Protecting Real-Time GPU Kernels in Integrated CPU-GPU SoC Platforms

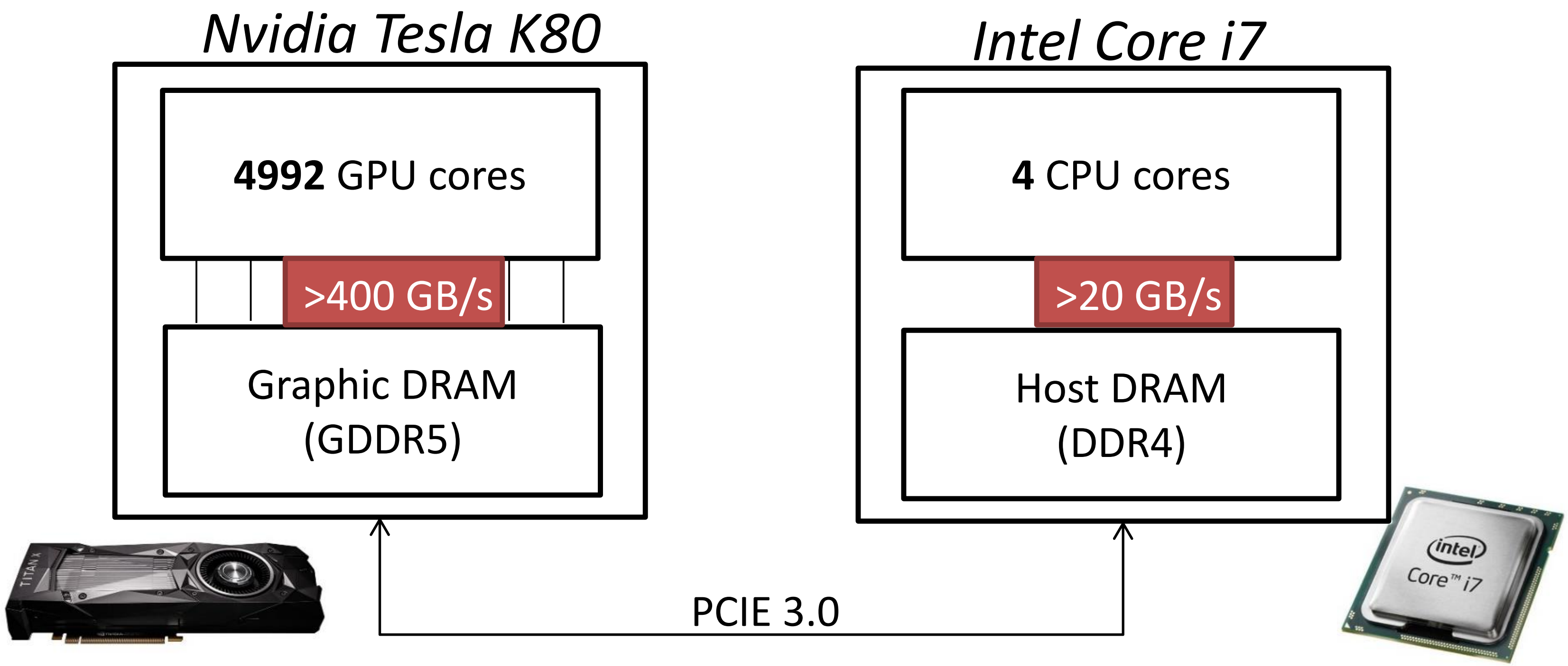
Waqar Ali, Heechul Yun

University of Kansas

GPU in Autonomous CPS

- Needed for real-time processing of high bandwidth sensor data (e.g., vision), deep neural networks, AI, etc.
- Must meet size, weight, and power (SWaP) and cost constraints

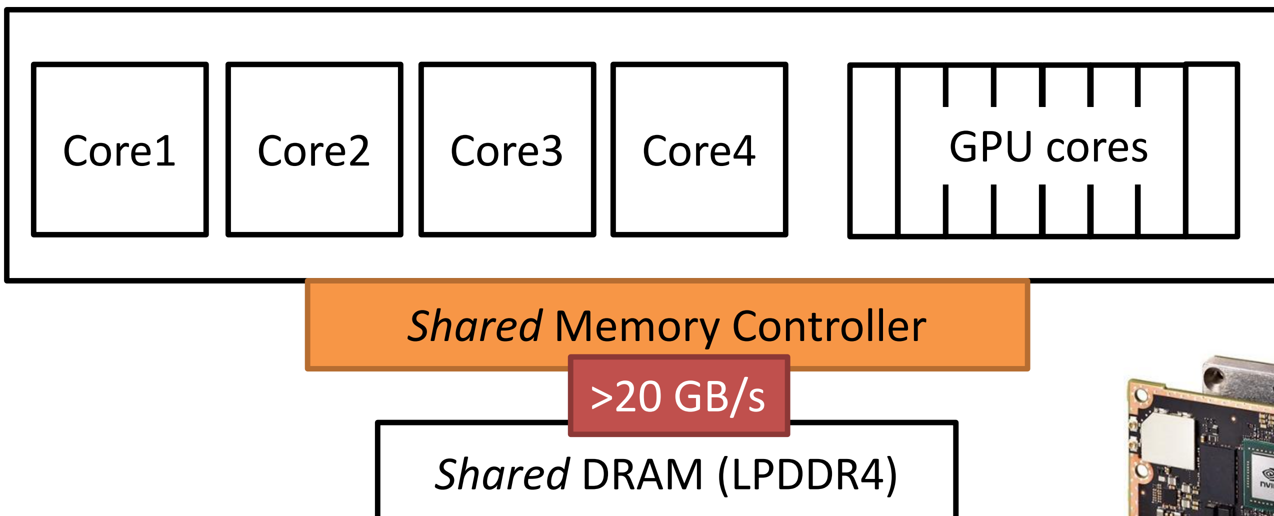
Discrete GPU



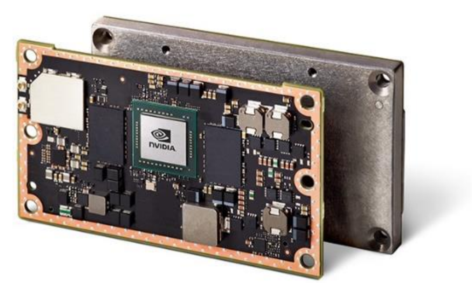
- GPU uses **dedicated** GPU memory
- Good for performance, but bad for cost & SWaP

