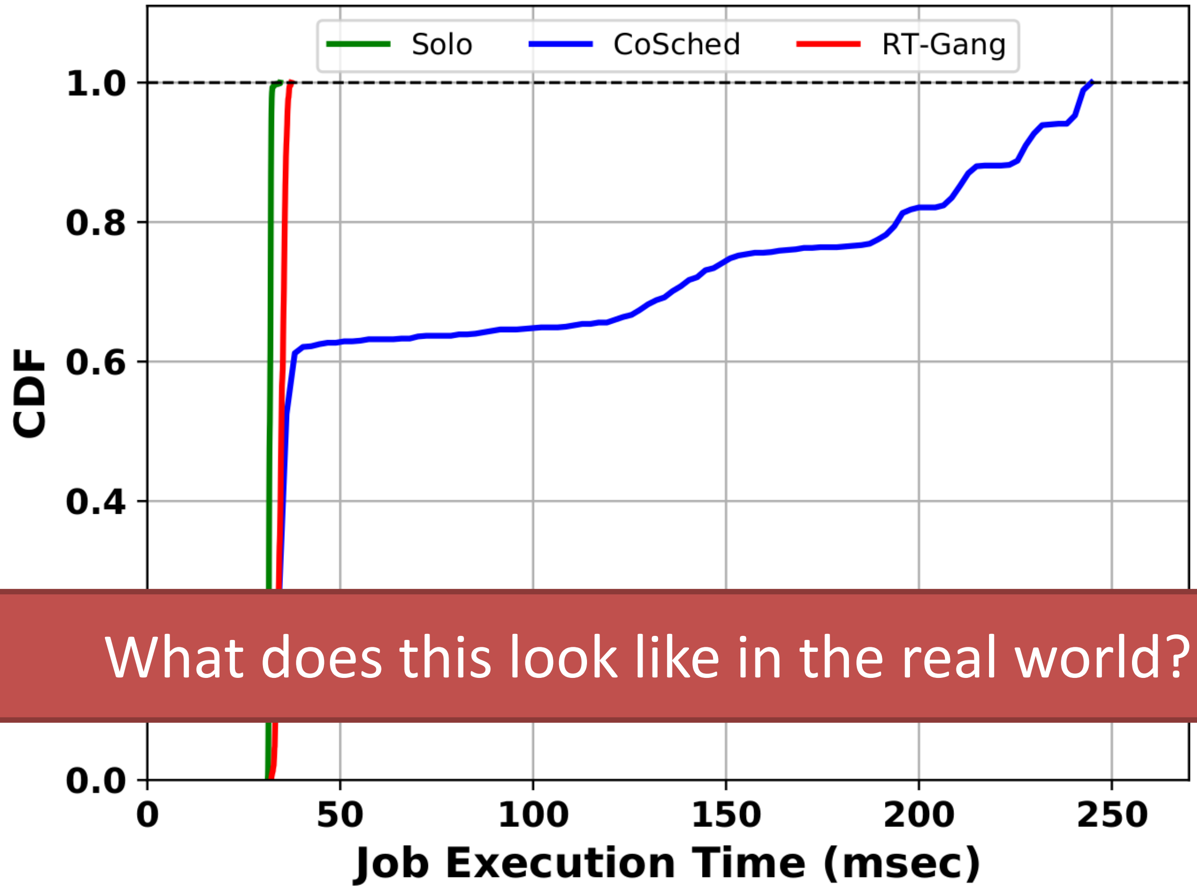
Experiment Setup

- DNN control task of DeepPicar (real-world RT)
- IsolBench BwWrite benchmark (synthetic RT)
- Parboil benchmarks (real-world BE)

	Task	WCET (C ms)	Period (P ms)	# Threads
RT	t^{rt}_{dnn}	**34**	**100**	**2**
	t^{rt}_{bww}	**220**	**340**	**2**
BE	t^{be}_{cutcp}	∞	N/A	4
	t^{be}_{lbm}	∞	N/A	4



