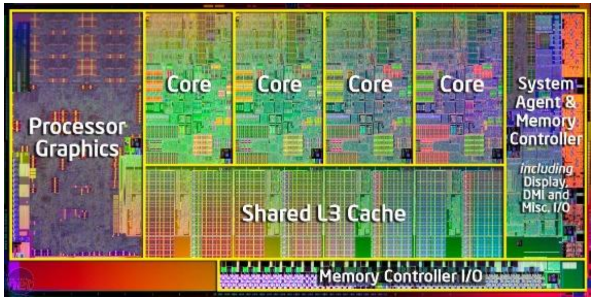
RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems

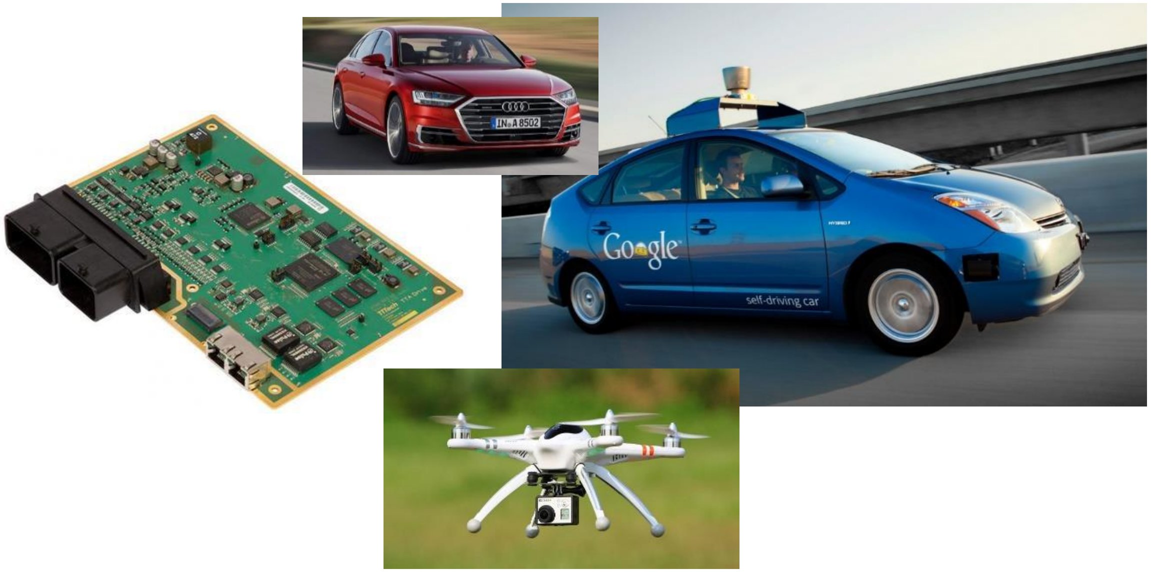
Waqar Ali, Heechul Yun

University of Kansas

Multicore Processors

- Provide high computing performance
- Needed for intelligent safety-critical real-time systems