Integrated CPU-GPU SoC

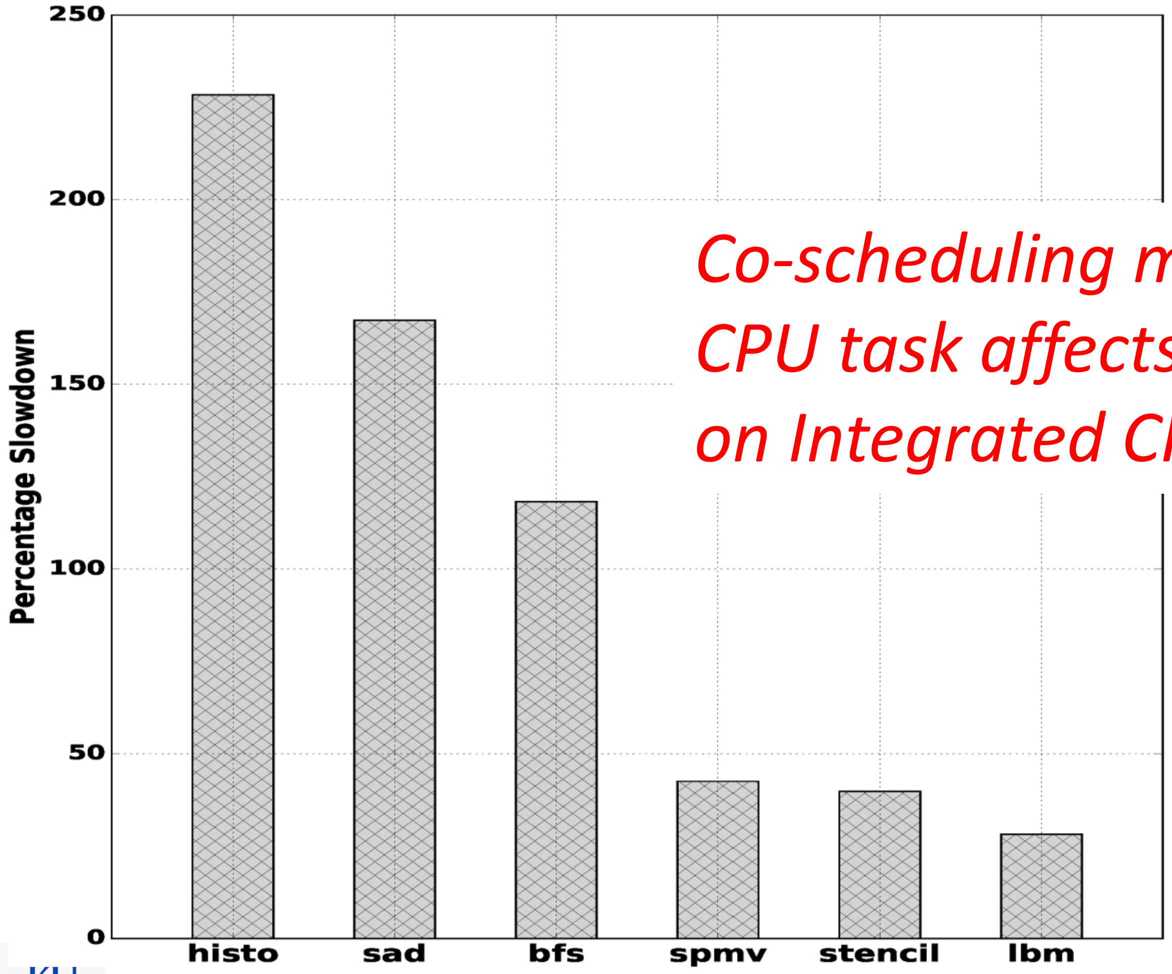
- CPU and GPU use the same **shared** DRAM
- Good for cost, SWaP, data movement, … *BUT*



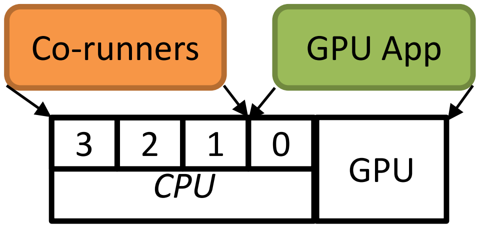
Nvidia Jetson TX2



Memory Bandwidth Contention



Co-scheduling memory intensive CPU task affects GPU performance on Integrated CPU-GPU SoC



CPU Memory Access Characteristic

- "*Low Latency (LL)* – the dominant characteristics of memory traffic coming from the CPUs are random, small size accesses (typically cache line fills) that are sporadic in nature. Key requirement for CPU accesses is low latency so as to provide maximum thread execution performance."