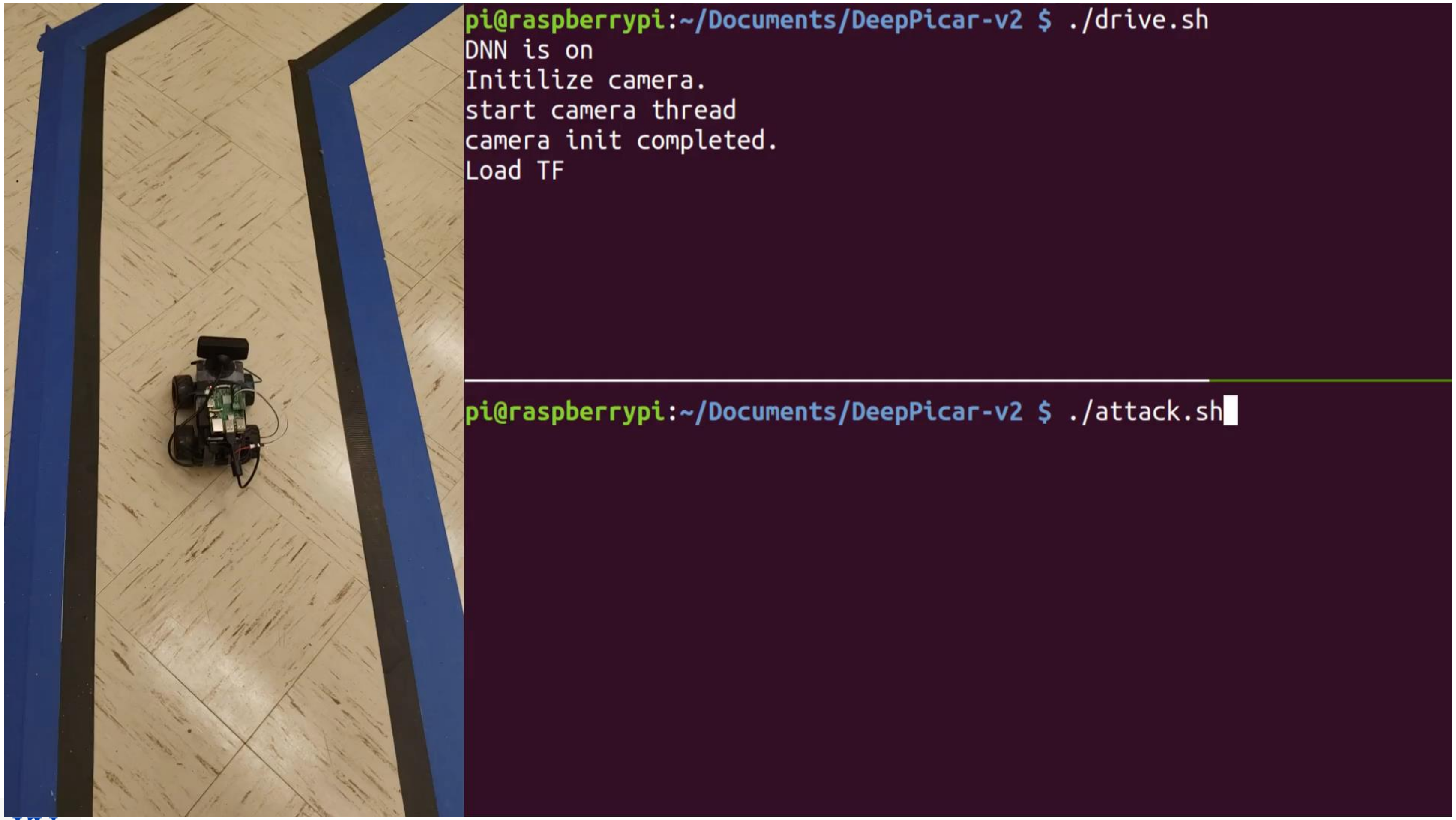
Execution Time Distribution



What does this look like in the real world?

