Multi-Tenant DNN Inference: Spatial GPU Sharing

Yijia Diao

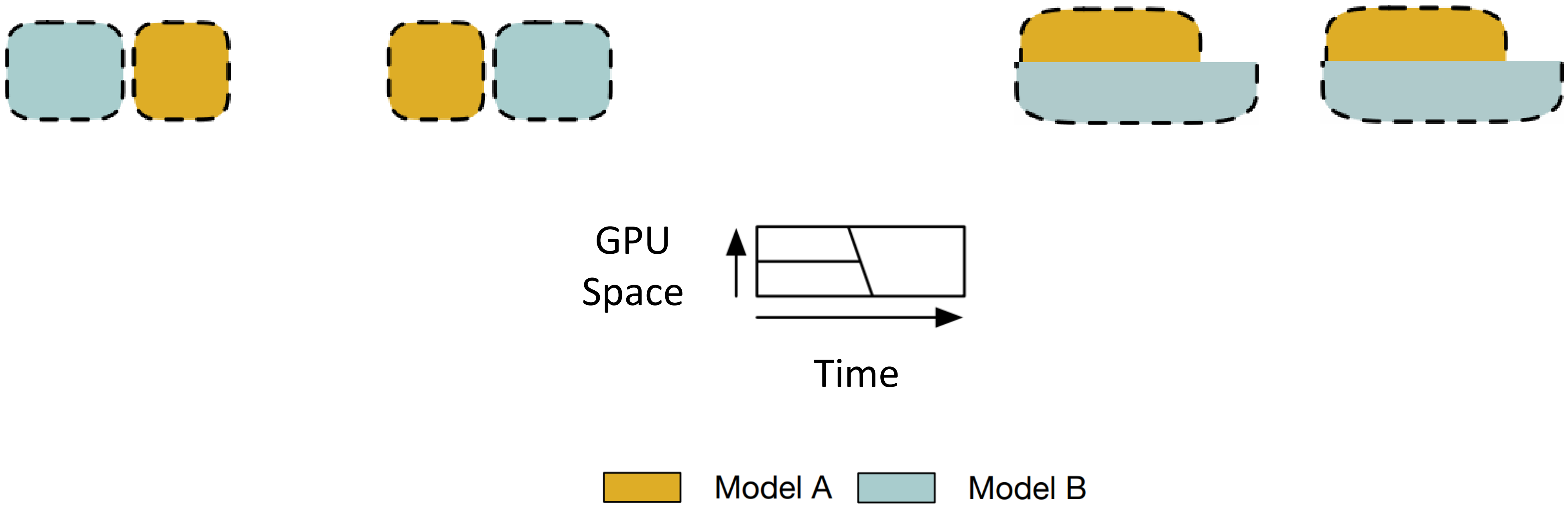
2023.05.11

Outline

- GPU Sharing Basic
- Background
- MPS Improvement: GSLICE
- Kernel-wise Sharing: KRISP
- Summary