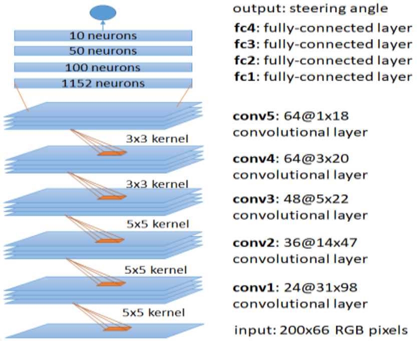




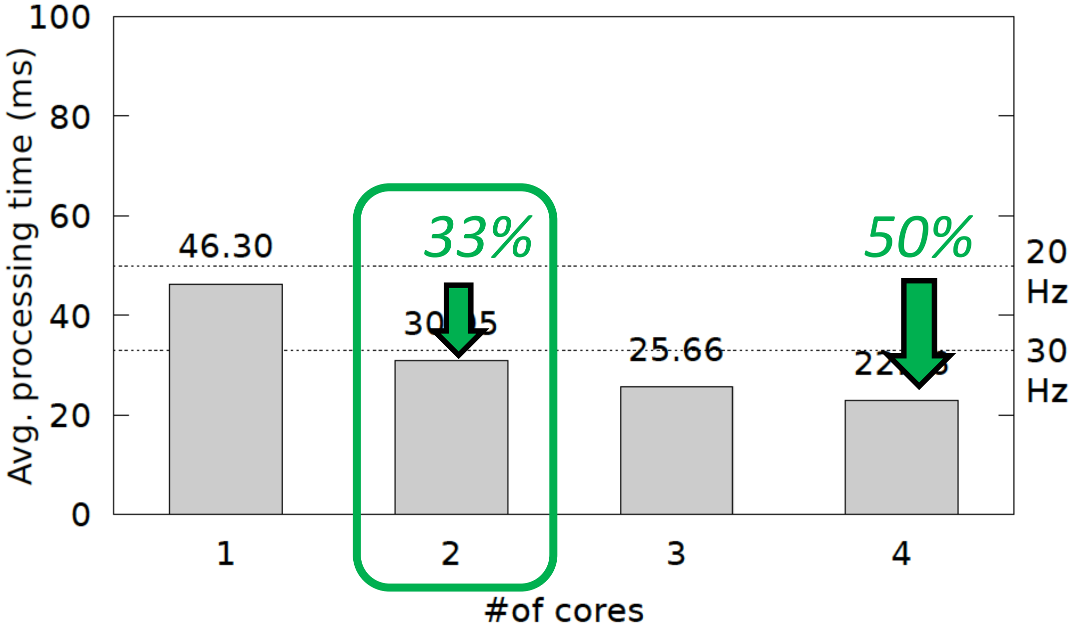
Parallel Real-Time Tasks

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

DNN based real-time control

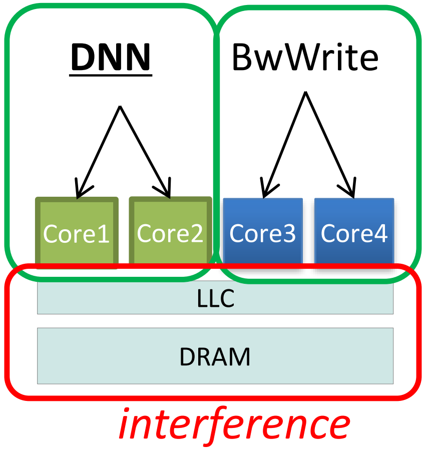

output: steering angle
10 neurons
50 neurons
100 neurons
1152 neurons
fc4: fully-connected layer
fc3: fully-connected layer
fc2: fully-connected layer
fc1: fully-connected layer
conv5: 64@1x18 convolutional layer
3x3 kernel
conv4: 64@3x20 convolutional layer
3x3 kernel
conv3: 48@5x22 convolutional layer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

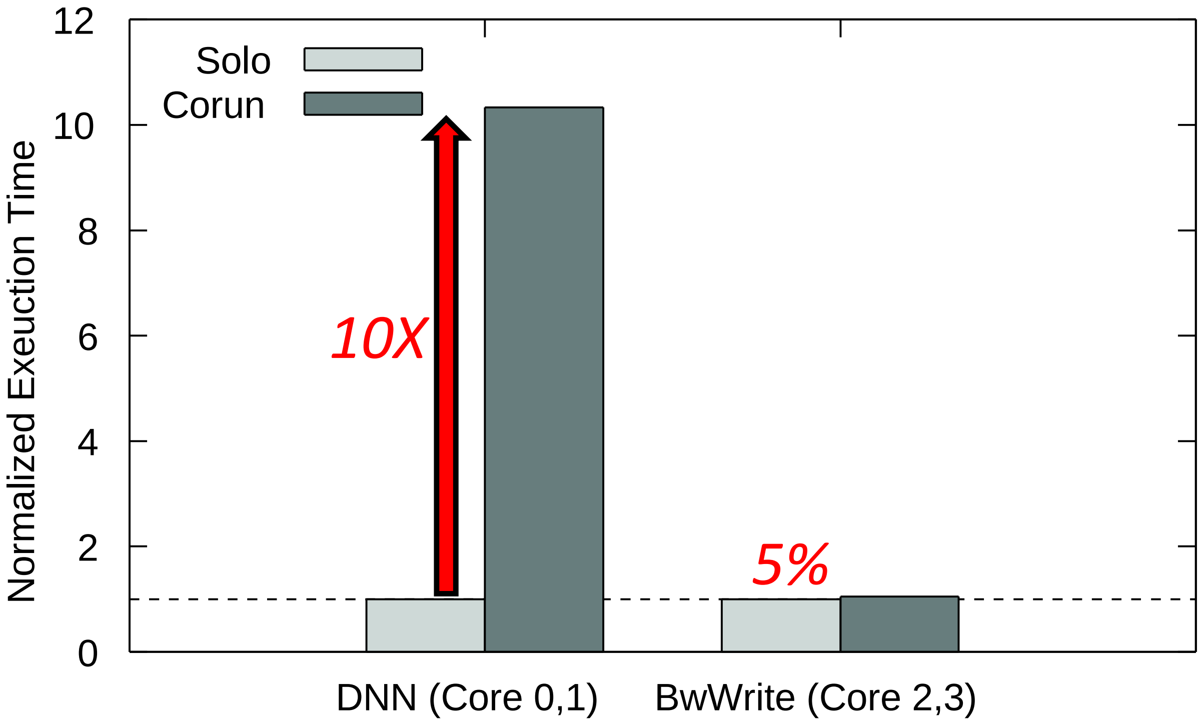
Effect of parallelization on DNN control task



