A Server-based Approach for Predictable GPU Access Control

Hyoseung Kim[*] Pratyush Patel[†] Shige Wang[‡] Raj Rajkumar[†]

[*] University of California, Riverside
[†] Carnegie Mellon University
[‡] General Motors R&D

Benefits of GPUs

- **High computational demands** of recent safety-critical systems
  - Long execution times → Hard to meet their deadlines
- **General-Purpose Graphics Processing Units (GPUs)**
  - 4-20x faster than a CPU for data-parallel, compute-intensive workloads*
  - Many embedded multi-core processors have **on-chip GPUs**

NVIDIA TK1/TK2

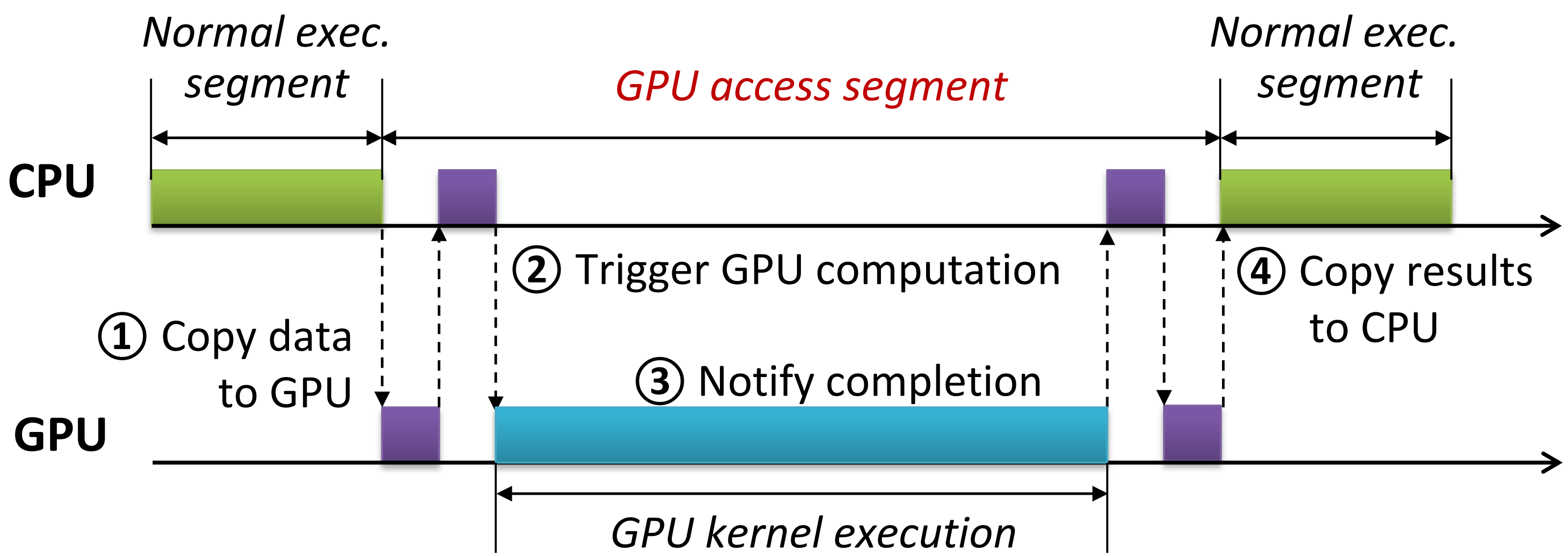
NXP i.MX6

Samsung Exynos 9

* Govindaraju et al. High Performance Discrete Fourier Transforms on Graphics Processors. ACM/IEEE conference on Supercomputing (SC), 2008.

GPU Execution Pattern

- **Task accessing a GPU**



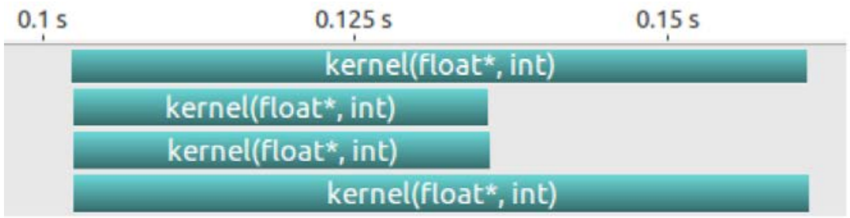
Need for predictable GPU access control

- To bound and minimize GPU access time
- To achieve better task schedulability

Many of Today's COTS GPUs

1. **Do not support preemption**
   - Due to the high overhead expected on GPU context switching*
   - Some recent GPU architectures support preemption (e.g., NVIDIA Pascal)

2. **Handle GPU requests in a sequential manner**
   - Concurrent execution of GPU kernels may result in unpredictable delay