Ashwin Matta, "Optimizing Performance for an ARM Mobile Memory Subsystem." ARM White Paper, 2016

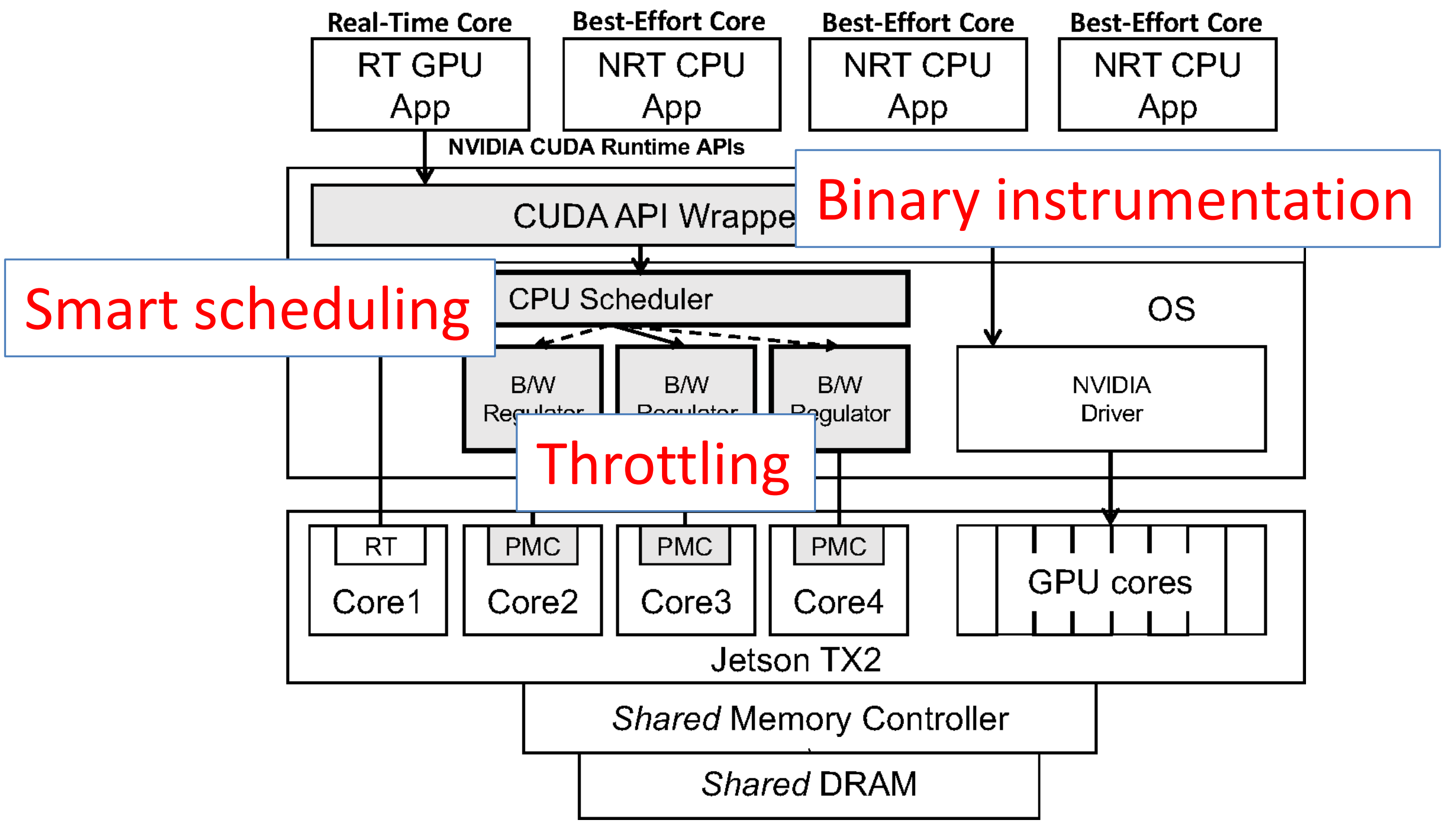
- Prioritizing CPU traffic over GPU is *usually* good, but **bad for real-time GPU kernels**

KU THE UNIVERSITY OF KANSAS

Outline

- Motivation
- **BWLOCK++**
  - **Memory bandwidth throttling**
  - **Binary instrumentation**
  - **Throttle fair scheduler (TFS)**
  - **Schedulability analysis**
- Evaluation
- Conclusion

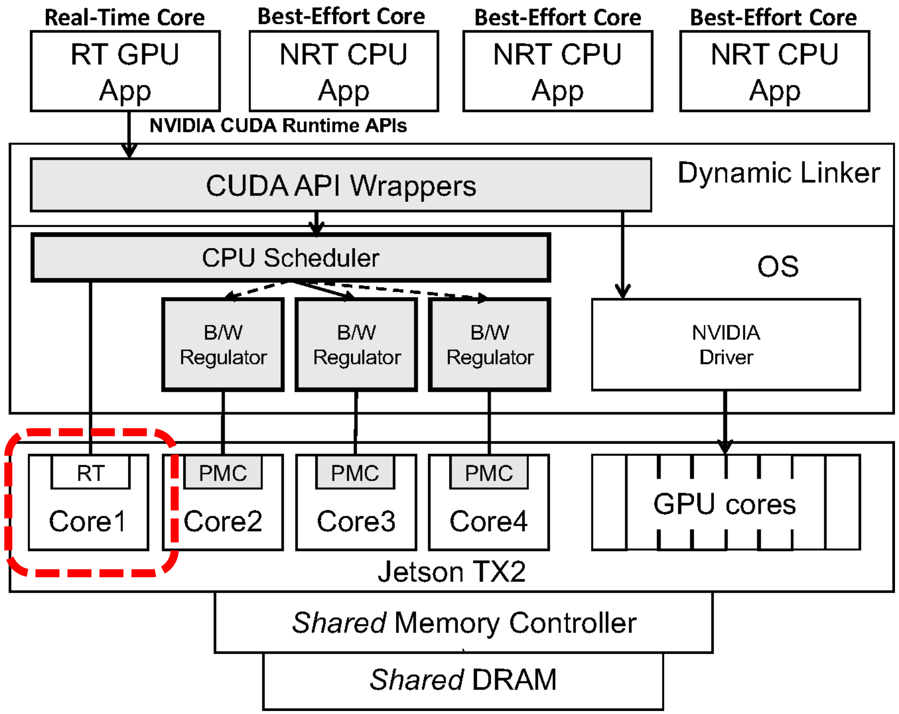
BWLOCK++



- Goal: **automatically** protect real-time GPU kernel while minimizing CPU **throughput** impact