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

Effect of Co-Scheduling





- DNN control task suffers **>10X slowdown**
  – Due to inte

It can be worse! [Bechtel, RTAS'19]

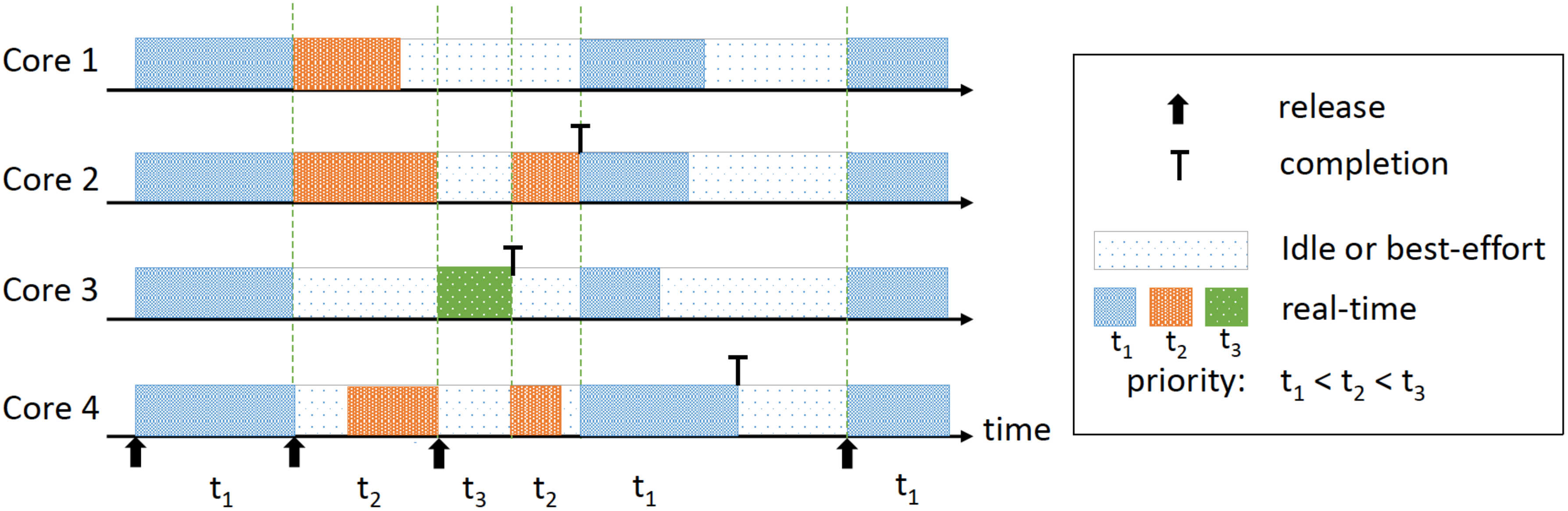
[Bechtel, RTAS'19] Michael G. Bechtel and Heechul Yun. "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention." In *RTAS*, 2019 (to appear)