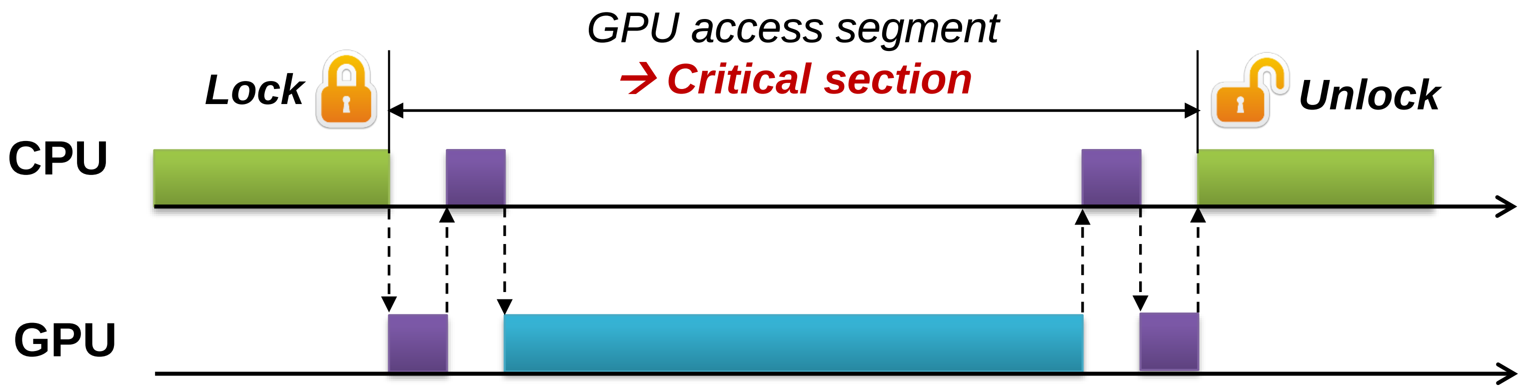
Four same kernels on NVIDIA GTX 980
- **97% slowdown** on two kernels
- Unpredictable who gets the delay

3. **Do not respect task priorities and the scheduling policy used**
   - May result in unbounded priority inversion

* I. Tanasic et al. Enabling preemptive multiprogramming on GPUs. In *International Symposium on Computer Architecture (ISCA)*, 2014.

Prior Approach

- **Synchronization-based approach*†‡**
  - Models each GPU access segment as a critical section
  - Uses a real-time synchronization protocol to handle GPU requests



Benefits
- Does not require any change in GPU device drivers
- Existing schedulability analyses can be directly reused

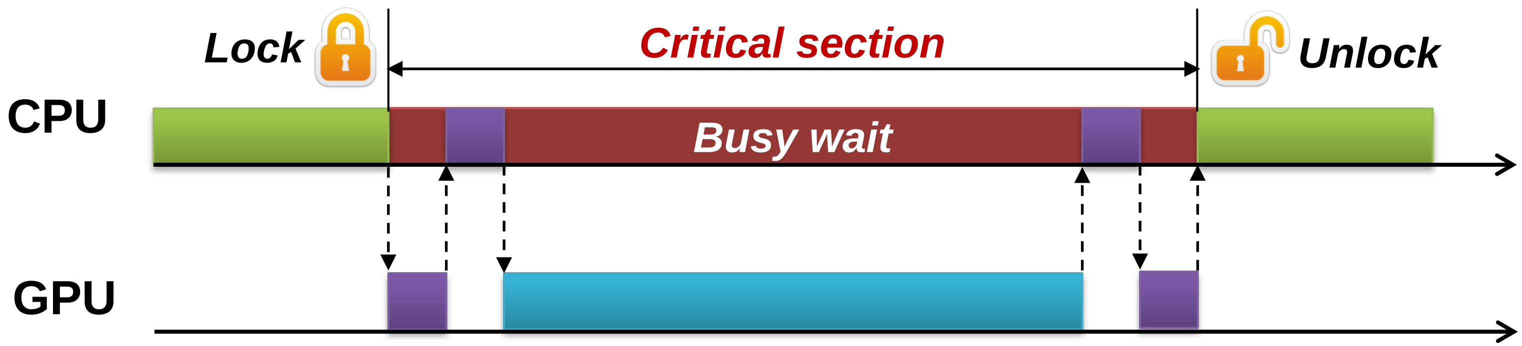
* G. Elliott and J. Anderson. Globally scheduled real-time multiprocessor systems with GPUs. *Real-Time Syst.*, 48(1):34–74, 2012.
† G. Elliott and J. Anderson. An optimal k-exclusion real-time locking protocol motivated by multi-GPU systems. *Real-Time Syst.*, 49(2):140–170, 2013.
‡ G. Elliott et al. GPUSync: A framework for real-time GPU management. In *IEEE Real-Time Systems Symposium (RTSS)*, 2013.