Real-Time Core

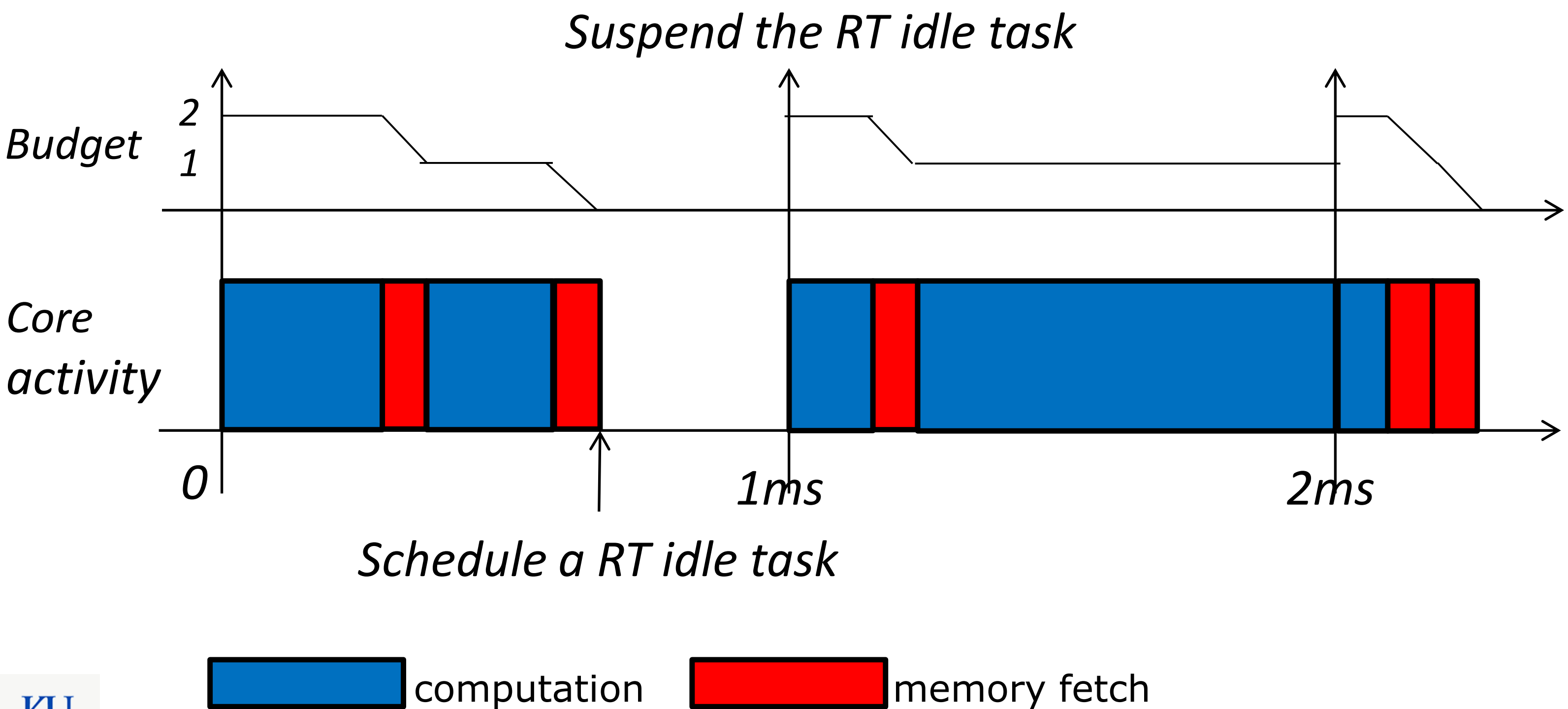
- Dedicated core to schedule ALL real-time tasks
  - – GPU kernels from diff tasks are *serialized** anyway



(*) They are time multiplexed rather than being truly concurrent. N. Otterness et al., "An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads." RTAS, 2017

