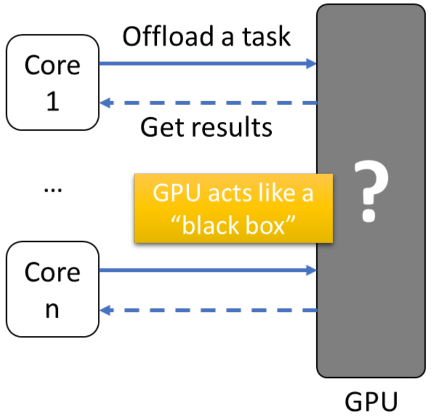
GCAPS: GPU Context-Aware Preemptive Priority-based Scheduling for Real-Time Tasks

Yidi Wang, Cong Liu, Daniel Wong, Hyoseung Kim

University of California, Riverside

Challenges

- ***Challenge 1***: Prior work models the GPU as a "black box"
  - "Once we offload task to GPU, it will run somehow"
  - No fine-grained control over GPU HW/SW



- ***Challenge 2***: Approaches for GPU tasks with End-to-End Timing Constraints

Vanilla GPU driver (Nvidia, AMD, etc.):
- Unpredictable GPU workload interleaving

➡ **No end-to-end guarantees**

Synch.-based approach (RT community):
- Run one task at a time on the GPU

➡ **Long waiting time**

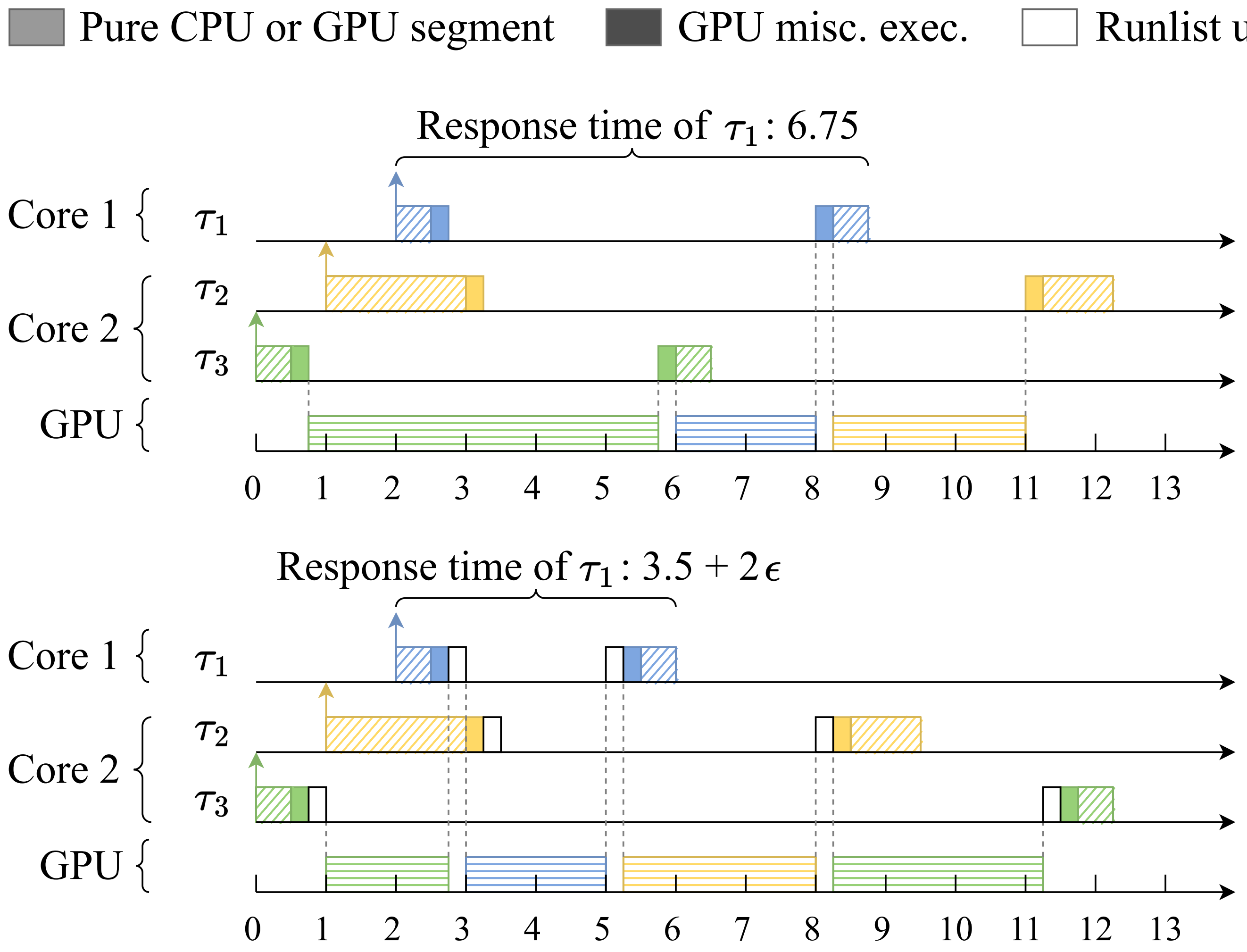
➡ Some efforts on ***preemptive GPU scheduling***
- ✗ Significant user program rewriting
- ✗ Neglected CPU – GPU interaction

Related Work

- Synchronization-based GPU access control (= non-preemptive)
  - GPU is modelled as critical sections [1][2] – Suffers from long blocking time
- Preemptive GPU scheduling
  - Decomposes big kernels into smaller segments [3][4] – Requires heavy code modifications
  - Hypervisor-based Preemptive GPU scheduling on VMs [5] – Lacking response time analysis
  - Microsecond-scale, Reset-based preemption [6] – not applicable to a wide range of apps
- GPU partitioning
  - Spatial partitioning of GPU in user-space [7][8] and driver [9] – Works within a single context
- GPU scheduling rules
  - Unveil GPU scheduling rules for safe GPU management [10] – Falls short in preemptive scheduling

[1] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," in Proceedings., 10th International Conference on Distributed Computing Systems. IEEE Computer Society, 1990, pp. 116–117.
[2] B. B. Brandenburg, "The FMLP+: An asymptotically optimal real-time locking protocol for suspension-aware analysis," in 2014 26th Euromicro Conference on Real-Time Systems. IEEE, 2014, pp. 61–71.
[3] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A responsive GPGPU execution model for runtime engines. *RTSS*, 2011
[4] H. Zhou, G. Tong, and C. Liu. GPES: a preemptive execution system for GPGPU computing. *RTAS*, 2015
[5] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for GPU with preemption support," in 2018 IEEE Real-Time Systems Symposium (RTSS). IEEE, 2018, pp. 119–130.
[6] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)
[7] S. K. Saha, Y. Xiang, and H. Kim. STGM: Spatio-temporal GPU management for real-time tasks. *RTCSA*, 2019
[8] Y. Wang, M. Karimi, Y. Xiang, and H. Kim, "Balancing energy efficiency and real-time performance in GPU scheduling," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021
[9] J. Bakita and J. H. Anderson, "Hardware Compute Partitioning on NVIDIA GPUs," in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2023.
[10] J. Bakita and J. H. Anderson, "Demystifying NVIDIA GPU Internals to Enable Reliable GPU Management", in ," in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2024.