Observations

- Interference in shared memory hierarchy
  - Can be very high and unpredictable
  - Depends on the hardware (black box)
- 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**

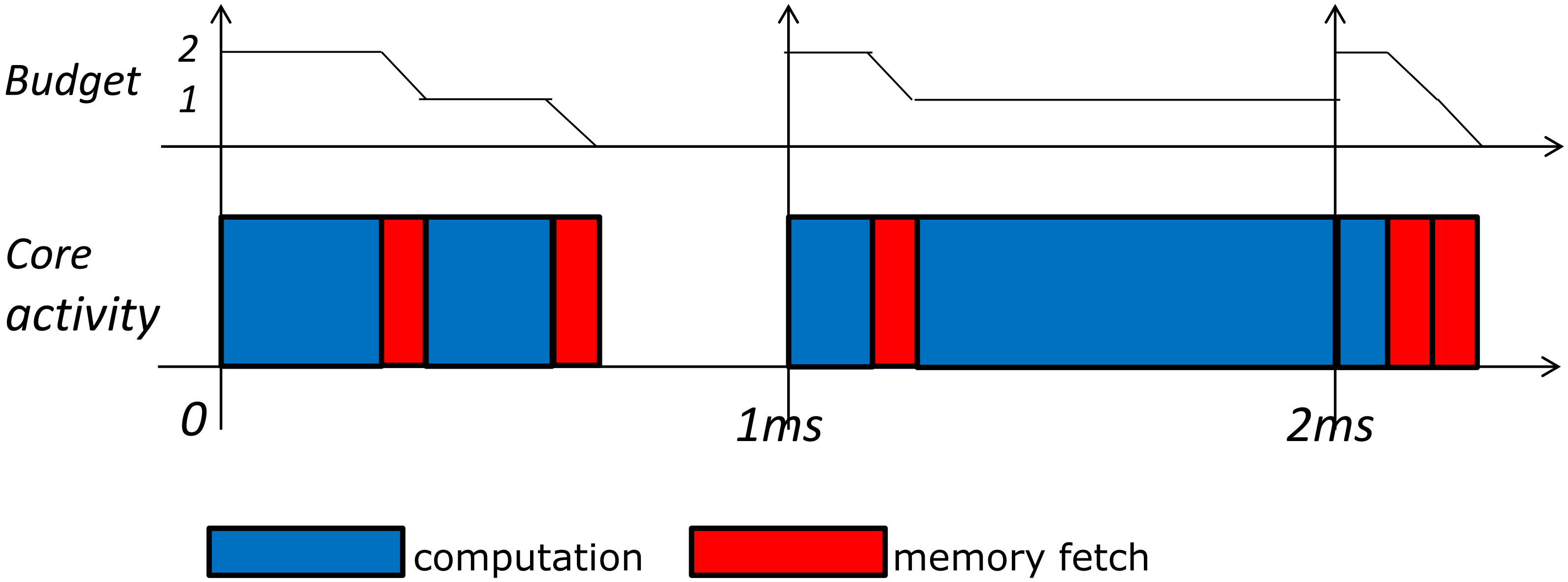
RT-Gang



- **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

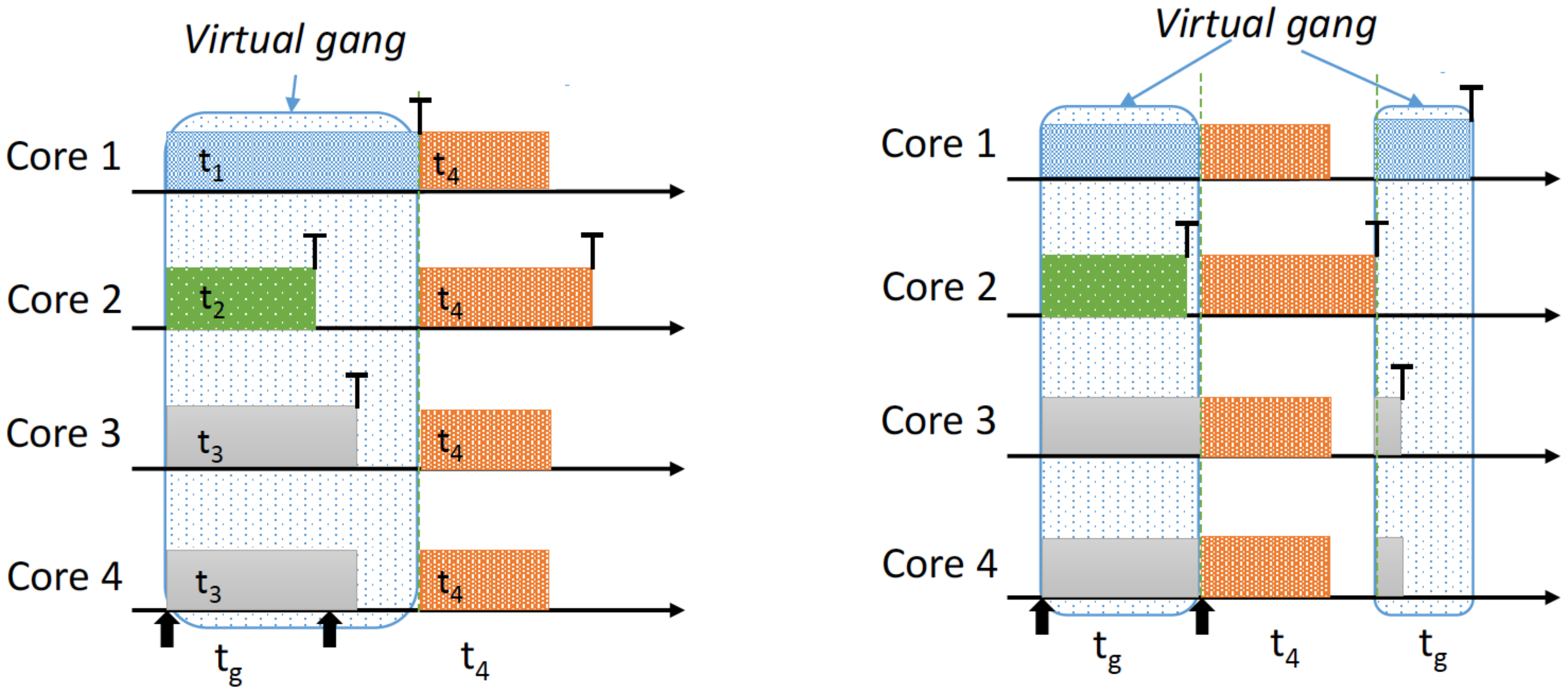
- Throttle the best-effort core(s) if it exceeds a given bandwidth budget **set by the RT task**



Throttling mechanism [Yun, RTAS'13]

[Yun, RTAS'13] Yun et al., "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms." In *RTAS*, 2013

Virtual Gang



(a) prio (tg) > prio (t4)

(b) prio (tg) < prio (t4)

- **Statically** group RT tasks as a “virtual gang”
  - All threads of a virtual gang are scheduled together

Implementation

- Modified Linux's RT scheduler
  - Implemented as a "feature" of SCHED_FIFO (sched/rt.c)
  - Enforce one real-time priority across all cores (invariant)
  - A high priority RT thread preempts lower priority RT threads on any cores (gang preemption)

- Best-effort task throttling
  - Based on BWLOCK++ [Ali, ECRTS'18]
  - Each RT task sets the tolerable throttling threshold
  - Enforced by the kernel-level bandwidth regulators for any co-scheduled best-effort tasks