Memory Bandwidth Throttling

- MemGuard*: Throttle CPU core's memory bandwidth using its performance counters



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

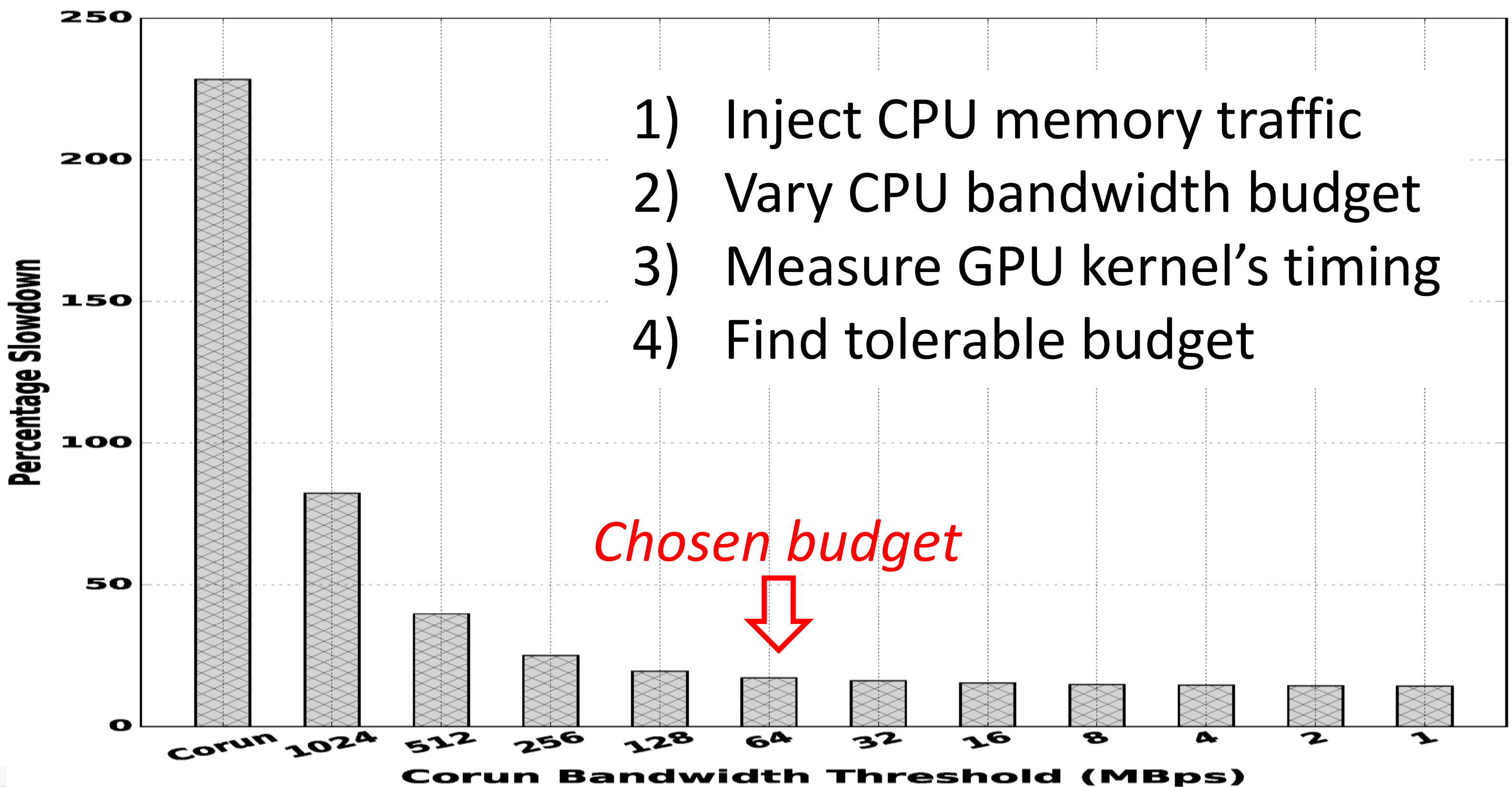
Real-Time GPU Kernel Protection

- Idea: *Throttle* CPU memory bandwidth usage *while* running real-time GPU kernels to protect their performance

- Questions
  - How much do we need to throttle?
  - When and how to start/stop throttling?
  - How to minimize CPU throughput loss?
  - How to analyze schedulability?

