STGM: Spatio-Temporal GPU Management for Real-Time Tasks

Sujan Kumar Saha [†*], Yecheng Xiang[*], Hyoseung Kim[*]

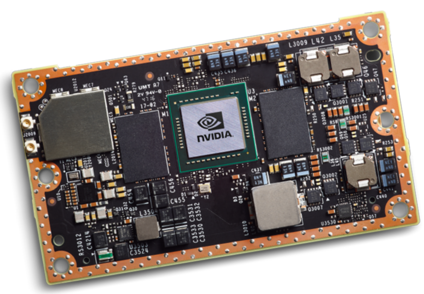
UC RIVERSIDE UNIVERSITY OF CALIFORNIA RTEN Real-time Embedded and Networked Systems Laboratory [*]

UF UNIVERSITY of FLORIDA [†]

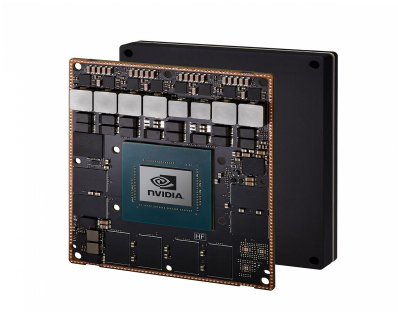
Status Quo

- **Heavy computational demand** on real-time embedded systems.
  - Hard to meet task deadlines
  - Usage of GPU can significantly improve the performance

- Real-time embedded systems usually have one GPU and consider GPU **as a single indivisible resource**.
  - Only one task can access GPU at one time
  - **Underutilization of the GPU**
  - **Hard to schedule all tasks**