Motivational Example



Sync.-based approach

GCAPS approach

No blocking time for high-priority task

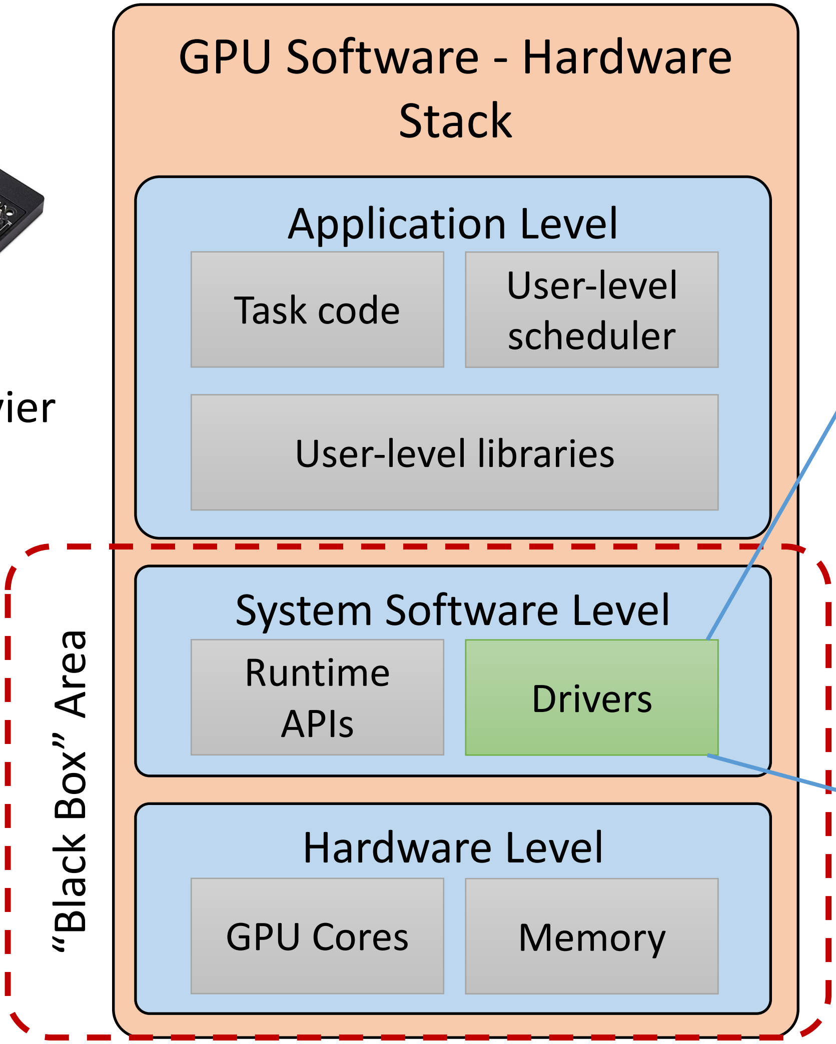
Our Contributions

		No blocking	Task priority respected	Analyzable response time	Inter-GPU context
Prior Work	Unmanaged GPU	╳	╳	✓	✓
	Sync.-based approaches	╳	✓	✓	✓
	GPU partitioning	✓	╳	✓	╳
	Preemptive GPU	✓	✓	╳	✓
Proposed	GCAPS	✓	✓	✓	✓

- **GCAPS: GPU Context-Aware Preemptive Priority-based Scheduling Approach for Real-Time Tasks**
  - A novel approach to manage GPU task preemption.
  - First work to provide response time analysis for preemptive GPU scheduling approach as well as the default GPU round-robin scheduling approach.

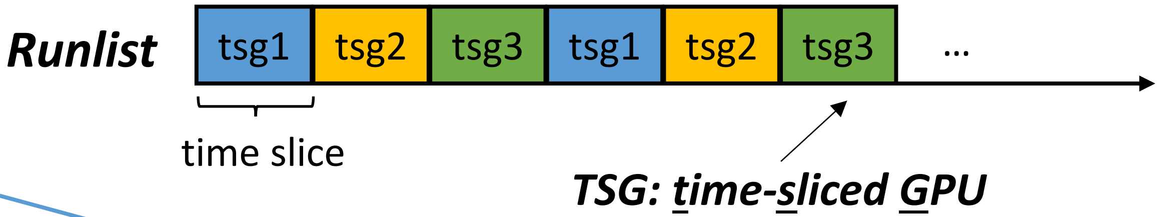
Unveiling the "Black Box": Driver

NVIDIA Jetson Xavier NX Dev Kit



- The default Tegra GPU driver
  - TSG: Time-Sliced Group
  - Runlist is populated with entries of TSGs

Employs a ***time-sliced round-robin*** scheduling approach, aiming at ***fairness***



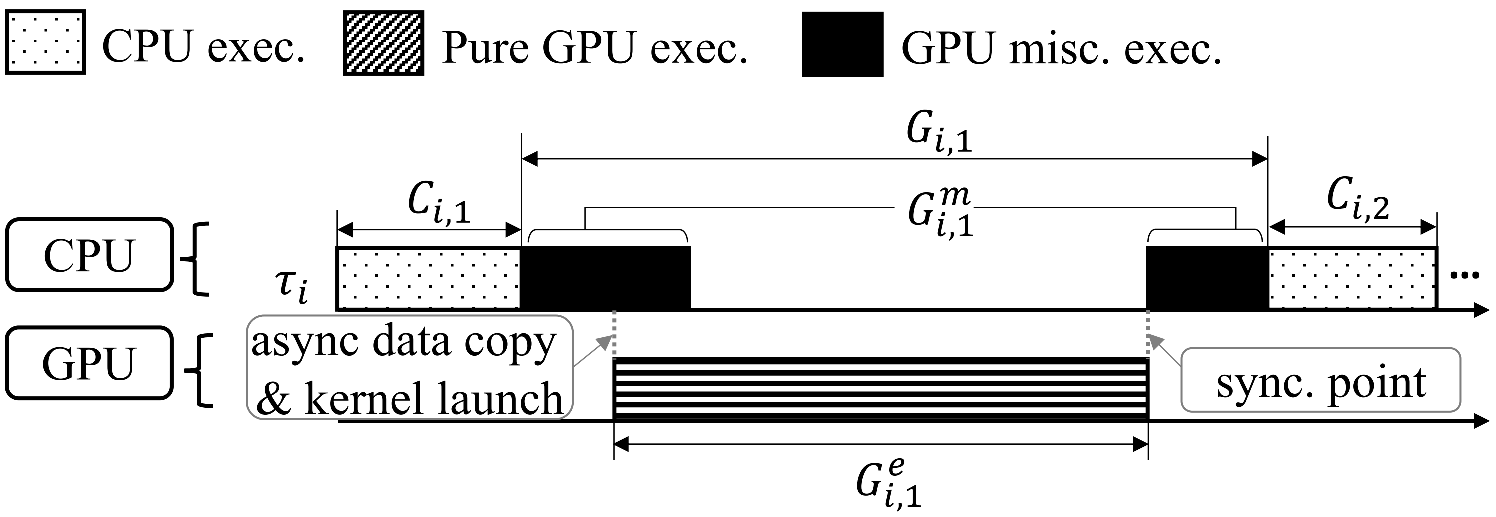
- ✗ No priority-respected
- ✗ No real-time responsiveness