Two Kinds of GPU Sharing

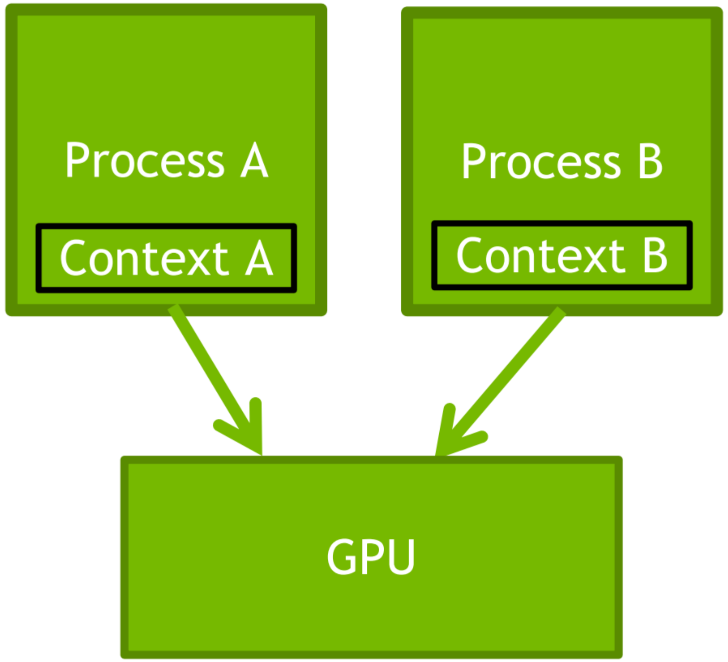
- Temporal Sharing
- Spatial Sharing

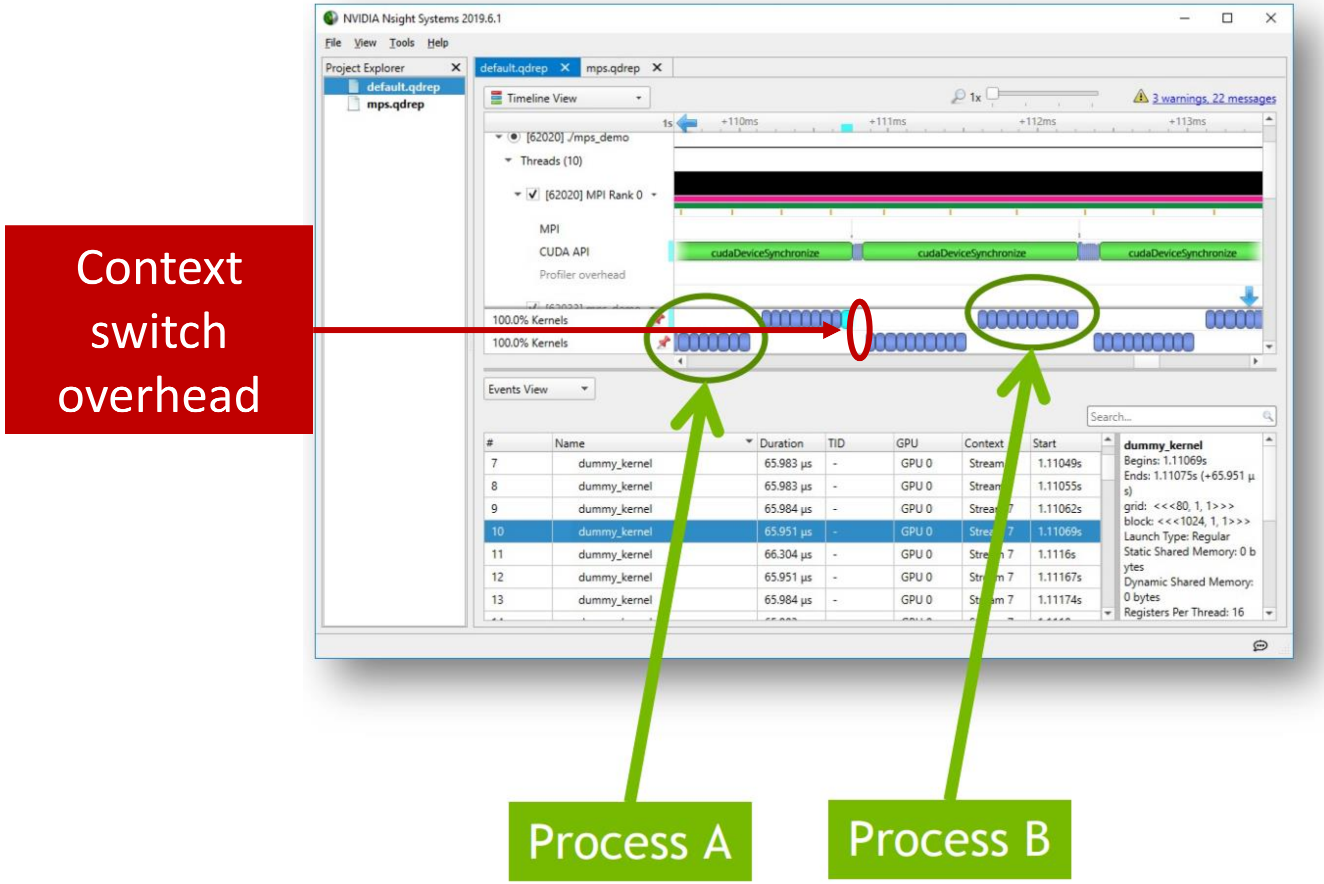


S. Choi et al. Multi-model Machine Learning Inference Serving with GPU Spatial Partitioning (arXiv 21).

Temporal GPU Sharing

- Reached simply by different process on GPU without other operation.