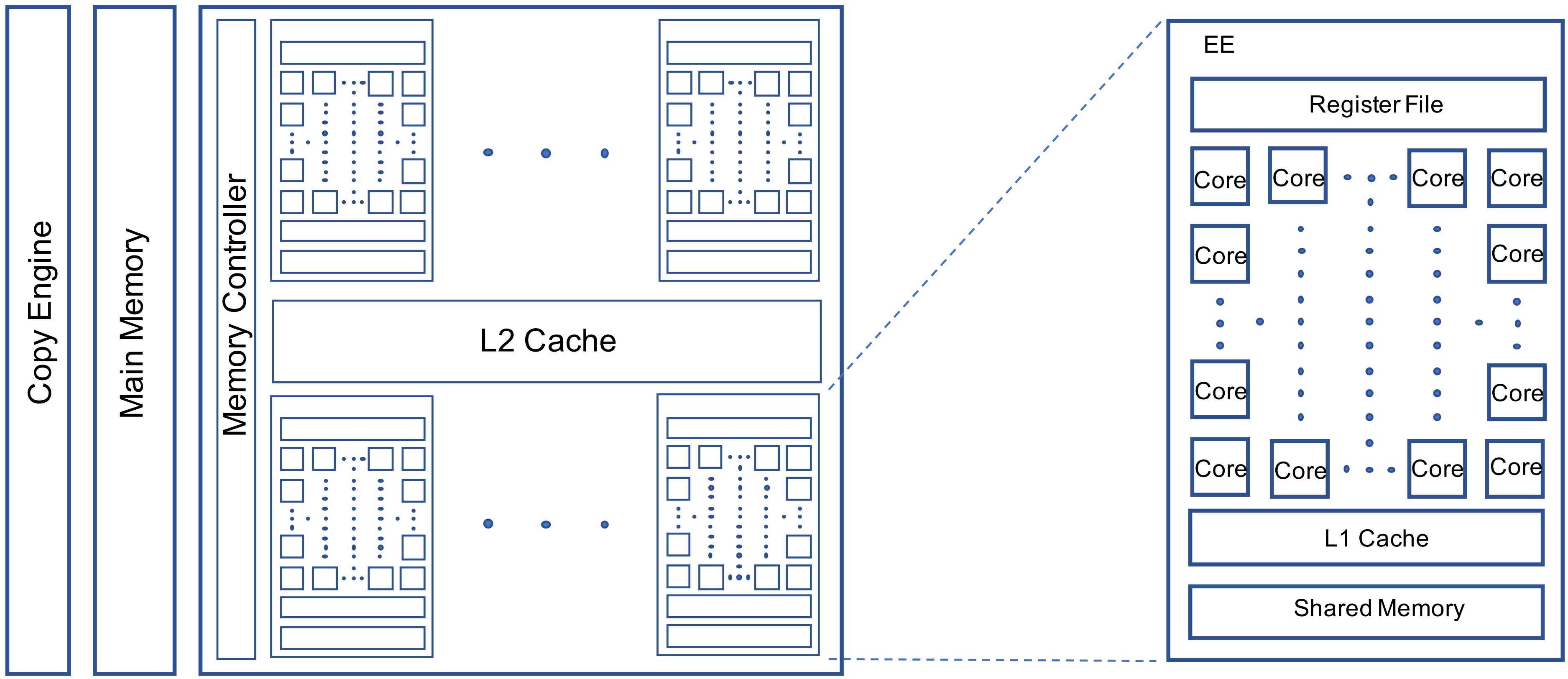
NVIDIA TX1/TX2



NVIDIA Xavier

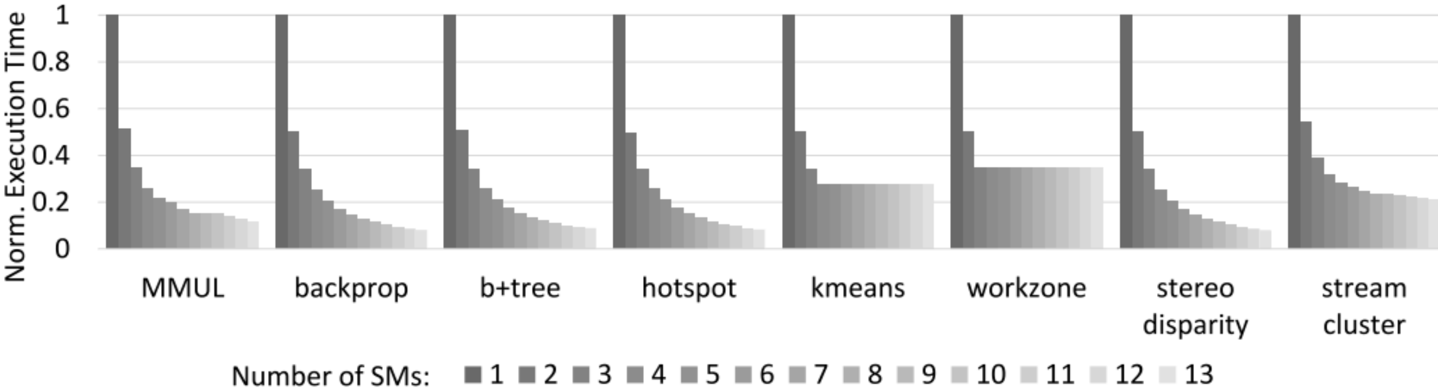
Apple A12X

Background – GPU Architecture

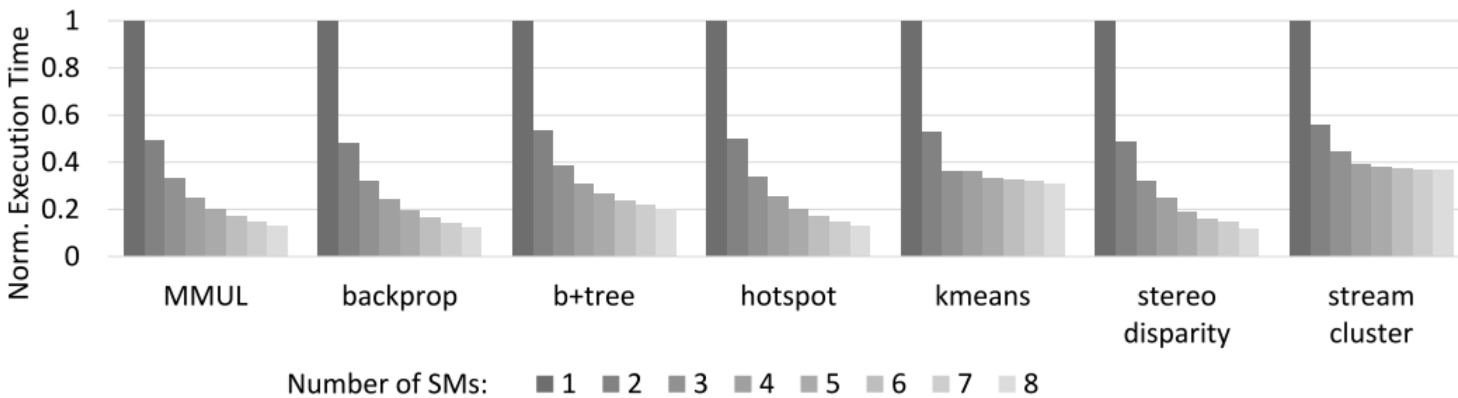


Overview of GPU Architecture

Kernel Execution Time



(a) GTX970



(b) AGX Xavier

Prior Work & Motivation

- GPUSync[1], Server-based GPU Control[2],
  - Efforts towards predictable real-time GPU control.
  - GPU is modeled as a non-preemptive indivisible resource.

- RGEM[3], GPES[4]
  - Allowing GPU preemptions by splitting kernel and data operations into sub-parts.
  - But GPU executes only one kernel at a time.

- GPU partitioning for general-purpose systems[5]
  - Enabling the allocation of SMs to tasks to improve overall GPU utilization.
  - No systematic support or analysis to derive the worst-case response time and schedulability of tasks. (e.g., blocking time due to shared copy engine)

[1] G. Elliott et al. GPUSync: A framework for real-time GPU management (RTSS, 2013)
[2] H. Kim et al. A server-based approach for predictable GPU access control (RTCSA, 2017)
[3] S. Kato et al. RGEM: A responsive GPGPU execution model for runtime engines (RTSS, 2011)
[4] H. Zhou et al. GPES: a preemptive execution system for GPGPU computing (RTAS, 2015)
[5] J. Janz´en et al. Partitioning GPUs for improved scalability (SBAC-PAD, 2016)