- RT-Gang achieves deterministic timing

CoSched (w/o RT-Gang)

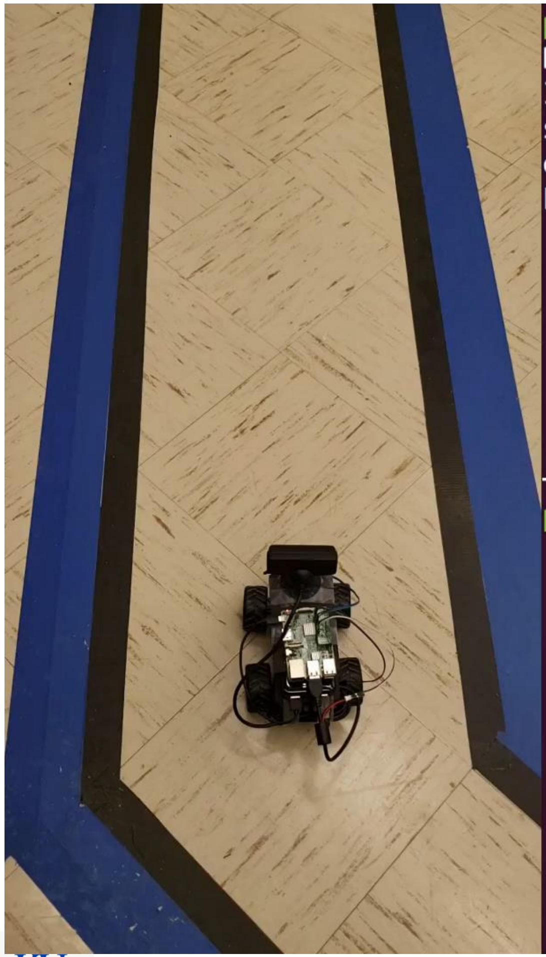


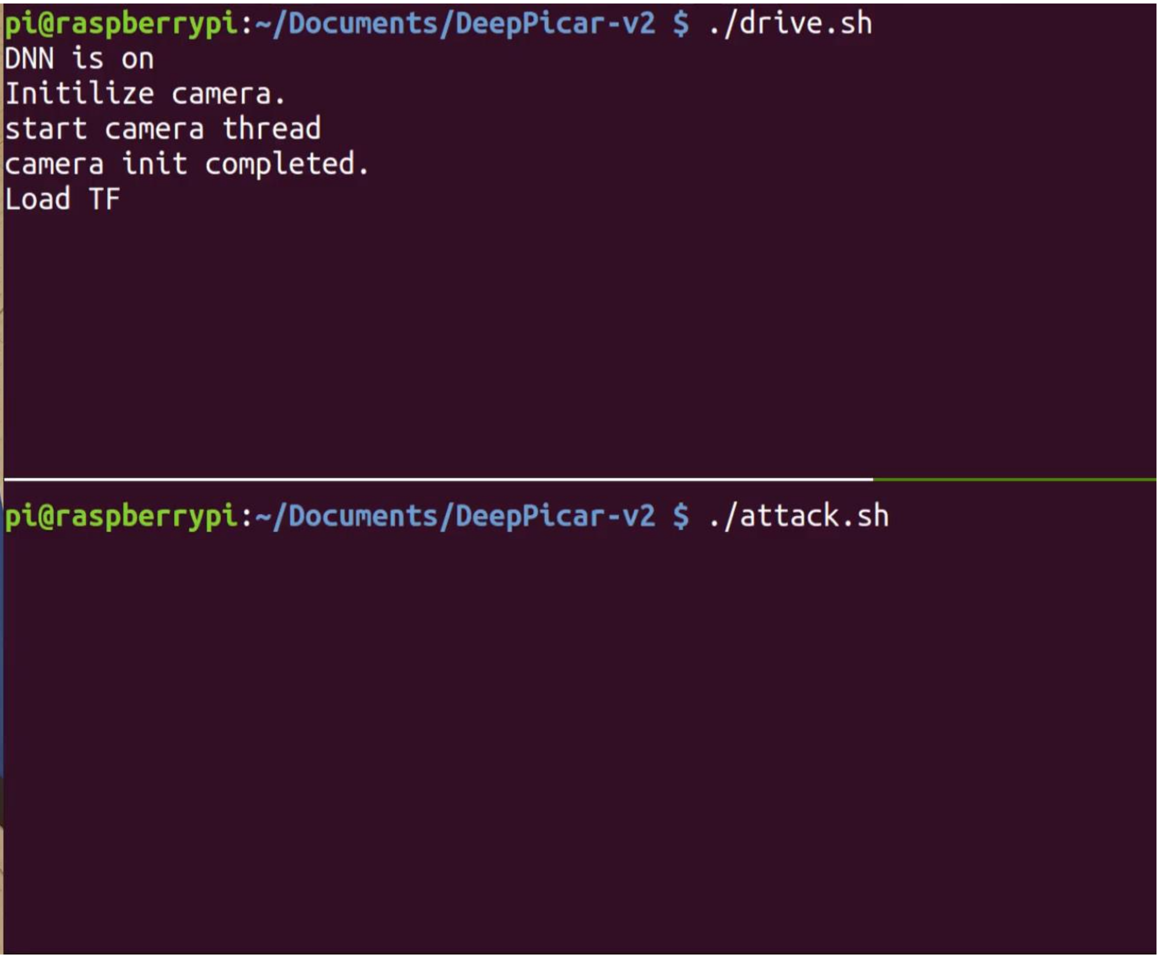
```
pi@raspberrypi:~/Documents/DeepPicar-v2 $ ./drive.sh
DNN is on
Initilize camera.
start camera thread
camera init completed.
Load TF
```

```
pi@raspberrypi:~/Documents/DeepPicar-v2 $ ./attack.sh
```