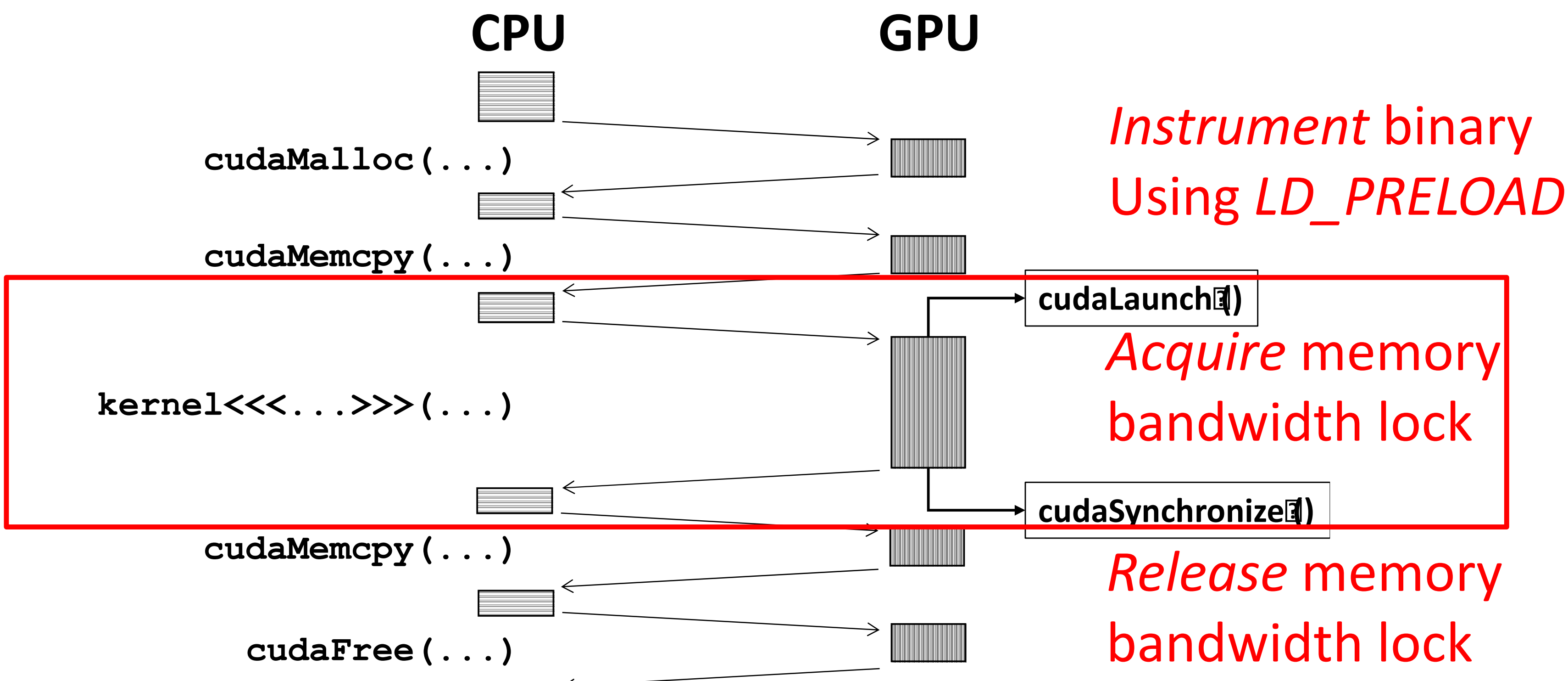
Determining Throttling Budget

- Based on each GPU task's bandwidth sensitivity



Dynamic Instrumentation

- Begin/stop throttling by instrumenting CUDA



No source code modification is needed

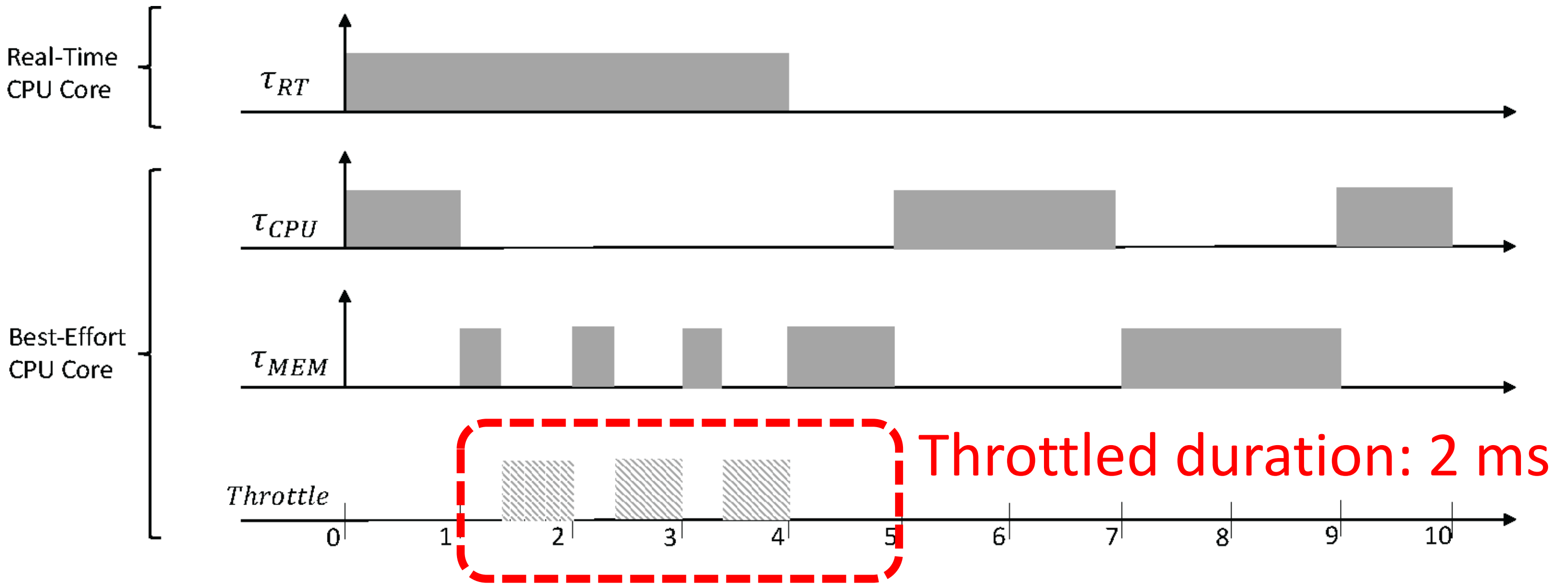
CPU Throttling and Scheduling

- Completely Fair Scheduler (CFS)
  - Linux's default scheduler (for non-real-time tasks)
  - Virtual runtime: weighted execution time
  - Pick the task with smallest virtual runtime

- *Destructive* interplay of throttling and CFS
  - More throttling → less virtual runtime increase
  - CFS prefers throttled tasks → more throttling

Example Schedule under CFS

- CFS preferred memory intensive task τ_{mem}



$V_{runtime}^{CPU}$	1	1	1	1	1	2	3	3	3	4
$V_{runtime}^{MEM}$	0	0.33	0.67	1	2	2	2	3	4	4

Throttle Fair Scheduler (TFS)