Contribution

- STGM: Spatio-Temporal GPU Management framework
  - Allows multiple tasks to utilize GPU simultaneously in a time-analyzable manner.
  - Efficient allocation algorithm of GPU resources to tasks and tasks to CPU cores.
  - Provides schedulability analysis that bounds the maximum blocking time and worst-case response time of tasks.

- We have implemented STGM on COTS platforms and evaluated its performance compared to existing approaches.
  - Experiment results indicate STGM outperforms significantly in schedulability compared to other existing approaches.

STGM – System Model

- Multi-core System (N_P CPU Cores)
- Single GPU (N_{SM} SMs)
- Single Copy Engine (CE), serves in FIFO manner.
- Partitioned Fixed-priority Preemptive Scheduling

General Task Model

$$\tau_i := (\underbrace{C_i}_{\text{CPU Seg.}}, \underbrace{G_i(k)}_{\text{GPU Seg.}}, \underbrace{T_i}_{\text{Period}}, \underbrace{D_i}_{\text{Deadline}}, \eta_i, \theta_i)$$

GPU Task Model

$$\tau^*_{i,j} := (\underbrace{G^{m_{hd}}_{i,j}}_{\text{H2D Memcpy}}, \underbrace{G^{e}_{i,j}(k)}_{\text{Kernel Execution}}, \underbrace{G^{m_{dh}}_{i,j}}_{\text{D2H Memcpy}})$$

STGM – Spatial & Temporal Management

Spatial Management

- Partition a GPU into SMs
- Allocate SMs to tasks by using resource allocation algorithm.