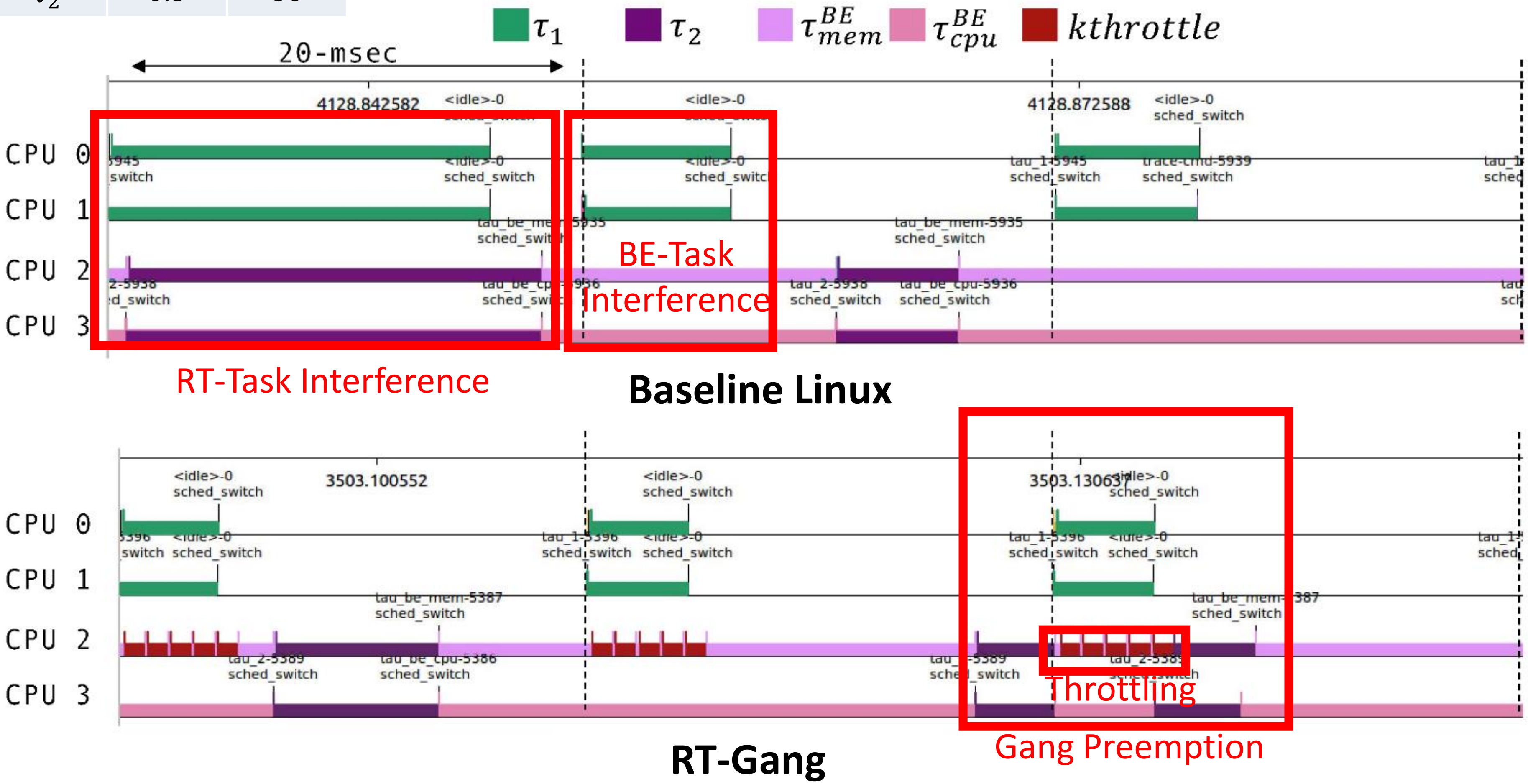
[Ali, ECRTS'18] W. Ali and H. Yun., "Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms." In *ECRTS*, 2018

Evaluation

- Setup
  - Linux 4.14 baseline
  - Raspberry Pi 3 (4x Cortex-A53)
  - NVIDIA Jetson TX2 (4x Cortex-A57)
- Benchmarks
  - IsolBench (synthetic RT/BE)
  - DNN control task of DeepPicar (real-world RT)
  - Parboil benchmarks (real-world BE)