https://youtu.be/Jm6KSDqlqiU

RT-Gang





```
pi@raspberrypi:~/Documents/DeepPicar-v2 $ ./drive.sh
DNN is on
Initilize camera.
start camera thread
camera init completed.
Load TF
```

```
pi@raspberrypi:~/Documents/DeepPicar-v2 $ ./attack.sh
```

https://youtu.be/pk0j063cUAs

Summary

- Parallel real-time task scheduling
  - Hard to analyze on COTS multicore
  - Due to interference in shared memory hierarchy
- RT-Gang
  - **Analyzable** and **efficient** parallel real-time gang scheduling framework, implemented in Linux
  - Avoid interference by construction
    - Can protect critical real-time tasks

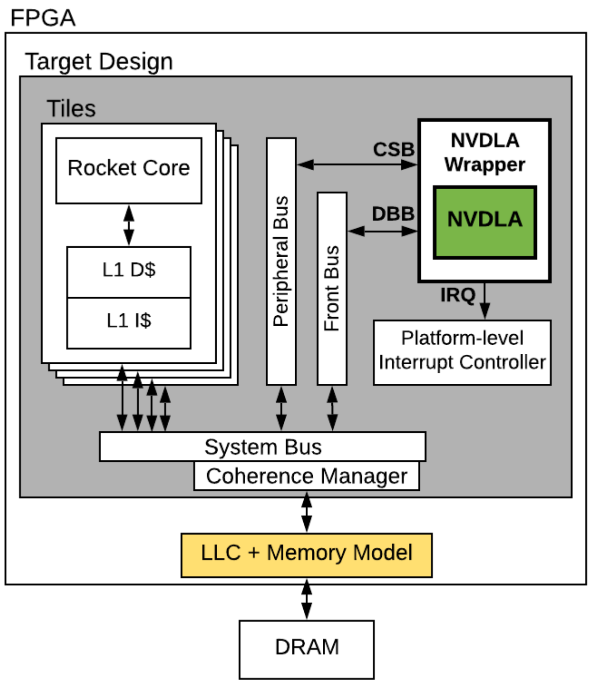
https://github.com/CSL-KU/rt-gang

Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim.

Farzad Farshchi, Qijing Huang, and Heechul Yun.