System Model

- Partitioned multiprocessor scheduling
- The process (task) can either busy-wait or self-suspend during GPU execution
- Each task $\tau := (C_i, G_i, T_i, D_i, \eta_i^c, \eta_i^g)$

  CPU WCET   GPU WCET   Period <= Deadline   # of CPU segs   # of GPU segs
- j-th GPU segment $G_{i,j} := (G_{i,j}^m, G_{i,j}^e)$

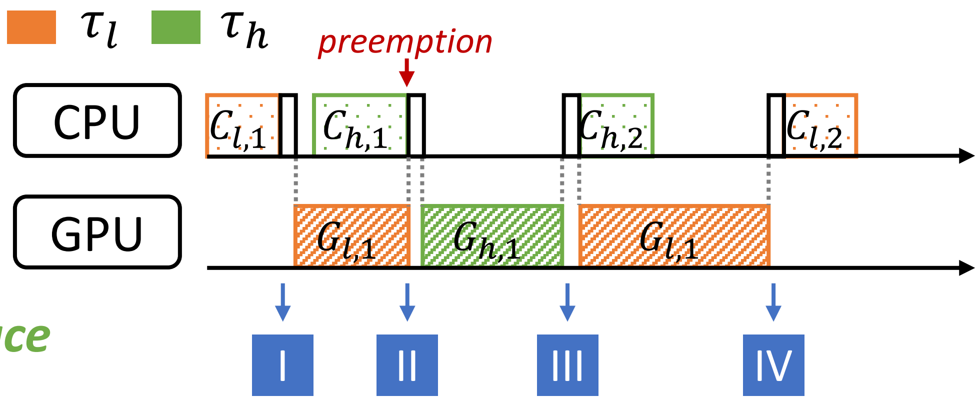
  Misc. GPU WCET   Pure GPU WCET



GCAPS Scheduler Overview

- Enabling preemption
  - Two User-level macros to mark the boundaries of GPU execution
  - IOCTL commands wrapped in the macros to update runlist

< 20 lines code for the macros in userspace




```
int task_function() {
    ...
    gcapsGpuSegBegin(fd, getpid());
    cudaMemcpyAsync(d_in, h_in, mem_size_in,
    cudaMemcpyHostToDevice, stream);
    MyKernel<<<grid, threads, 0, stream>>>(d_in,
    d_out);
    cudaMemcpyAsync(h_out, d_out, mem_size_in,
    cudaMemcpyHostToDevice, stream);
    gcapsGpuSegEnd(fd, getpid());
    ...
}
```

	CPU/GPU Interaction	RL (Runlist)			
I	$C_{l,1} \rightarrow G_{l,1}$: τ_l's TSG added to RL.	TSG_l	…		
II	$C_{h,1} \rightarrow G_{h,1}$: $C_{h,1}$ made "***add***" req. Since $\pi_h > \pi_l$, only τ_h is kept in RL.	TSG_h	…		
III	$G_{h,1} \rightarrow C_{h,2}$: $C_{h,2}$ made "***rm***" req. τ_l's TSG added back to RL.	TSG_l	…		
IV	$G_{l,1} \rightarrow C_{l,2}$: $G_{l,1}$ made "***rm***" req.				

Overhead

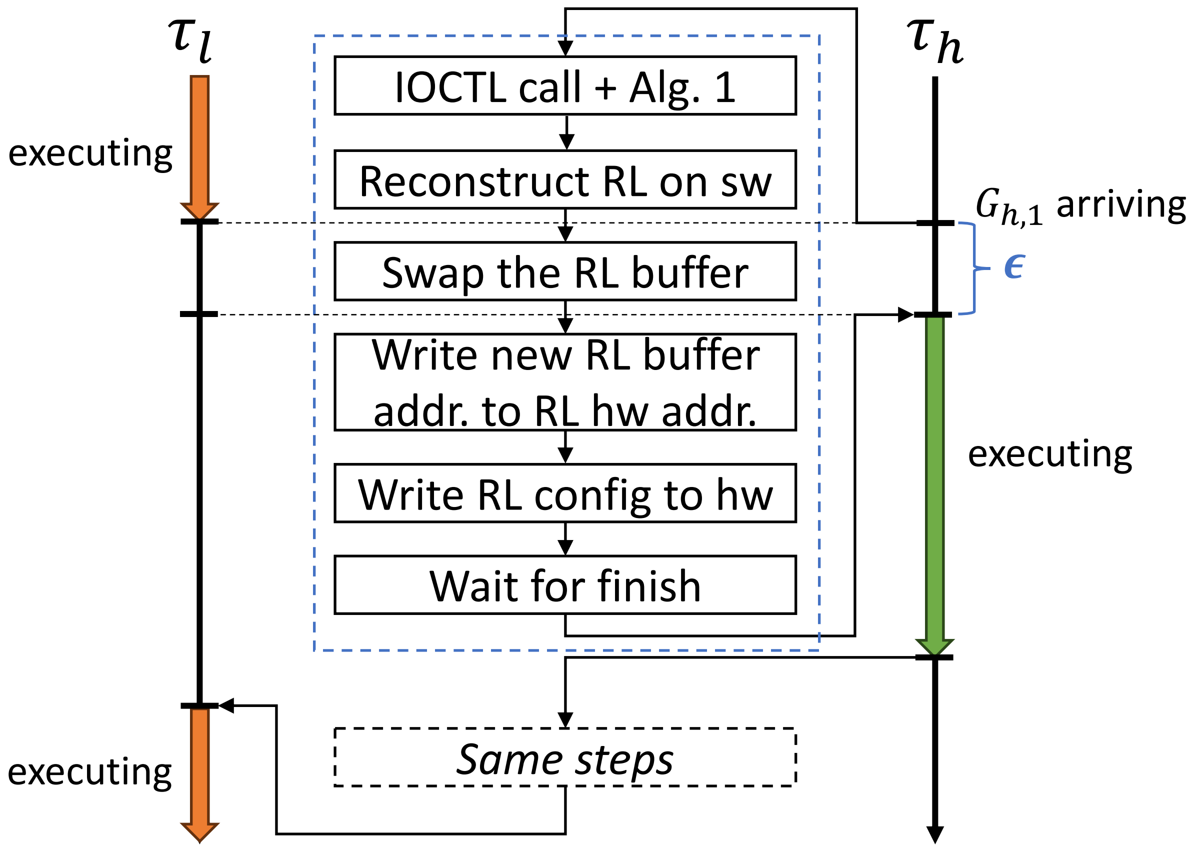
- **Definition 1 (*GPU context switch overhead*).** The GPU context switch overhead, $\boldsymbol{\theta}$, is the time required to switch from the GPU context of one process to that of another process.