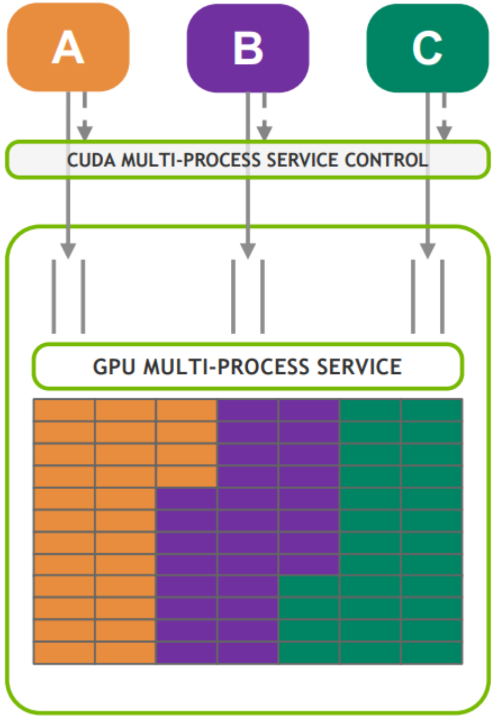




https://www.olcf.ornl.gov/wp-content/uploads/2021/05/GPU-Concurrency-Overview.pdf

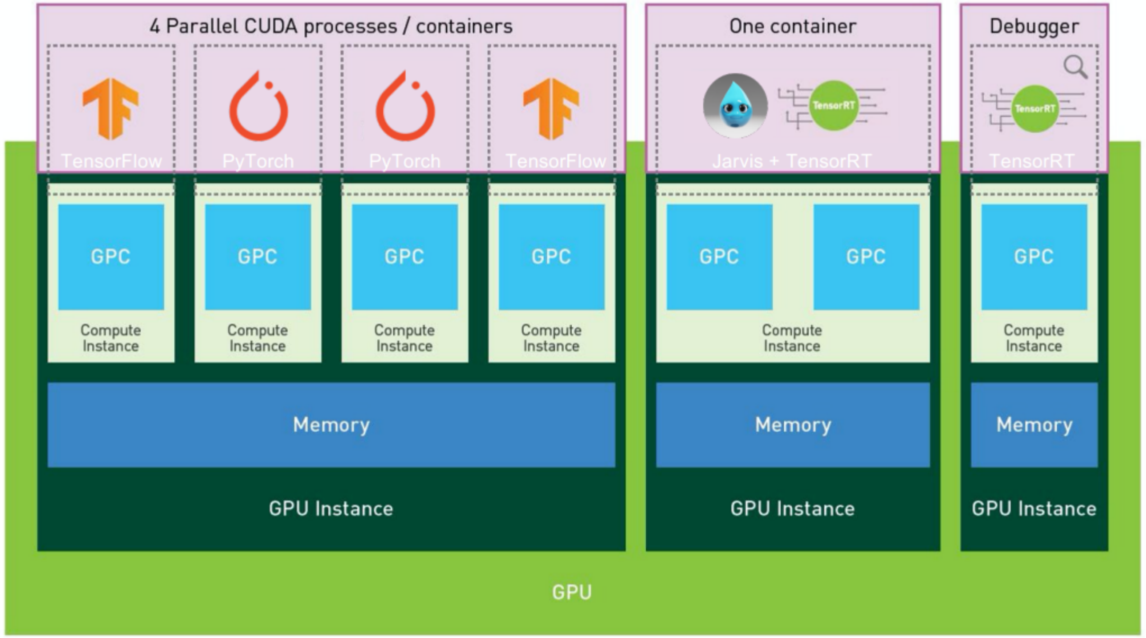
Spatial GPU Sharing: MPS&MIG

- Multi-Process Service, Multi-Instance GPU



Multi-Process Service

Dynamic contention for GPU resources
Single tenant



Multi-Instance GPU

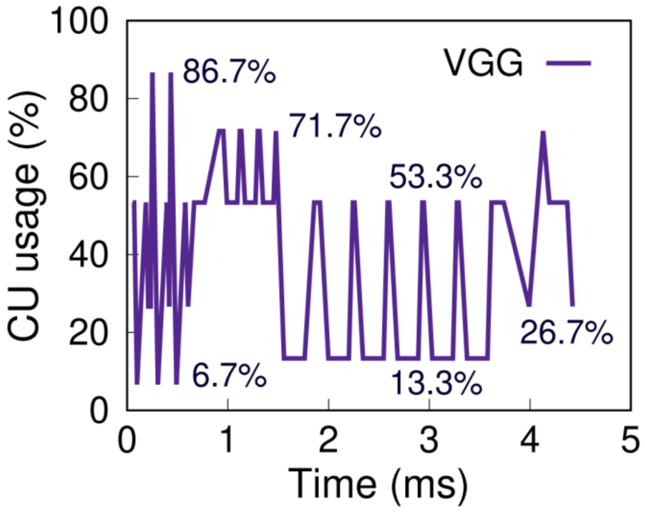
Hierarchy of instances with guaranteed resource allocation
Multiple tenants