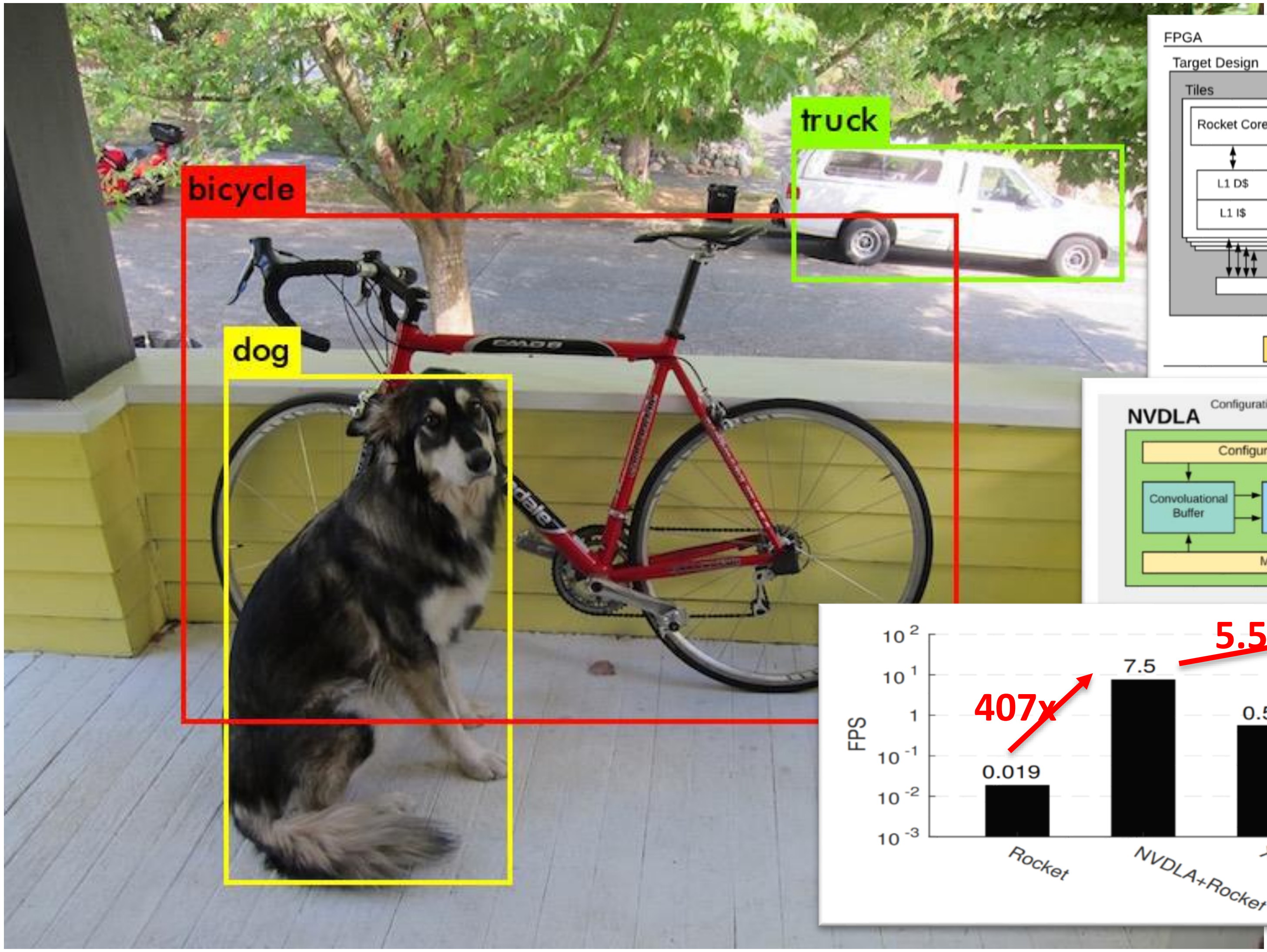
Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC^2) Washington DC, February, 2019.