Limitations

Common assumptions of most RT synch. protocols, e.g., MPCP, FMLP†, OMLP‡*

1. Busy waiting

- Critical sections are executed entirely on the CPU
- No suspension during the execution of a critical section



2. Long priority inversion

- High priority tasks may suffer from unnecessarily long priority inversion
- Due to priority boosting used by some protocols, e.g., MPCP and FMLP

* R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, 1988.
† A. Block et al. A flexible real-time locking protocol for multiprocessors. In *IEEE Embedded and Real-Time Comp. Systems and Apps., (RTCSA)*, 2007.
‡ B. Brandenburg and J. Anderson. The OMLP family of optimal multiprocessor real-time locking protocols. *Design Automation for Embedded Systems*, 17(2):277–342, 2013.

Our Contributions

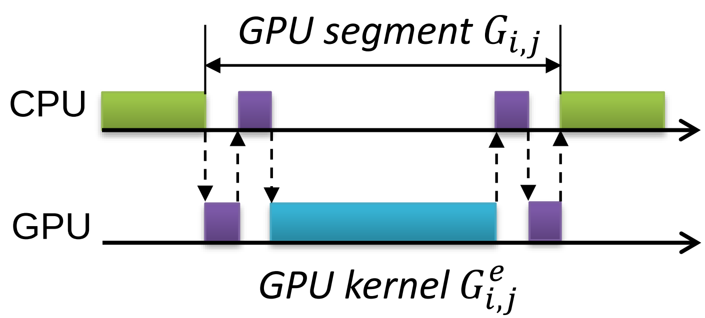
- **Server-based approach** for predictable GPU access control
- Addresses the limitations of the synchronization-based approach
  - Yields CPU utilization benefits
  - Reduces task response time
- Prototype implementation on an NXP i.MX6 running Linux
- Can be used for other types of computational accelerators, such as a digital signal processor (DSP)

Outline

- Introduction and motivation
- Server-based approach for predictable GPU access control
  - System model
  - GPU server and analysis
  - Comparison with the synchronization-based approach
- Evaluation
- Conclusions

System Model

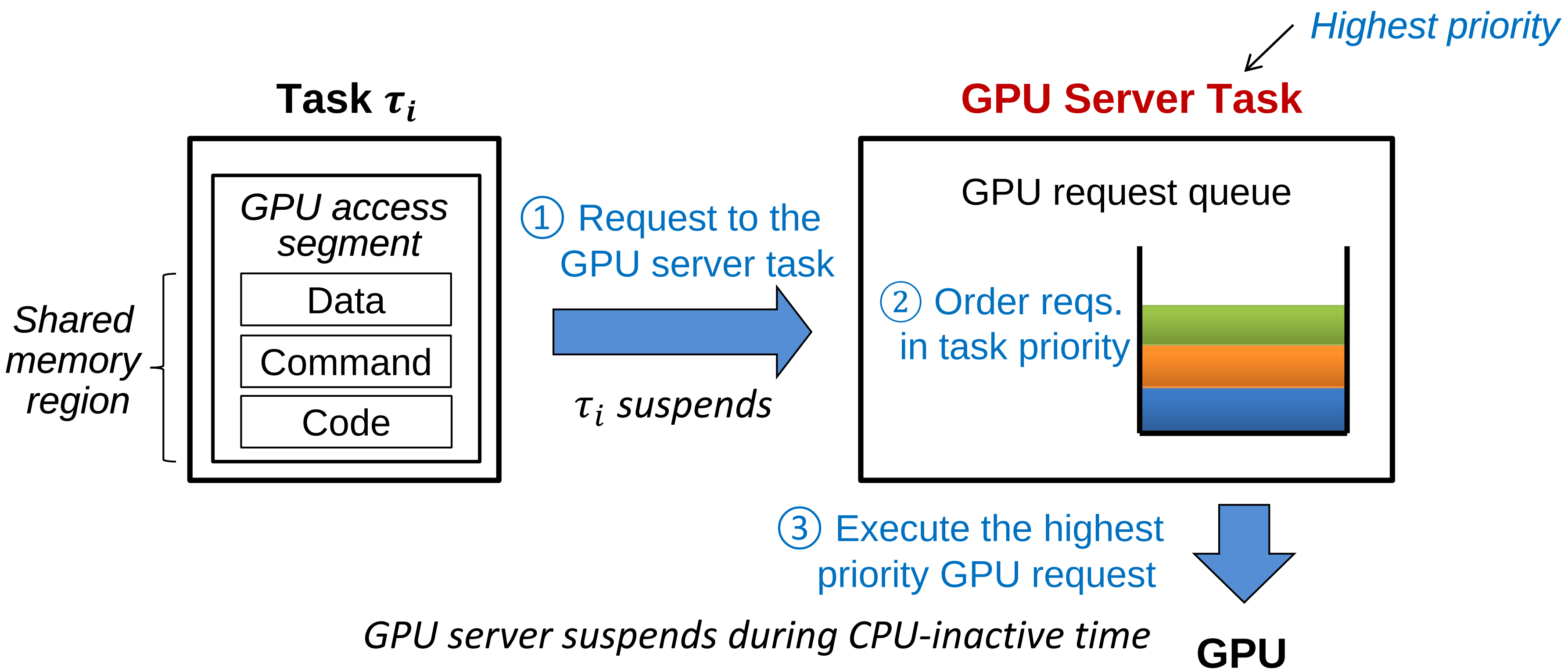
- Single general-purpose GPU device
  - Shared by tasks in a sequential, non-preemptive manner
- Sporadic tasks with constrained deadlines
  - Task $\tau_i := (C_i, T_i, D_i, G_i, \eta_i)$
    - C_i: Sum of the WCET of all normal execution segments
    - T_i: Minimum inter-arrival time
    - D_i: Relative deadline
    - G_i: Max. accum. length of all GPU segments
    - η_i: Number of GPU access segments
  - GPU segment $G_{i,j} := (G_{i,j}^e, G_{i,j}^m)$