➡ *This overhead is inherent to the default round-robin GPU scheduling approach.*

- **Definition 2 (*Runlist update delay*).** The runlist update delay, $\boldsymbol{\epsilon}$, is defined as the sum of the time it takes to complete our TSG scheduler (represented by α, including the cost for IOCTL system call, TSG scheduling algorithm, and runlist update) and the resulting GPU context switching overhead (θ). Hence, $\epsilon = \theta + \alpha$.

➡ *GCAPS introduces an extra overhead of α, to complete the proposed scheduler.*

GCAPS Context Switching Procedures



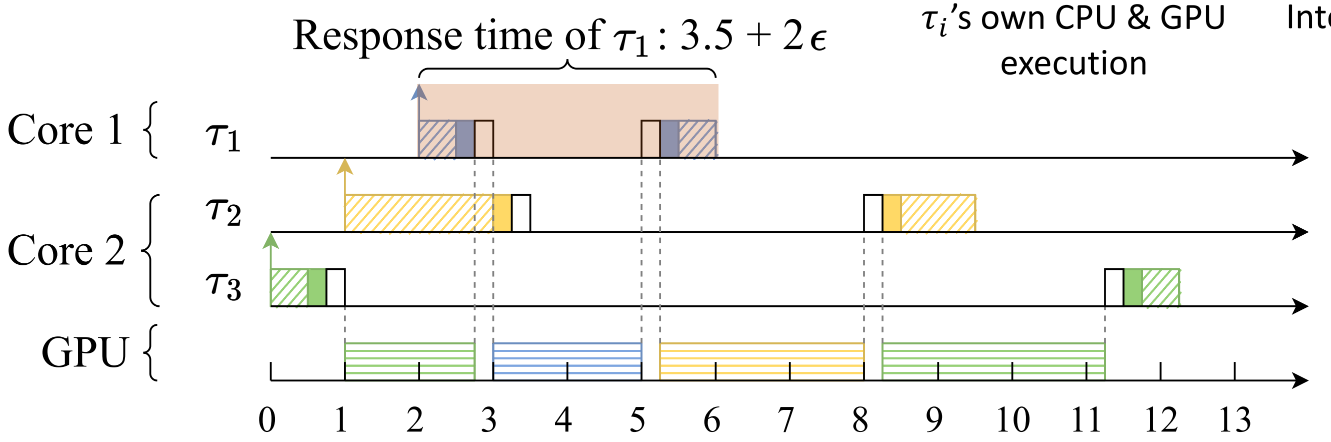
- GCAPS preemption is realized by reconstructing RL (runlist).
- The work of τ_l is preempted, not "aborted".
- τ_l resumes execution after τ_h yields the GPU.

Response Time Breakdown

Worst-case response time of a task τ_i: $R_i = C_i + G_i + I_i^C + I_i^G$

τ_i's own CPU & GPU execution

Interference from other tasks



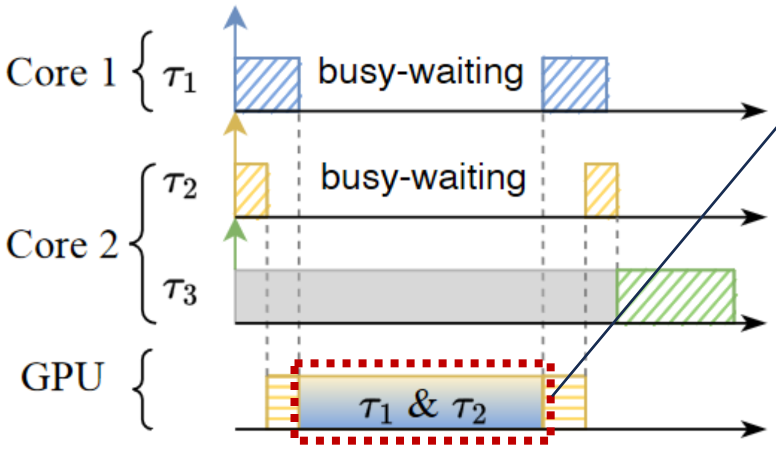
1. CPU Interference I_i^C

- Default RR & GCAPS: Preemption P_i^C
- GCAPS: Blocking B_i^C due to runlist update

2. GPU Interference I_i^G

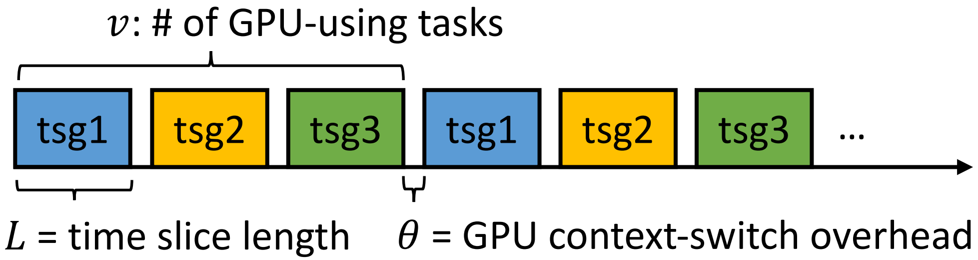
- Default RR: Interleaved execution I_i^{ie}
- GCAPS: Direct preemption I_i^{dp}
- Default RR & GCAPS: Indirect delay I_i^{id} due to busy-waiting

Analysis: Default RR Scheduling



GPU interleaved execution I_i^{ie}

Work-conserving Round-Robin



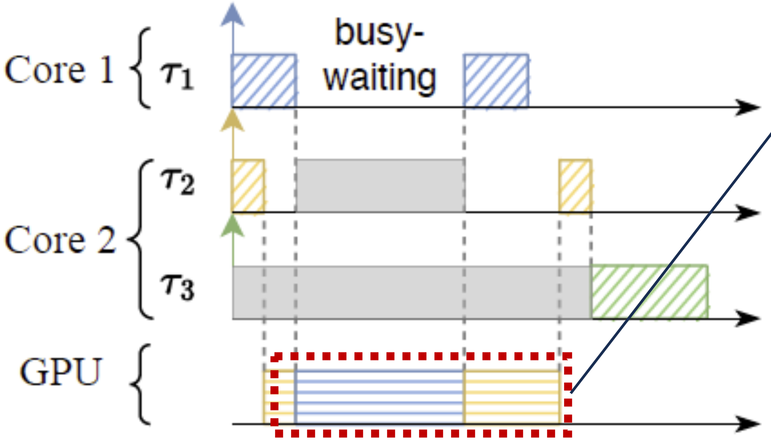
➡ $I_i^{ie} = \sum_{j=1}^{\eta_i^g} \mathcal{J}(|\{k|\tau_k \neq \tau_i \wedge \eta_k^g > 0\}|, G_{i,j}^e)$, where $\mathcal{J}(v, G_{i,j}^e) := (L + \theta) \cdot v \cdot \left\lceil \frac{G_{i,j}^e}{L} \right\rceil$