- Account throttled time in virtual runtime

Scale factor

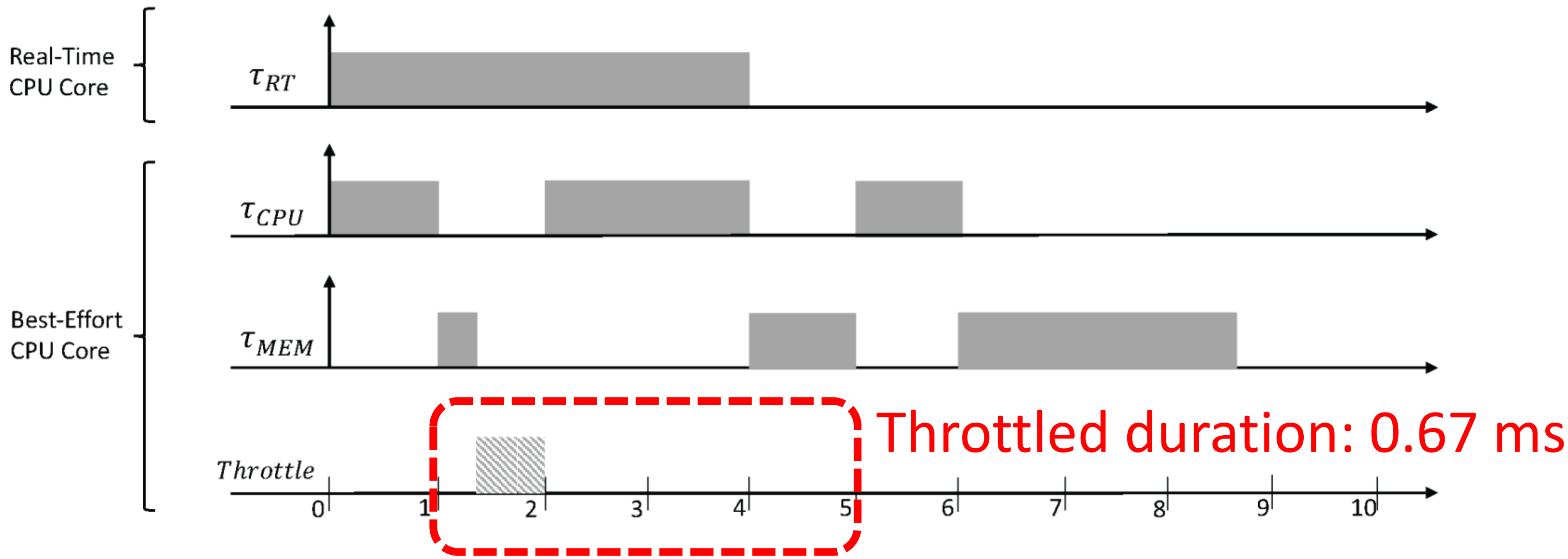
$$V_i^{new} = V_i^{old} + \delta_i^j \times \rho$$

Task's virtual runtime Throttled duration

- Effect
  - prefer more CPU intensive tasks
  - less CPU throttling
  - improved CPU throughput

Example Schedule under TFS

- TFS preferred CPU intensive task τ_{cpu}



$V^{CPU}_{runtime}$	1	1	2	3	3	4	4	4	4	
$V^{MEM}_{runtime}$ (TFS-3X)	0	2.34	2.34	2.34	3.34	3.34	4.34	5.34	6	
$V^{MEM}_{runtime}$ (Actual)	0	0.33	0.33	0.33	1.33	1.33	2.33	3.33	4	

Schedulability Analysis

- Classical RTA for preemptive fixed priority scheduling with blocking

Blocking time

$$R_i^{n+1} = E_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{P_j} \right\rceil E_j$$

Task execution time

- Treat GPU kernel execution as critical section
- Use priority ceiling protocol (PCP)

(*) N. Audsley et al., "Applying new scheduling theory to static priority preemptive scheduling." Software Engineering Journal, 1993

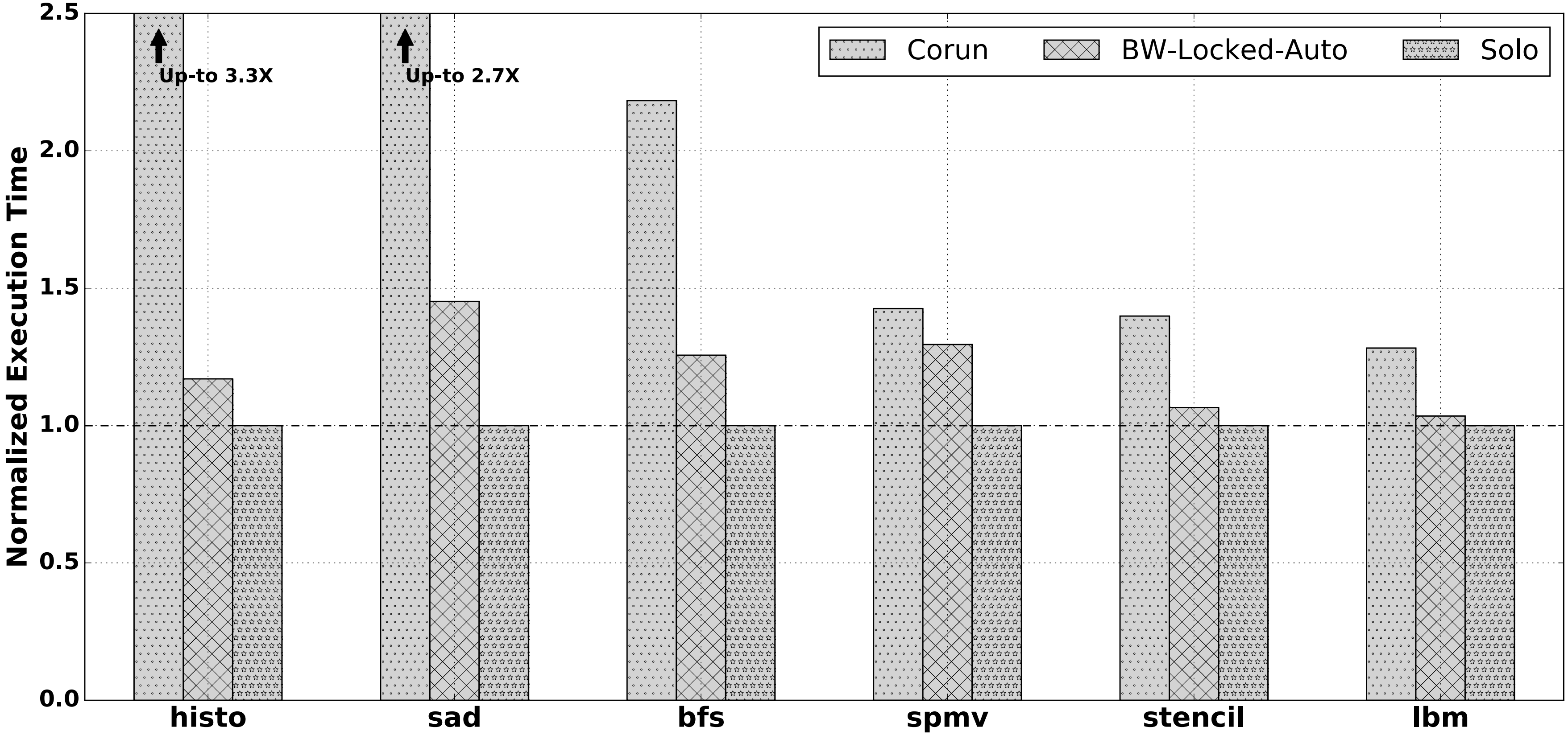
Outline

- Motivation
- BWLOCK++
  - Memory bandwidth throttling
  - Binary instrumentation
  - Throttle fair scheduler (TFS)
  - Schedulability analysis
- **Evaluation**
- Conclusion