- Partitioned fixed-priority preemptive task scheduling

Server-based Approach

- **GPU server task**
  - Handles GPU access requests from other tasks on their behalf
  - Allows tasks to suspend whenever no CPU intervention is needed



Timing Behavior of GPU Server

- GPU server overhead ϵ
  - Receiving a request and waking up the GPU server
  - Checking the request queue
  - Notifying the completion of the request

- Maximum handling time of all GPU requests of τ_i by GPU server

Total waiting time *GPU segment length* *Server overhead (twice per segment)*

$$B_i^{gpu} = \begin{cases} \boxed{B_i^w} + \boxed{G_i} + \boxed{2\eta_i\epsilon} & : \eta_i > 0 \\ 0 & : \eta_i = 0 \end{cases}$$

Task Response Time with GPU Server

- Case 1: a task τ_i and the GPU server are on the same CPU core

GPU segment handling time

Self-suspension by higher-priority tasks[*]

$$W_i^{n+1} = C_i + \boxed{B_i^{gpu}} + \sum_{\tau_h \in \mathbb{P}(\tau_i) \wedge \pi_h > \pi_i} \left\lceil \frac{W_i^n + \boxed{(W_h - C_h)}}{T_h} \right\rceil C_h$$

$$+ \boxed{\sum_{\tau_j \neq \tau_i \wedge \eta_j > 0} \left\lceil \frac{W_i^n + \{D_j - (G_j^m + 2\eta_j \epsilon)\}}{T_j} \right\rceil (G_j^m + 2\eta_j \epsilon)}$$

Interference from the GPU server (e.g., GPU mem copy by server)