- Busy-waiting mode
  - GPU indirect delay $I_i^{id} = \sum_{\tau_h \in hpp(\tau_i) \wedge \eta_h^g > 0} \left\lceil \frac{R_i}{T_h} \right\rceil \cdot \sum_{j=1}^{\eta_h^g} (\mathcal{J}|\{k \mid \tau_k \neq hpp(\tau_i) \wedge \eta_k^g > 0 \cup \tau\_h\}|, G_{h,j}^e)$

    (extra delay imposed on CPU due to busy-waiting GPU segments)
- Self-suspension mode
  - GPU indirect delay $I_i^{id} = 0$

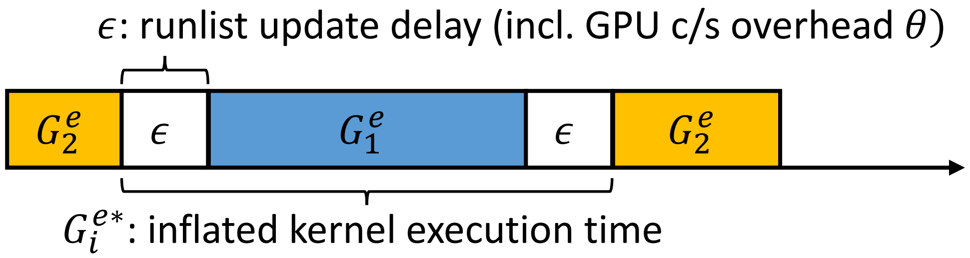
See the paper for details on other delay factors

Analysis: GCAPS



GPU direct preemption I_i^{dp}

Preemptive Priority-based



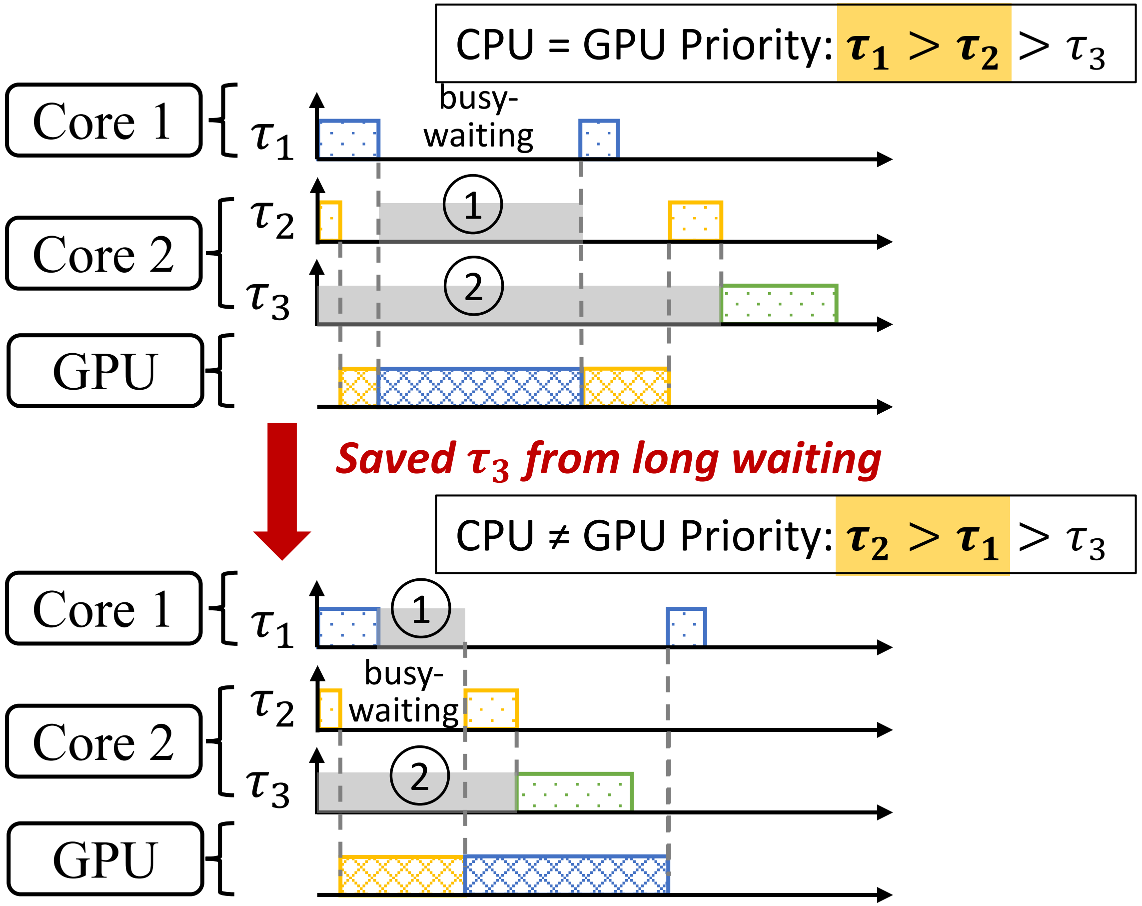
➡ $$I_i^{dp} = \sum_{\substack{\tau_h \in hpp(\tau_i) \\ \wedge \eta_h^g > 0 \wedge \eta_i^g > 0}} \left\lceil \frac{R_i}{T_h} \right\rceil \cdot G_h^{e*} + \sum_{\substack{\tau_h \in hp(\tau_i) \wedge \tau_h \notin hpp(\tau_i) \\ \wedge \eta_h^g > 0 \wedge \eta_i^g > 0}} \left\lceil \frac{R_i + J_h^g}{T_h} \right\rceil \cdot G_h^{e*}$$

HP tasks on the same core — HP tasks on different cores

- Busy-waiting mode
  - GPU indirect delay $I_i^{id} = \sum_{\tau_h \in hp(\tau_i) \wedge \tau_h \notin hpp(\tau_i) \wedge \eta_h^g > 0 \wedge \eta_i^g = 0} \left\lceil \frac{R_i + J_h^g}{T_h} \right\rceil \cdot G_h^{e*}$
- Self-suspension mode
  - GPU indirect delay $I_i^{id} = 0$

See the paper for details on other delay factors

Assigning Separate GPU Segment Priority



- Assigning separate priority to the GPU segment of a task, different from its OS-level priority to improve schedulability.
- We adopt Audsley's approach:
  - If a taskset is not schedulable, we iterate through the CPU priorities from low to high to see whether it can be assigned to the GPU segments of a task.
- To avoid deadlock:
  - We maintain the relative priority order of GPU segments identical to their OS-level priority.