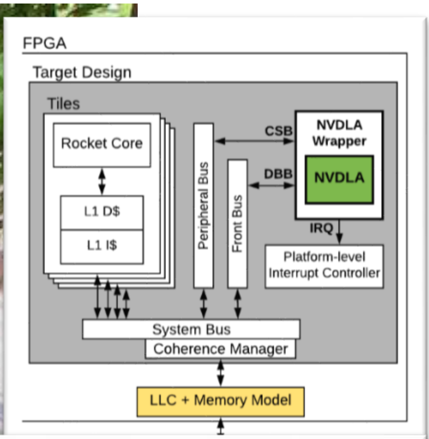
RISC-V + NVDLA SoC Platform

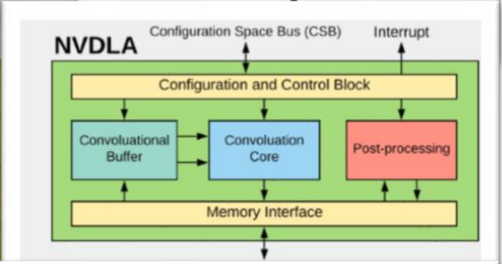


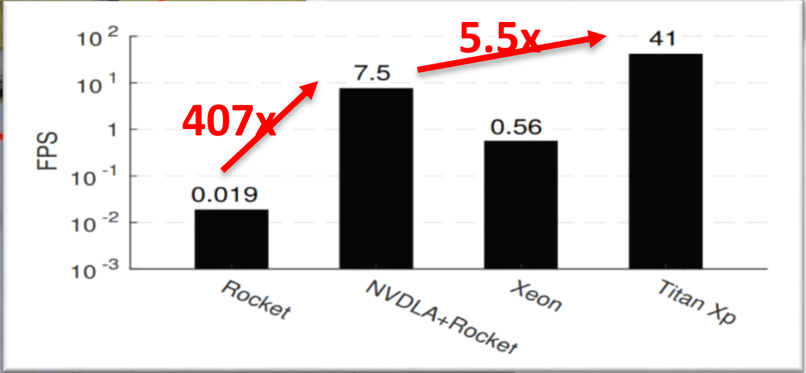
- Full-featured quad-core SoC with hardware DNN accelerator on Amazon FPGA cloud
  – Run Linux, YOLO v3 object detection

RISC-V + NVDLA SoC Platform









Conclusion

- Micro-architectural attacks on high-performance embedded SoCs are a serious threat for CPS
  - Can leak secret (confidentiality)
  - Can alter data (integrity)
  - Can affect real-time performance (correctness)

- Our research develops fundamental computing infrastructure technologies to enable safe, secure, and intelligent CPS

Thank You!

Acknowledgement:

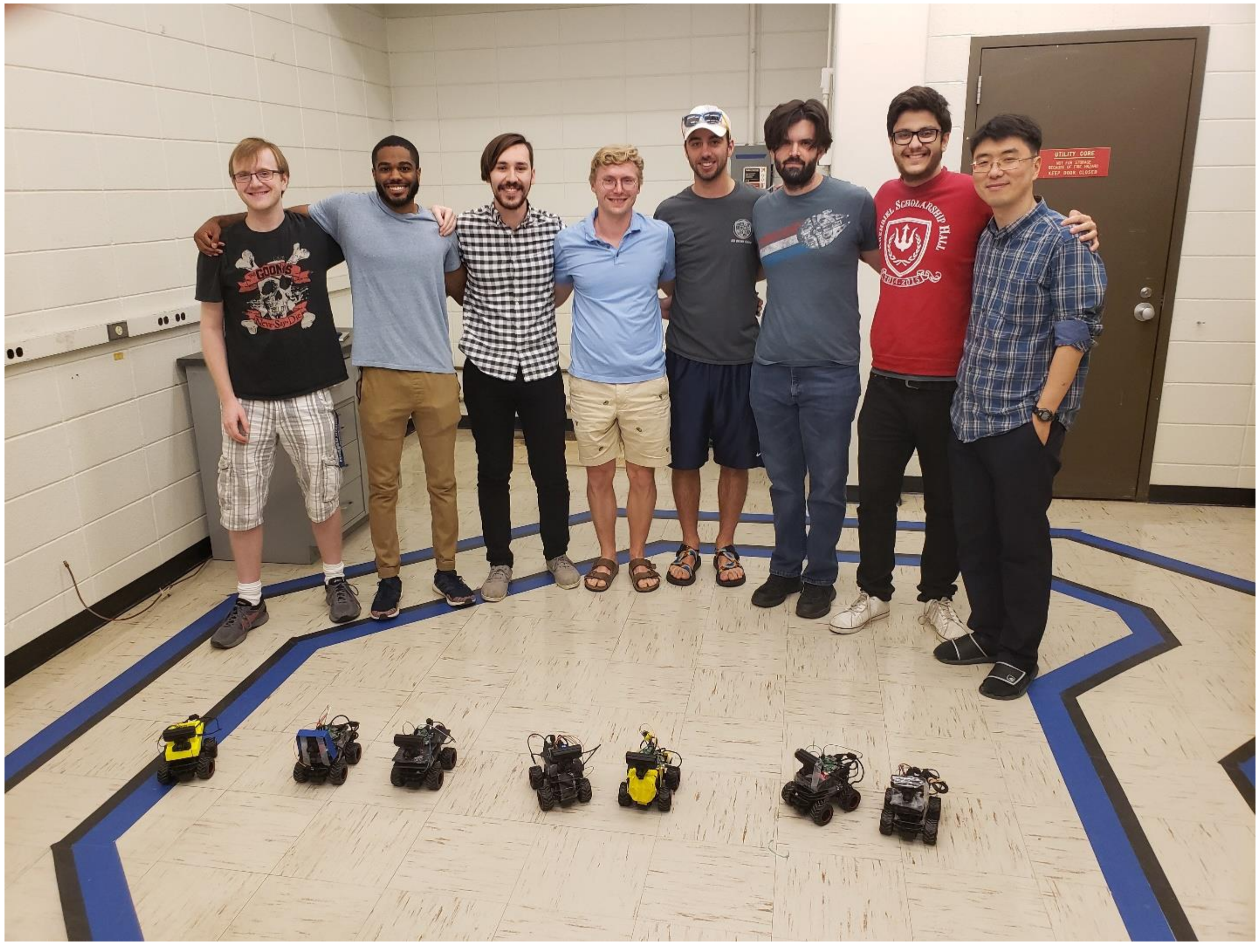
This research is supported by NSA Science of Security initiative contract #H98230-18-D-0009 and NSF CNS 1718880, 1815959.