- Case 2: a task τ_i and the GPU server are on different cores

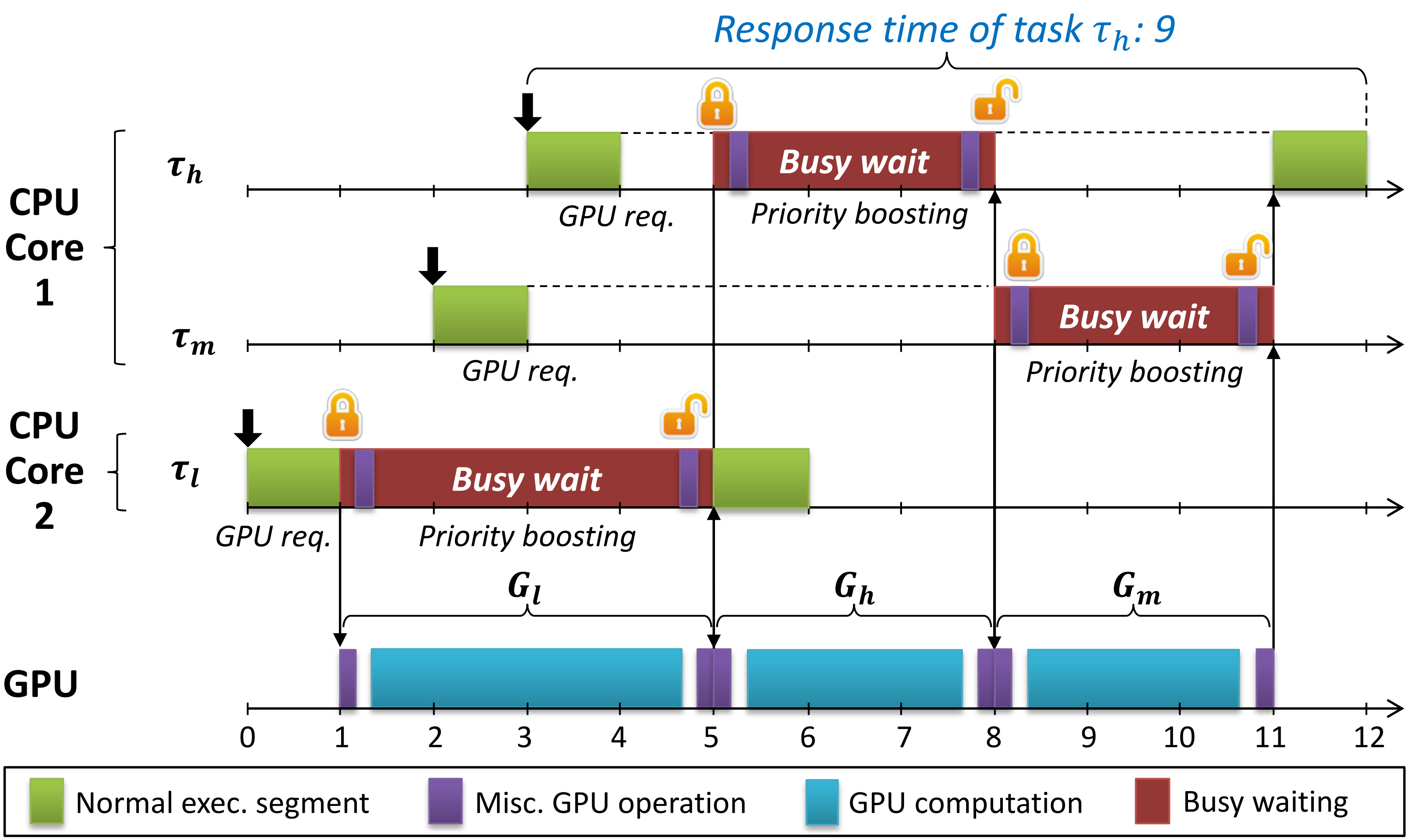
$$W_i^{n+1} = C_i + B_i^{gpu} + \sum_{\tau_h \in \mathbb{P}(\tau_i) \wedge \pi_h > \pi_i} \left\lceil \frac{W_i^n + (W_h - C_h)}{T_h} \right\rceil C_h$$

No interference from the GPU server (and misc. GPU operations)

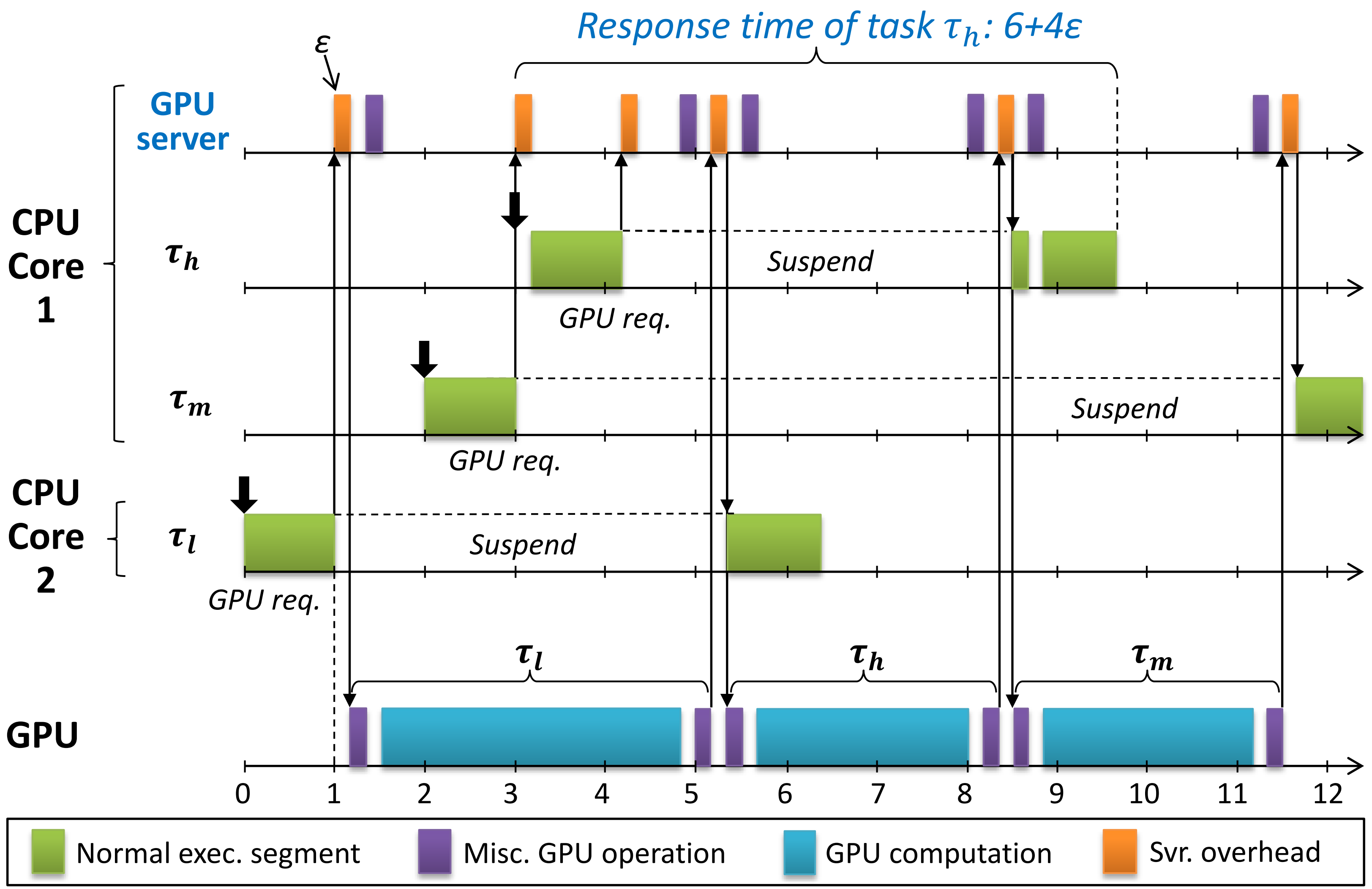
[*] J.-J. Chen et al. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Department of Computer Science, TU Dortmund, 2016.