Evaluation

- Experiment setup
  - NVIDIA Jetson Xavier NX running L4T R35.2.1 with Jetpack 5.0.2
- Scheduling approaches
  - Proposed preemptive approach: GCAPS (suspend, busy)
  - Default GPU Round-Robin approach (suspend, busy)
  - Synchronization-based approach:
    - FMLP+[1] (suspend, busy)
    - MPCP[2] (suspend, busy)
- Tasks

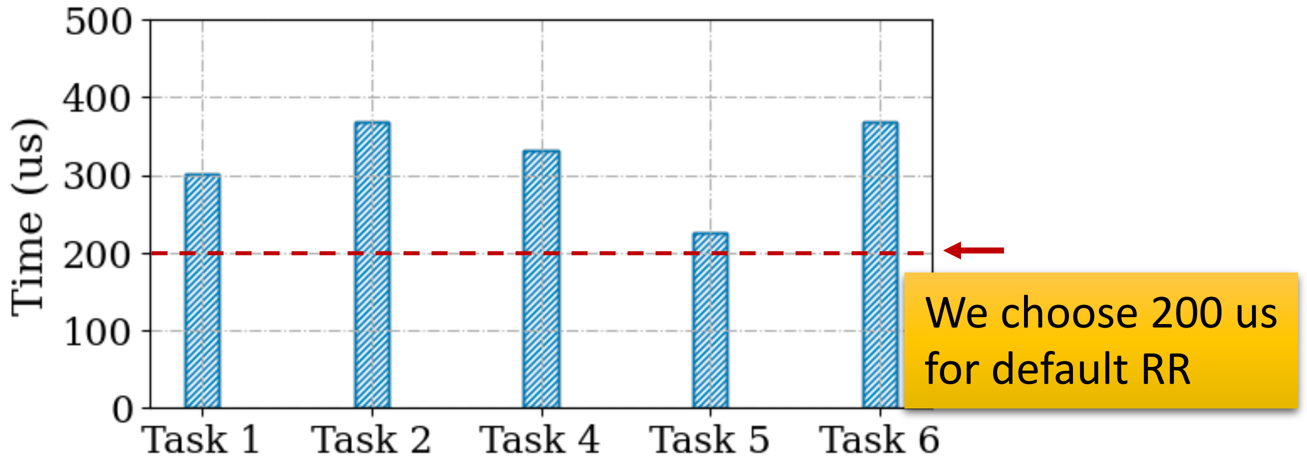
Task	Workload
1	histogram
2	mmul_gpu_1
3	mmul_cpu
4	projection
5	dxtc
6	mmul_gpu_2
7	simpleTexture3D (graphic app)

[1] Björn B Brandenburg. The FMLP+: An asymptotically optimal real-time locking protocol for suspension-aware analysis. In 2014 26th Euromicro Conference on Real-Time Systems, pages 61–71. IEEE, 2014.
[2] Pratyush Patel, Iljoo Baek, Hyoseung Kim, and Ragunathan Rajkumar. Analytical enhancements and practical insights for MPCP with self-suspensions. In IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2018.

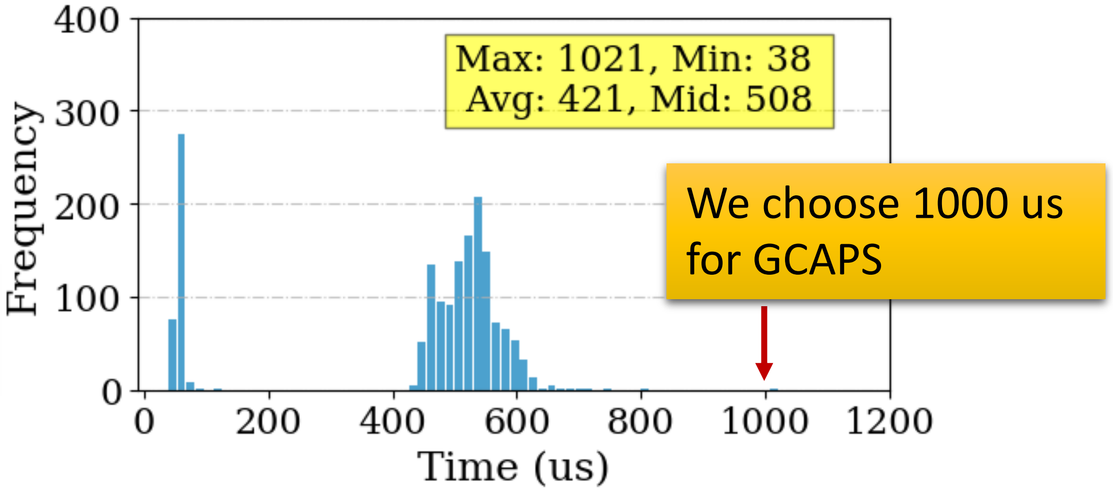
Experimental Results: Real System (1)

- Overhead Measurement

Overhead per TSG context switching (θ)



Distribution of overhead per runlist update (ϵ)



> 200 us

Not trivial, especially for long-running tasks

Acceptable;

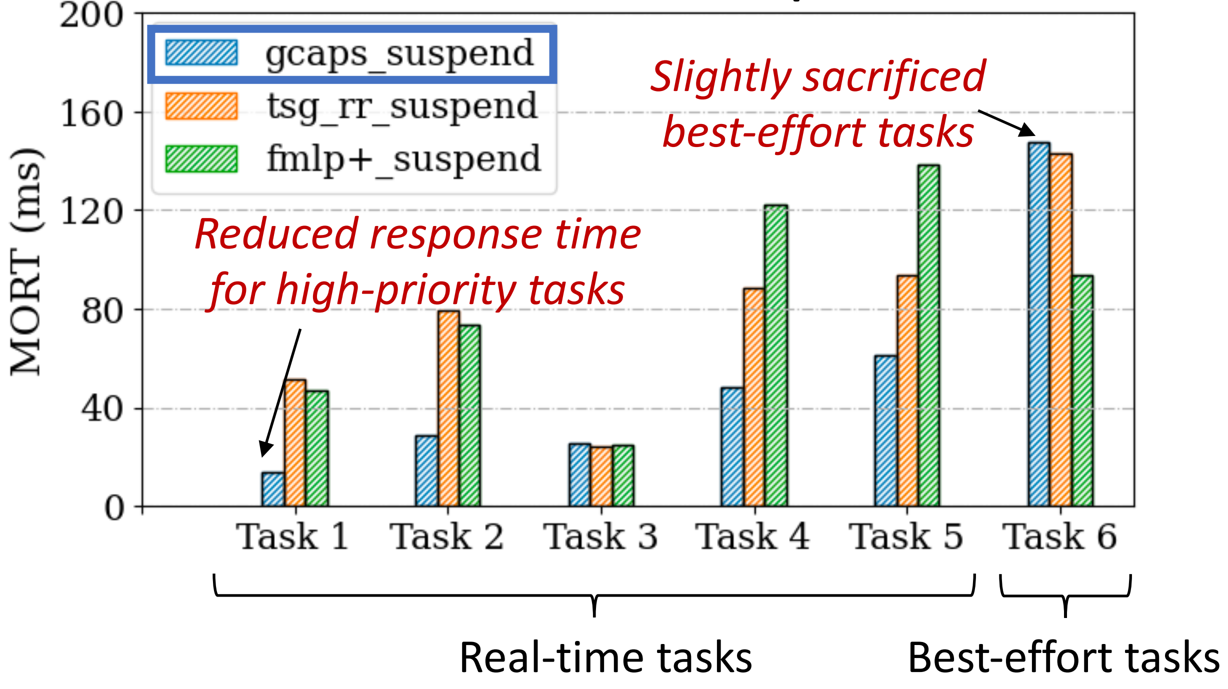
GCAPS schedulability still outperforms

- Parameter selection in evaluations:
  - Runlist Update Overhead θ: 1 ms
  - TSG context switching overhead ϵ: 200 us