Temporal Management

- **Case 1**: a task does not share SMs with other tasks.
    - Its kernels start execution immediately after data copy finishes.
- **Case 2**: a task shares SMs with others.
    - Wait until all previously-launched kernels with shared SMs are finished. (FIFO)
    - Priority-boosting is used to minimize interference during GPU segment execution.

STGM – Schedulability Analysis

When launching kernel, there are two modes of operations on the CPU side: **self-suspension** and **busy-waiting**.

Self-suspension:

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

Busy-waiting:

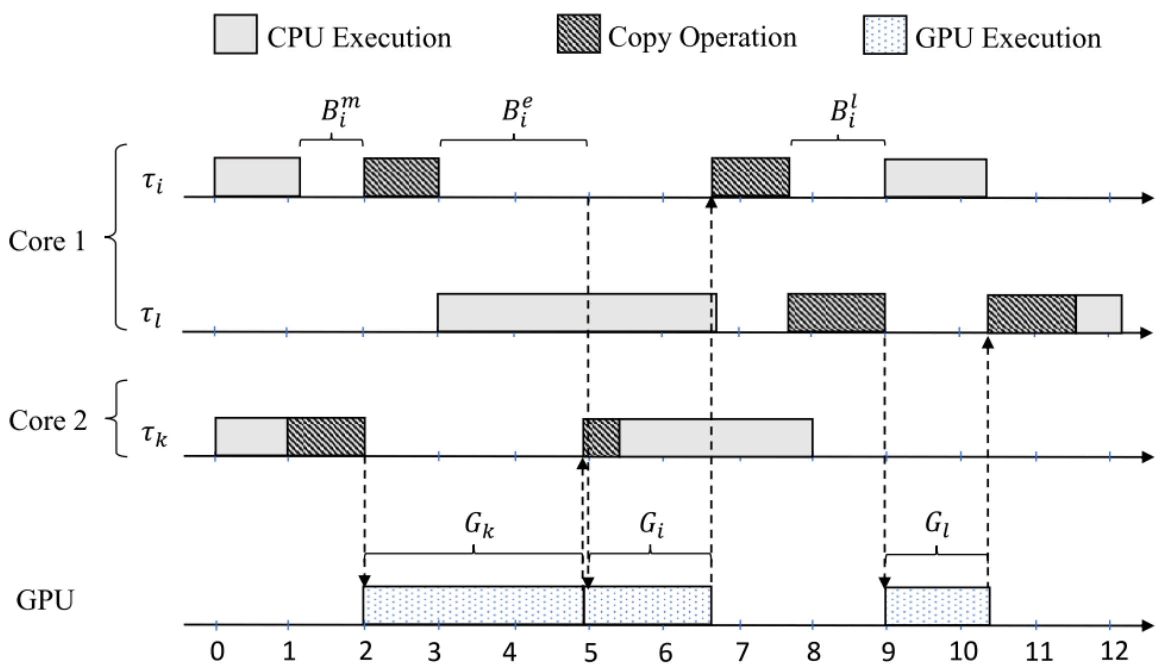
$$W_i^{k+1} = C_i + G_i + B_i + \sum_{\tau_h \in \mathbb{P}(\tau_i) \wedge \pi_h > \pi_i} \lceil \frac{W_i^k}{T_h} \rceil (C_h + G_h + B_i) \quad (9)$$

Capturing preemption from higher priority tasks.