Example under Synch-based Approach

* MPCP is used



Example under Server-based Approach



Implementation

- SABRE Lite board (NXP i.MX6 SoC)
  - Four ARM Cortex-A9 cores running at 1GHz
  - Vivante GC2000 GPU ⟶ *OpenCL*
  - NXP Embedded Linux kernel version 3.14.52
  - Linux/RK version 1.6 patch*

- GPU server overhead ϵ
  - Total of 44.97μs delay

* Linux/RK: http://rtml.ece.cmu.edu/redmine/projects/rk/

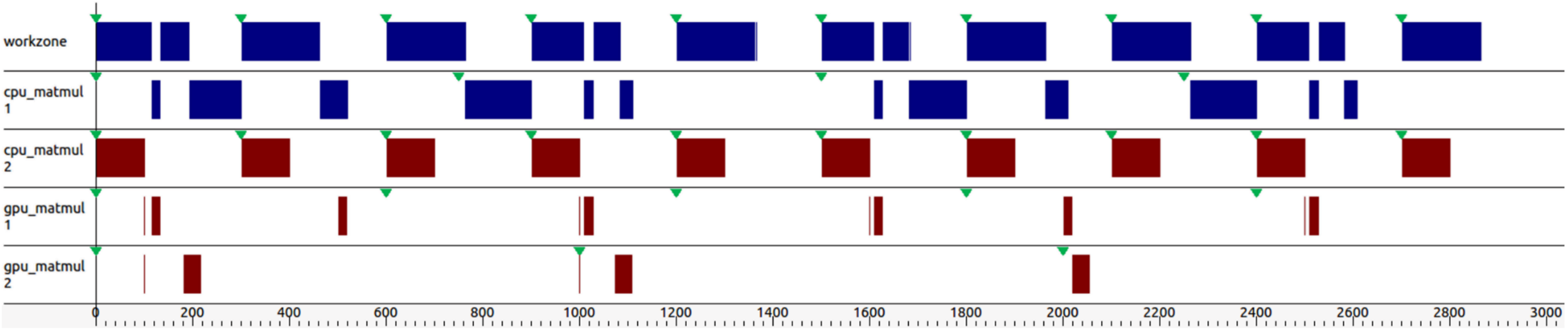
Case Study

- Motivated by the software system of the CMU's self-driving car[*]
  - Workzone Recognition Algorithm[†]
  - Two other GPU-using tasks and two CPU-only tasks