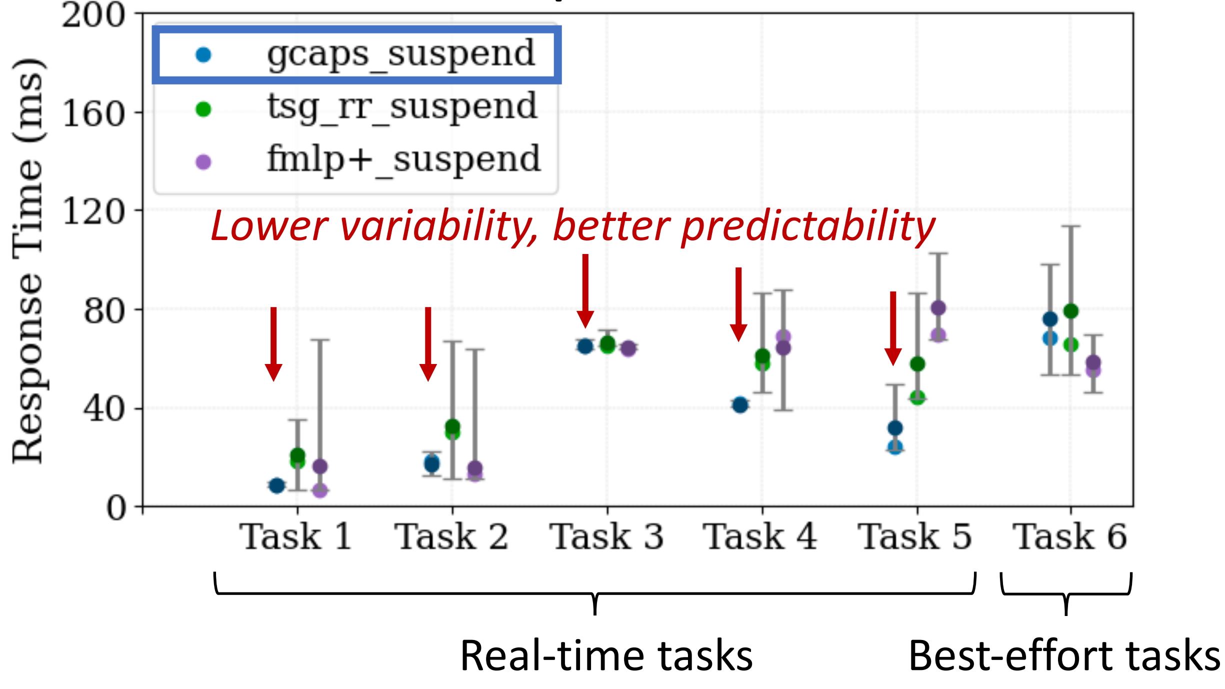
Experimental Results: Real System (2)

- Response time comparison
  - Self-suspension mode as an example
  - Benchmarks from CUDA Sample and Rodinia

Maximum observed response time



Measured response time distribution



Experimental Results: Real System (3)

- Effectiveness of the proposed analysis
  - Comparison of MORT (ms) and WCRT (ms)

Task	tsg_rr_suspend		tsg_rr_busy		gcaps_suspend		gcaps_busy	
	MORT	WCRT	MORT	WCRT	MORT	WCRT	MORT	WCRT
1	45.33	60	26.13	60	10.15	16	9.68	16
2	66.97	73.6	44.47	73.6	22.36	32	23.28	32
3	71.84	76	109.14	129.2	67.39	75	85.01	111
4	86.50	98.2	75.64	192.2	43.17	59	44.91	59
5	86.62	127.8	117.68	Failed	49.24	79	57.93	79

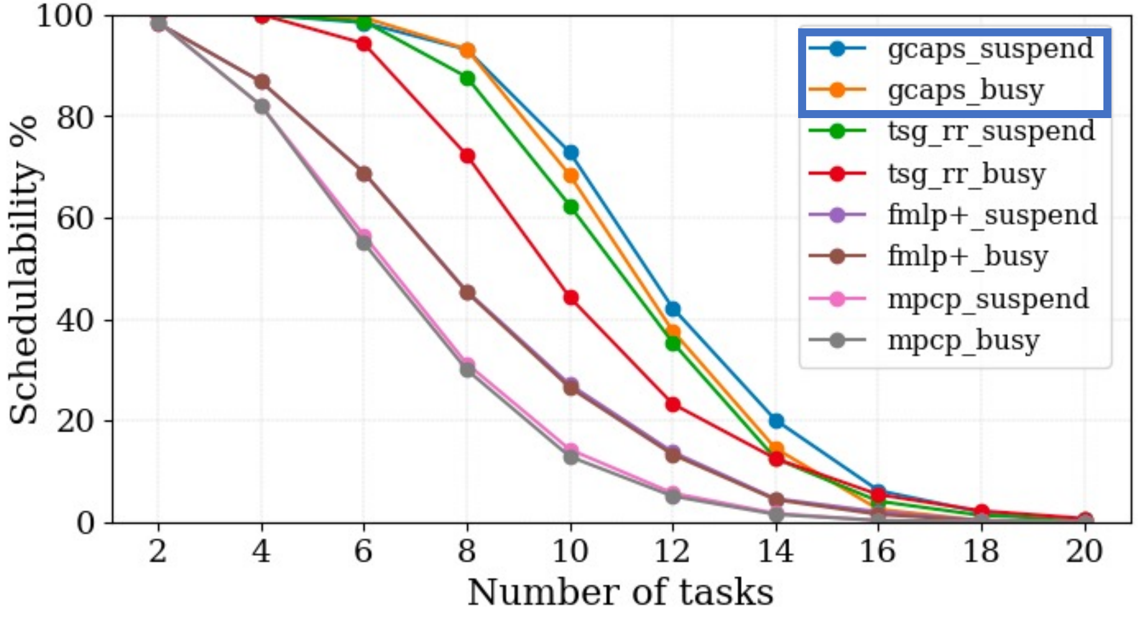
All WCRT >= MORT

- **First work** to bound response time for preemptive GPU tasks and the default GPU round-robin scheduling approach.