STGM – Schedulability Analysis

Blocking Time B_i

$$B_i = B_i^m + B_i^e + B_i^l$$



B_i^m The blocking time from **GPU data copy** and miscellaneous operations in GPU segments

B_i^e The blocking time from **kernel execution**

B_i^l The blocking time from **priority inversion**

STGM – Resource Allocation

- **This allocation algorithm allocates SMs to tasks, tasks to CPUs and check the schedulability.**
  - **It is based on an extension of WFD heuristic.**
  - **As the number of tasks is limited, the algorithm will converge after allocating the tasks to cores if the taskset is not schedulable even after assigning all available SMs.**

Algorithm 1 SM-aware Task Allocation Algorithm

Require: Γ: a taskset, N_p: Number of CPU cores, N_{SM}: Total number of SM in GPU, P: set of CPU cores (i.e., $|P| = N_p$)
Ensure: N_i: Number of SMs for each task $\tau_i \in \Gamma$, S_i: SM indices for each task $\tau_i \in \Gamma$, Γ_p: a taskset allocated to a CPU core p, U_p: Utilization of tasks in Γ_p if schedulable and ∞ otherwise.

1: **for all** $\tau_i \in \Gamma$ **do**
2:   **if** $\eta_i > 0$ **then** /* GPU-using task */
3:     $N_i \leftarrow 1$
4: **for** $p \leftarrow 1$ to N_p **do**
5:   $U_p \leftarrow 0; \Gamma_p \leftarrow \emptyset$
6: /* SM Allocation */
7: $sm\_idx \leftarrow 0$
8: **for all** $\tau_i \in \Gamma$ **do**
9:   **if** $\eta_i > 0$ **then** /* GPU-using task */
10:     $S_i \leftarrow \emptyset$
11:     **for** $k \leftarrow 1$ to N_i **do**
12:       $S_i \leftarrow S_i \cup \{sm\_idx\}$
13:       $sm\_idx \leftarrow (sm\_idx + 1) \mod N_{SM}$
14: **for all** $\tau_i \in \Gamma$ in decreasing order of utilization **do**
15:   **for** $p \in P$ in increasing order of utilization **do** /* WFD */
16:     **if** $1 - U_p \geq C_i/T_i$ and τ_i satisfies Eq. (1) or (9) **then**
17:       $U_p \leftarrow U_p + C_i/T_i$
18:       $\Gamma_p \leftarrow \Gamma_p \cup \tau_i$
19:       Mark τ_i schedulable
20:       **break**
21: **if** all tasks in Γ schedulable **then**
22:   **return** $\{N_i, S_i, \Gamma_p, U_p\}$
23: **else if** $\exists N_i \leq N_{max}$ **then**
24:   $i \leftarrow \underset{\forall i:\tau_i \in \Gamma \wedge \eta_i > 0}{\text{argmax}} (G_i(N_i + 1) - G_i(N_i))/T_i$
25:   $N_i \leftarrow N_i + 1$
26:   Go to line 7
27: **else**
28:   **return** ∞

Evaluation – Experiment Setup

- **x86 Server**
  - Quad-core Intel i7 6700 3.4GHz CPU
  - NVIDIA GTX 970
    - 4GB Memory
    - 13 SMs
    - 2 CEs
  - Ubuntu 16.04
  - CUDA 9.0

- **Nvidia Xavier**
  - 8-core ARM CPU
  - Integrated Volta GPU
    - 16GB unified memory
    - 8 SMs
    - 1 CE
  - Ubuntu 18.04
  - CUDA 10.0