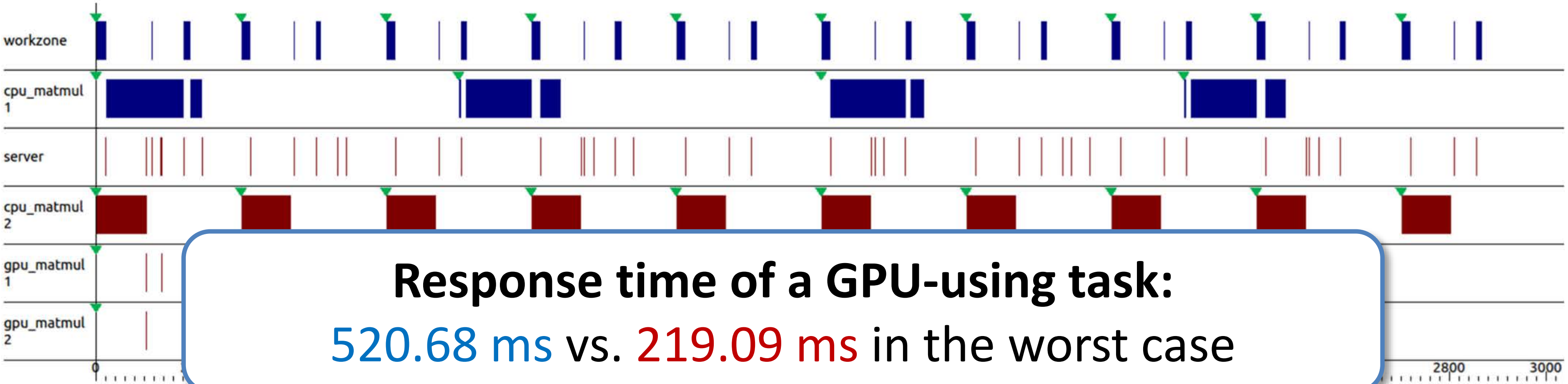
[*] J. Wei et al. Towards a viable autonomous driving research platform. In *IEEE Intelligent Vehicles Symposium (IV)*, 2013.
[†] J. Lee et al. Kernel-based traffic sign tracking to improve highway workzone recognition for reliable autonomous driving. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

Task Execution Timeline

- Synchronization-based approach (using MPCP)



- Server-based approach



Response time of a GPU-using task:
520.68 ms vs. 219.09 ms in the worst case

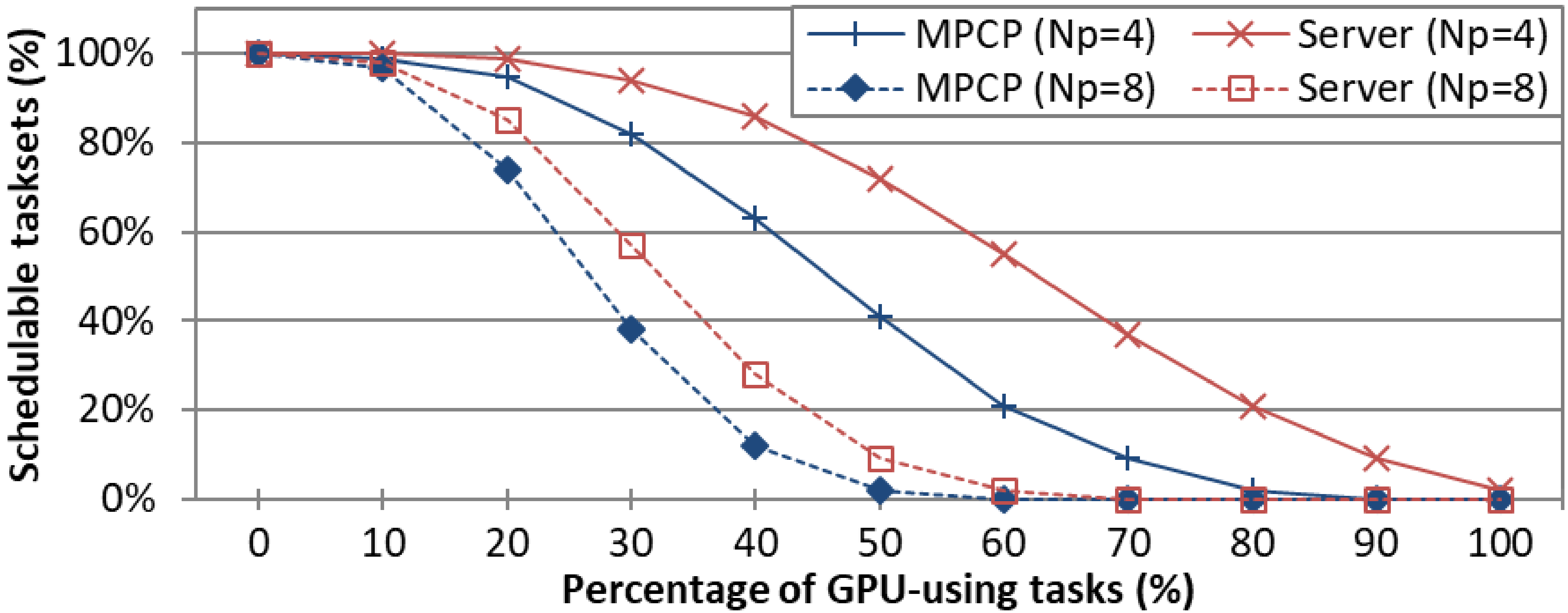
Schedulability Experiments

- **Purpose:** To explore the impact of the two approaches on task schedulability
- 10,000 randomly-generated tasksets