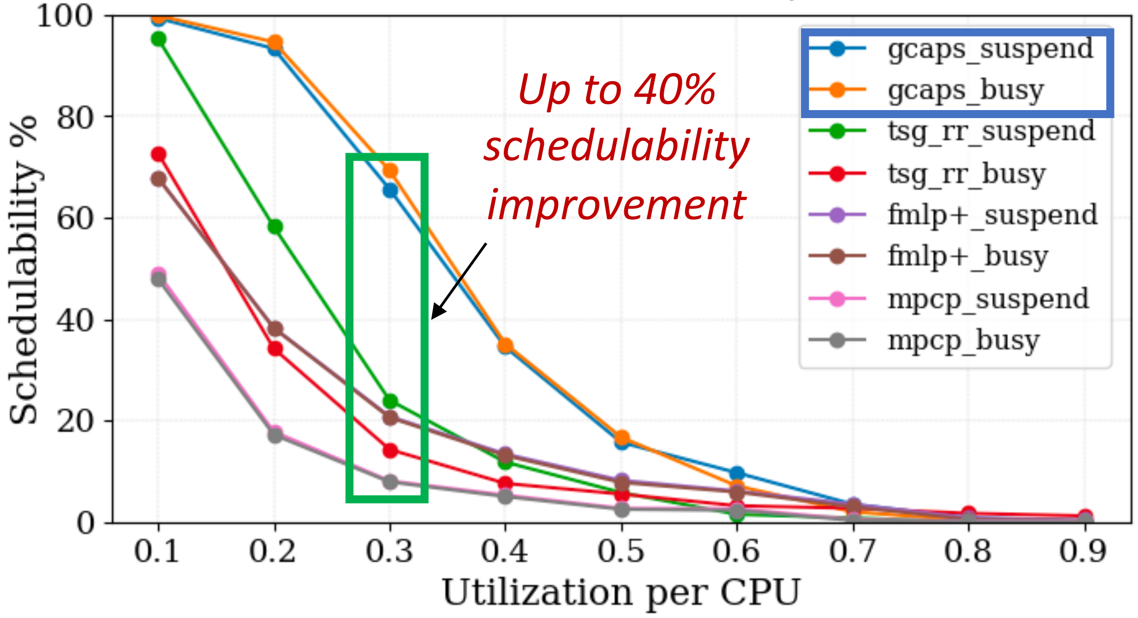
Experimental Results: Schedulability (1)

- We generated 1000 tasksets for each setting
- Taskset parameters are adopted from [1] with slight changes

Variable number of tasks



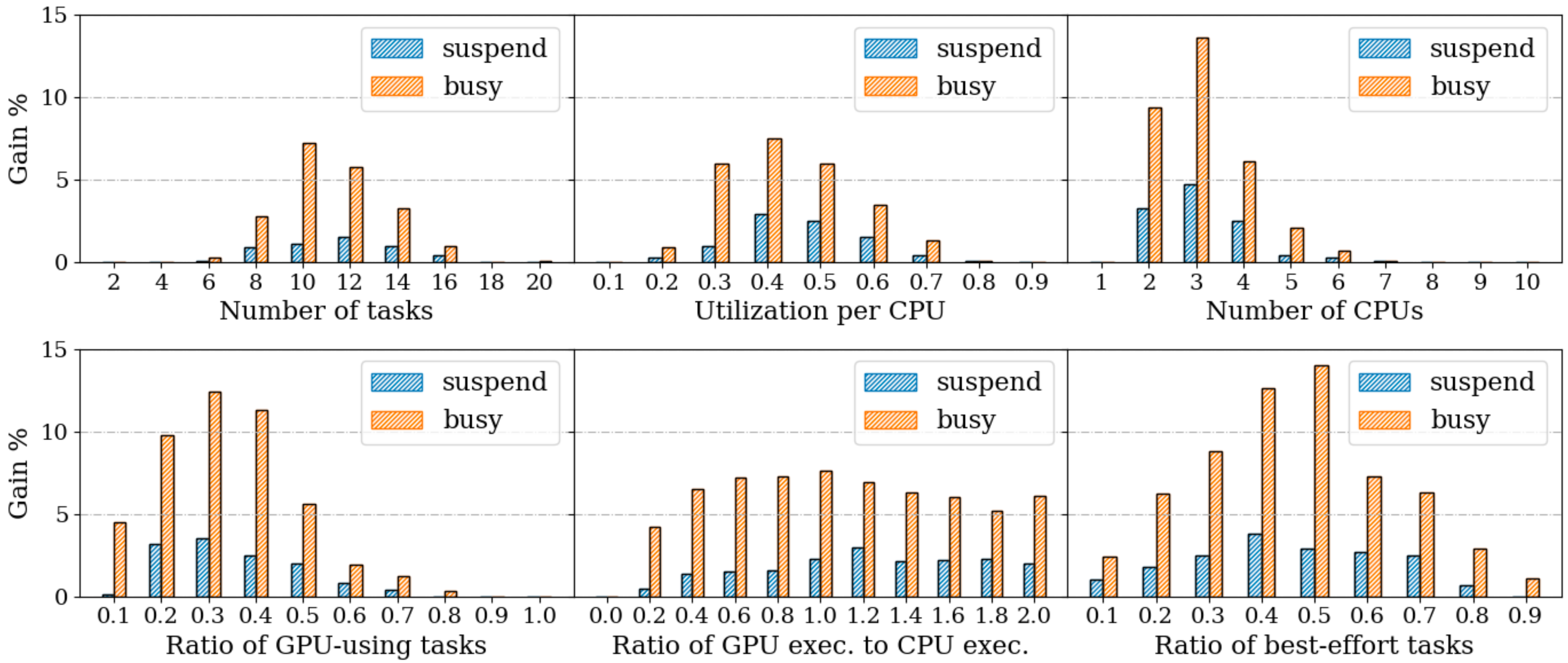
Variable task utilization per CPU



[1] P. Patel, I. Baek, H. Kim, and R. Rajkumar, "Analytical enhancements and practical insights for MPCP with self-suspensions," in IEEE Real- Time and Embedded Technology and Applications Symposium (RTAS), 2018.

Experimental Results: Schedulability (2)

- Effect of separate GPU segment priority assignment
- Compare baseline analysis of GCAPS with and without separate GPU priorities



This improvement effectively increases schedulability.

Summary

- **GCAPS: GPU Context-Aware Preemptive Priority-based Scheduling Approach for Real-Time Tasks**
  - A novel approach to manage GPU task preemption.
  - First work to bound response time for preemptive GPU scheduling approach as well as the default GPU round-robin scheduling approach.
  - Experiments show the effectiveness of our approach in predictability and responsiveness over the default driver and prior works.

  - Our work is open-source at: https://github.com/rtenlab/gcaps-super-repo

Thank You

GCAPS: GPU Context-Aware Preemptive Priority-based Scheduling for Real-Time Tasks

Yidi Wang, Cong Liu, Daniel Wong, Hyoseung Kim

University of California, Riverside