Recent Publications

1. [C] Jacob Michael Fustos, Farzad Farshchi, and Heechul Yun. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. *Design Automation Conference (**DAC**)*, 2019
2. [C] Waqar Ali and Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2019.
3. [C] Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2019 **Outstanding Paper Award**
4. [W] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (**EMC^2**)*, 2019.
5. [C] Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (**RTCSA**)*, 2018
6. [C] Waqar Ali, Heechul Yun. Protecting Real-Time GPU Applications on Integrated CPU-GPU SoC Platforms. *Euromicro Conference on Real-Time Systems (**ECRTS**)*, 2018
7. [C] Farzad Farshchi, Prathap Kumar Valsan, Renato Mancuso, Heechul Yun. Deterministic Memory Abstraction and Supporting Multicore System Architecture. *Euromicro Conference on Real-Time Systems (**ECRTS**)*, 2018
8. [J] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. Addressing Isolation Challenges of Non-blocking Caches for Multicore Real-Time Systems. ***Real-time Systems***, Vol: 53, Issue: 5, pp: 673–708, 2017
9. [J] Heechul Yun, Waqar Ali, Santosh Gondi, Siddhartha Biswas. BWLOCK: A Dynamic Memory Access Control Framework for Soft Real-Time Applications on Multicore Platforms. ***IEEE Transactions on Computers***, Vol: 66, Issue: 7, pp: 1247-1252, 2017
10. [C] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, Shawn Keshmiri. A Simplex Architecture for Intelligent and Safe Unmanned Aerial Vehicles. *IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications (**RTCSA**)*, 2016. **Best Student Paper Nomination**
11. [C] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi . Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (**RTAS**)*, 2016. **Best Paper Award**
12. [C] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory Bandwidth Management for Efficient Performance Isolation in Multi-core Platforms, ***IEEE Transactions on Computers***, Vol 65, Issue 2, 2016, pp. 562 – 576. **Editor's Pick of the year 2016**

Full List: http://www.ittc.ku.edu/~heechul/pub.html

EECS 753 DeepPicar Competition



DeepPicar Competition

EECS 753 Embedded Real-Time Systems Final Project

May 6, 2019