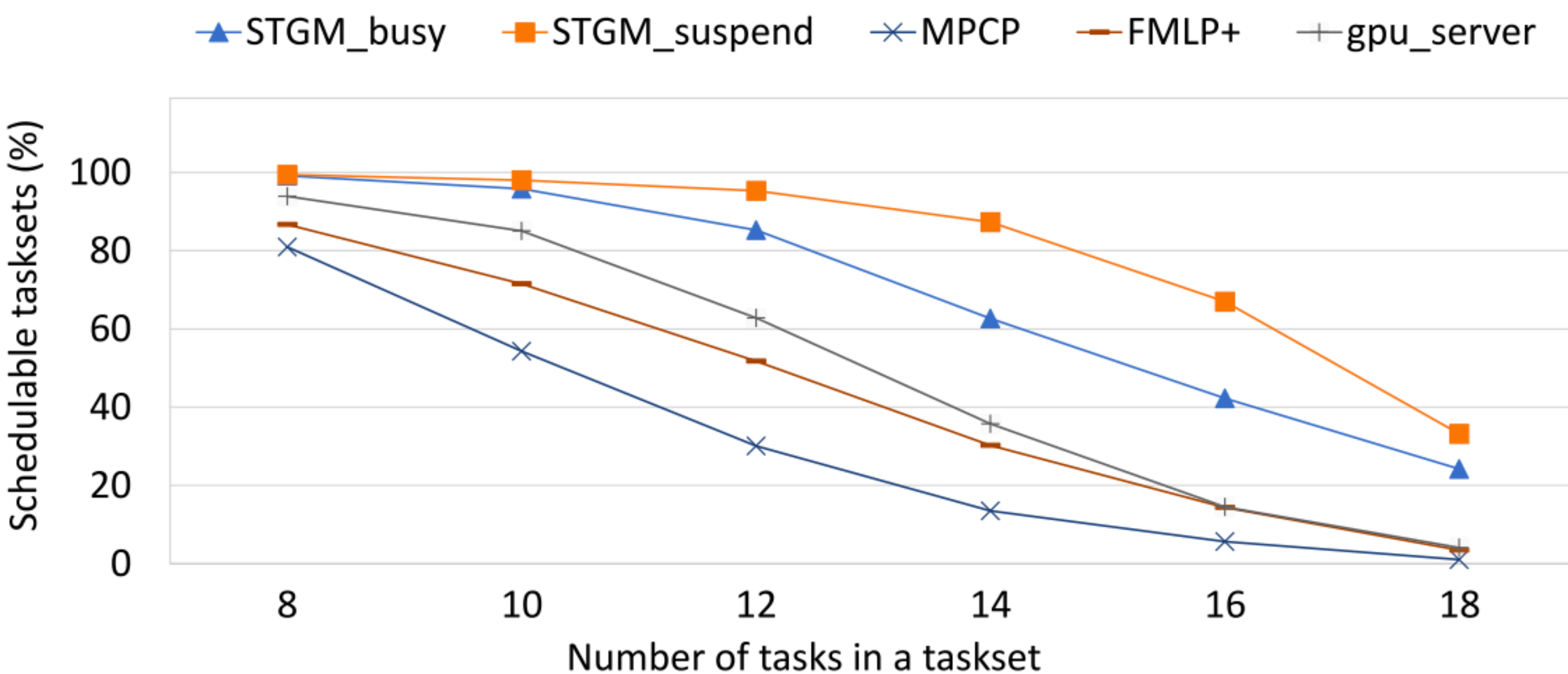
Evaluation – Schedulability

- 10,000 random tasksets from profile data for each schedulability experiment.
- Taskset generated randomly based on the execution time profile and parameters in Table I (see paper).

STGM_busy STGM_suspend MPCP FMLP+ gpu_server

- We compare the percentage of schedulable tasksets of STGM with MPCP, FMLP+ (sync-based approach), and server-based approach.
  - All these prior works **consider GPU as an indivisible device**.

Evaluation – Schedulability



Conclusion & Future Work

- **Conclusion**
  - We proposed STGM, which allows multiple tasks to utilize GPU simultaneously in a time-analyzable manner.
  - We designed an efficient allocation algorithm of GPU resources to tasks and tasks to CPU cores.
  - Schedulability analysis is provided which bounds the maximum blocking time and worst-case response time of tasks.
  - We have implemented STGM on COTS platforms and evaluated its performance compared to prior works.
  - Evaluation results indicate significant improvement in schedulability compared to other existing approaches.

- **Future Work**
  - Multiple-GPUs
  - Shared memory-induced interference (e.g., cache/memory bus)

STGM: Spatio-Temporal GPU Management for Real-Time Tasks

Sujan Kumar Saha, Yecheng Xiang, Hyoseung Kim

Thank you!

Q & A