Setup

- Hardware
  - Nvidia Jetson TX2
    - 4x Cortex-A57 (used) + 2x Denver (not used)
  - RT core: Core 0
- Software
  - Linux kernel 4.4.38 (+ TFS, BW regulator, …)
  - CUDA 8.0 + custom library (LD_PRELOAD)
- Benchmarks
  - Parboil benchmark suite (GPU tasks)
  - IsolBench benchmark suite (CPU tasks)

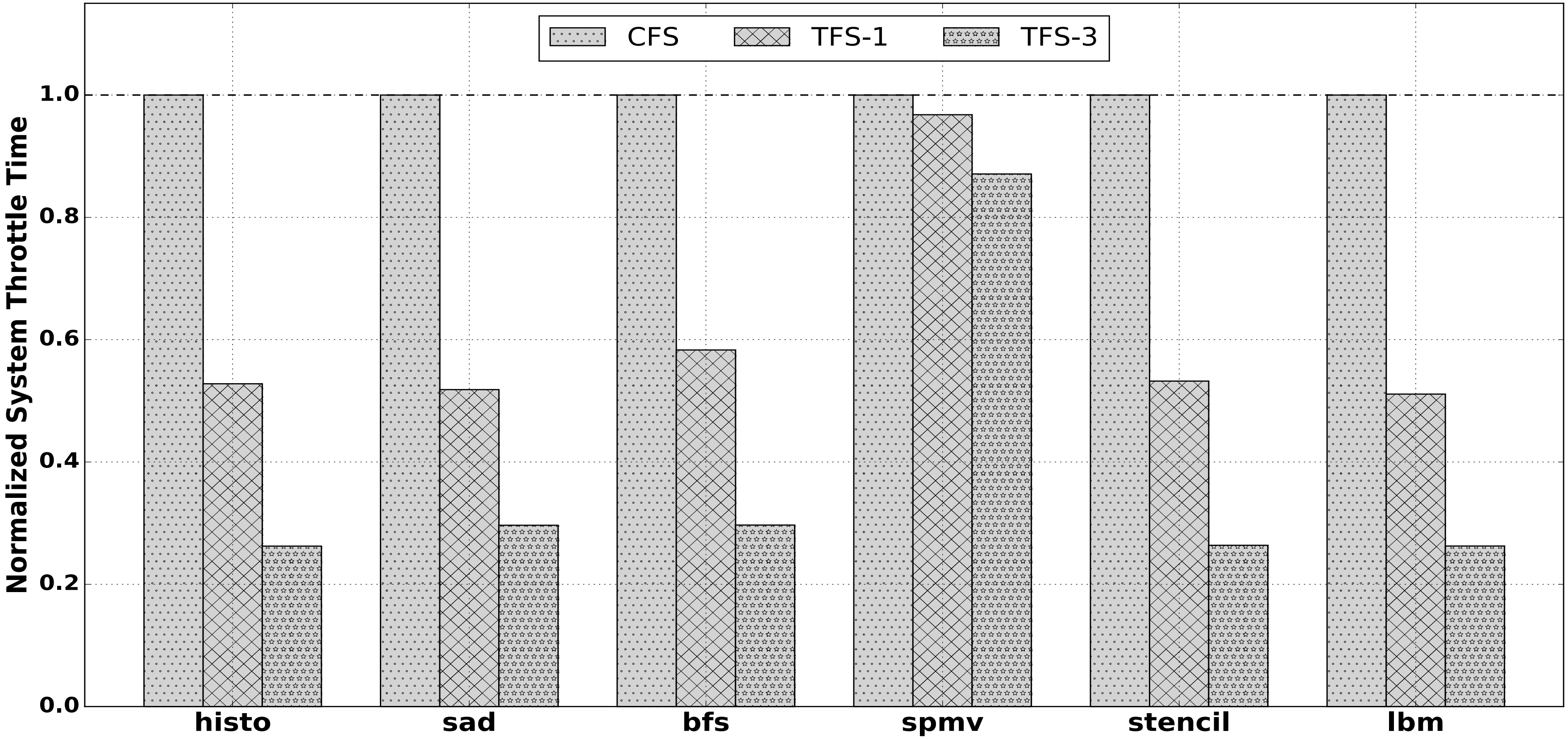
Real-Time Performance Impact



- Real-time GPU kernel performance is improved

The University of Kansas

CPU Throughput Impact



- TFS improves CPU throughput (reduce throttling)

Conclusion

- Integrated CPU-GPU SoC platforms
  - Good: performance, cost, size, weight, power
  - Bad: memory bandwidth contention
- BWLOCK++
  - **Automatically** and **efficiently** protect real-time GPU kernels on integrated CPU-GPU SoC
  - Throttling + runtime instrumentation + scheduling
  - **Practical** solution
- Availability
  - https://github.com/wali-ku/BWLOCK-GPU

Thank You

Disclaimer:
This research is supported by the National Science Foundation (NSF) under the grant numbers CNS 1718880, CNS 1815959