RT Task	WCET (ms)	Period (ms)
τ_1	3.5	20
τ_2	6.5	30

Synthetic Taskset



RT-Task Interference

BE-Task Interference

Baseline Linux

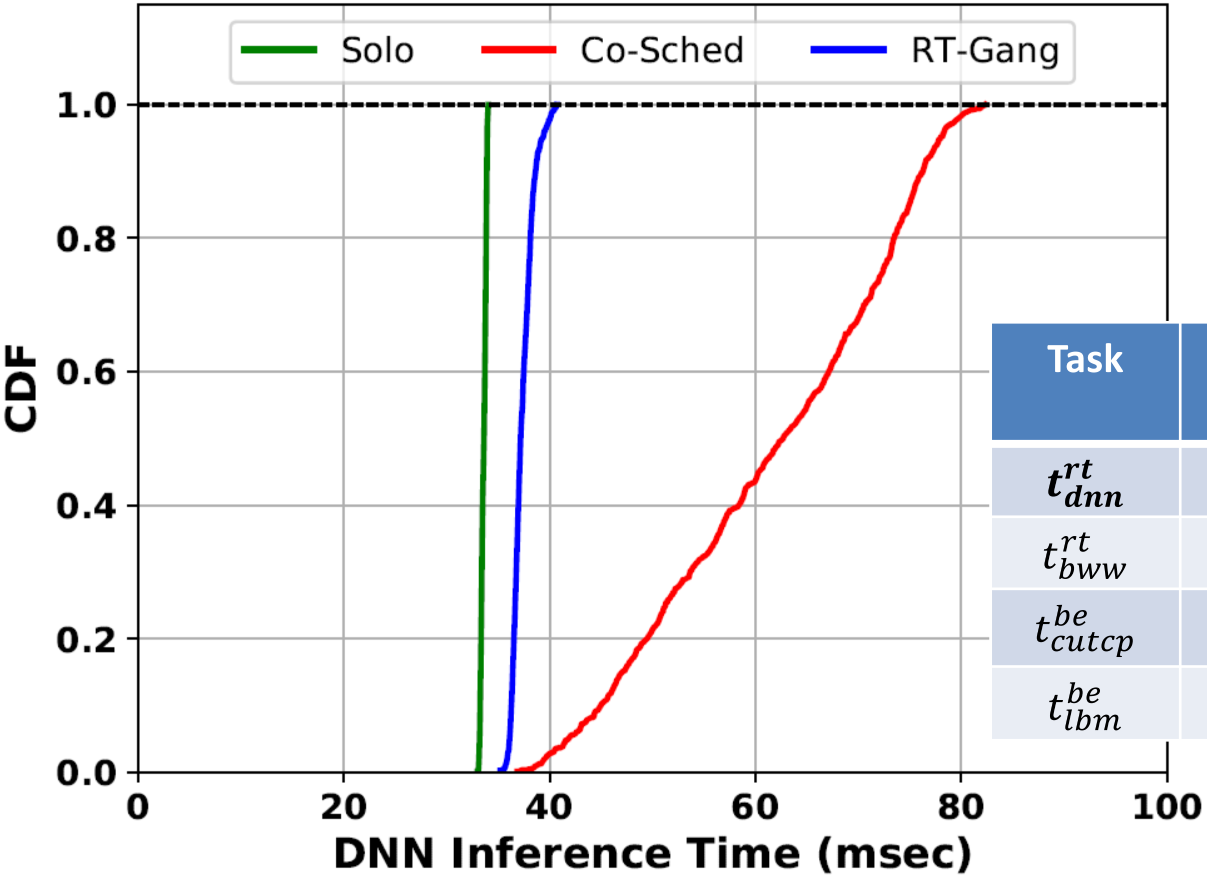
Throttling

Gang Preemption

RT-Gang

Deterministic timing is achieved

DNN Taskset



Task	WCET (C ms)	Period (P ms)	# Threads
t_{dnn}^{rt}	**34**	**78**	**2**
t_{bww}^{rt}	47	100	4
t_{cutcp}^{be}	∞	N/A	4
t_{lbm}^{be}	∞	N/A	4

Deterministic timing is achieved

Related Work

- Gang scheduling
  - J. Goossens et al. “Gang FTP scheduling of periodic and parallel rigid real-time tasks.” In *RTNS*, 2010
  - S. Kato et al. “Gang EDF scheduling of parallel task systems.” In *RTSS*, 2009
  - A. Melani et al., “A scheduling framework for handling integrated modular avionic systems on multicore platforms.” In *RTCSA*, 2017

- Key differences of our work
  - First gang scheduling **implementation on an actual OS**
  - Integrate throttling to safely co-schedule best-effort tasks

Conclusion

- 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

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

Thank You!

Disclaimer:

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