Parameters	**Values**
Number of CPU cores (N_P)	4, 8
Number of tasks per core	[3, 5]
Percentage of GPU-using tasks	[10, 30] %
Task period and deadline ($T_i = D_i$)	[100, 500] ms
Taskset utilization per core	[30, 50] %
Ratio of GPU segment len. to normal WCET (G_i/C_i)	[10, 30] %
Number of GPU segments per task (η_i)	[1, 3]
Ratio of misc. operations in $G_{i,j}$ ($G^m_{i,j}/G_{i,j}$)	[10, 20] %
GPU server overhead (ϵ)	50 μs

Results (1)

- Schedulability w.r.t. the percentage of GPU-using tasks

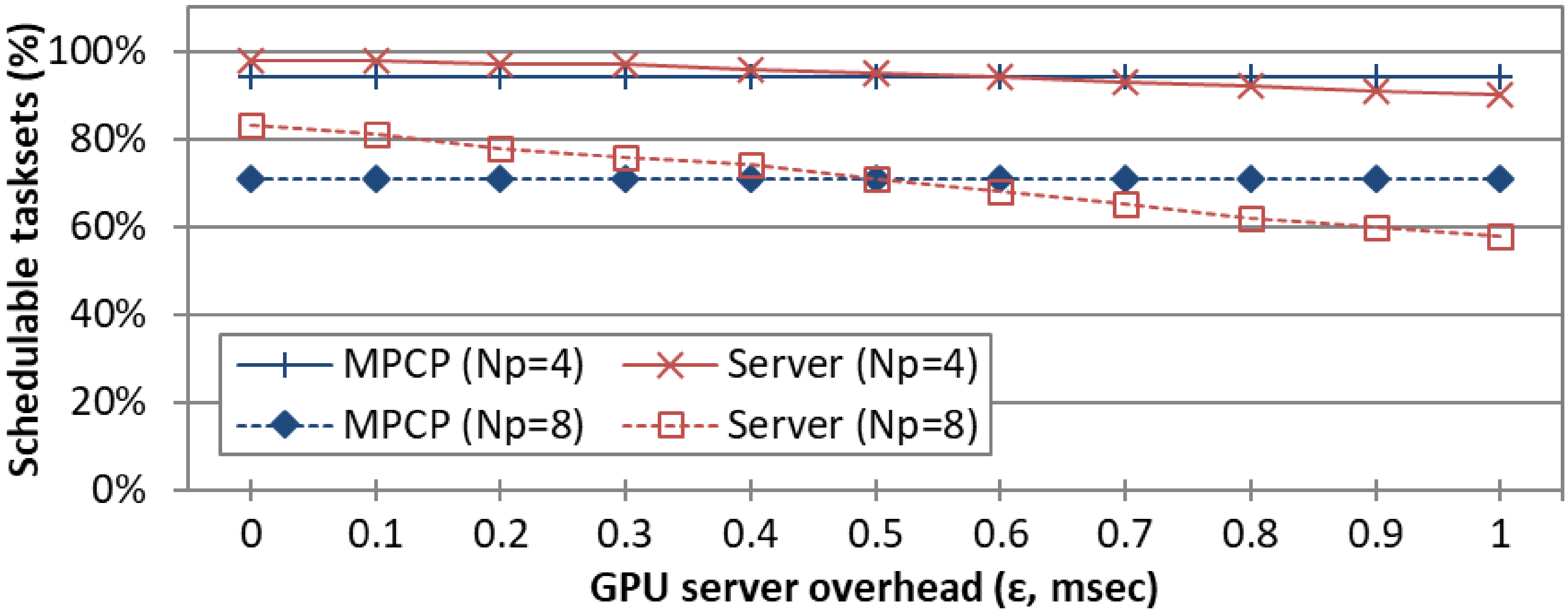


Server-based approach performs better in most cases (with realistic parameters)

Results (2)

44.97µs was the overhead in our platform

- Schedulability w.r.t. the GPU server overhead



Server-based approach does not dominate synchronization-based approach

Conclusions

- Server-based GPU access control
  - Motivated by the limitations of the synchronization-based approach
    - Busy-waiting and long priority inversion
  - Implementation with an acceptable overhead
  - Significant improvement over the synch-based approach in most cases
- Future directions
  - Improvement of analysis (worst-case waiting time calculation)
  - Comparison with other synchronization protocols
    - e.g., recent extension of FMLP+ allows self-suspension within critical sections
  - GPU server has a central knowledge of all GPU requests
    - Efficient co-scheduling of GPU kernels, GPU power management, etc.

Thank You

A Server-based Approach for Predictable GPU Access Control

Hyoseung Kim[*] Pratyush Patel[†] Shige Wang[‡] Raj Rajkumar[†]

[*] University of California, Riverside
[†] Carnegie Mellon University
[‡] General Motors R&D