Outline

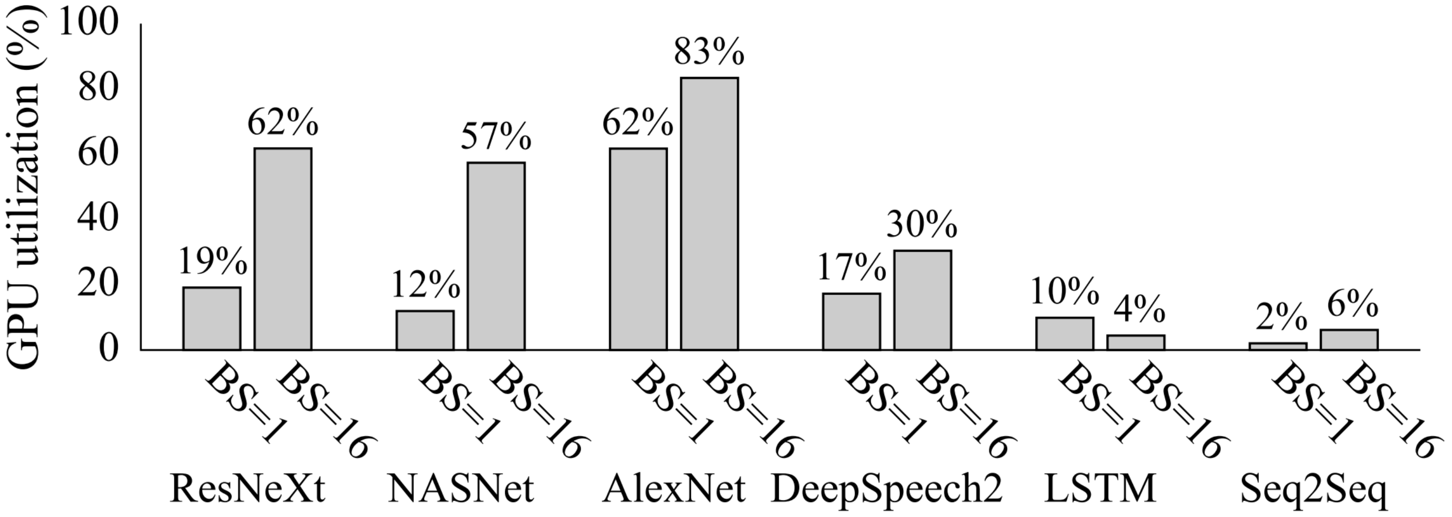
- GPU Sharing Basic
- Background
- MPS Improvement: GSLICE
- Kernel-wise Sharing: KRISP
- Summary

Why “Multi-Tenant” of DNN Inference?

- Low utilization allow to share single hardware among multiple tasks.



Timeline of CU usage of VGG on GPU

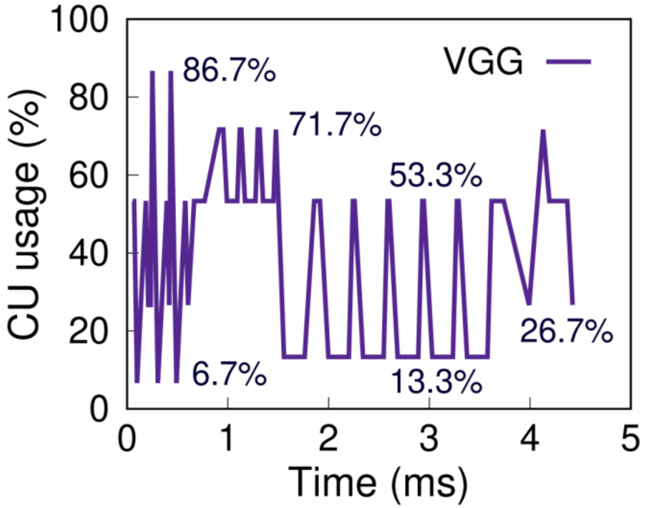


Average GPU utilization of different DNN models

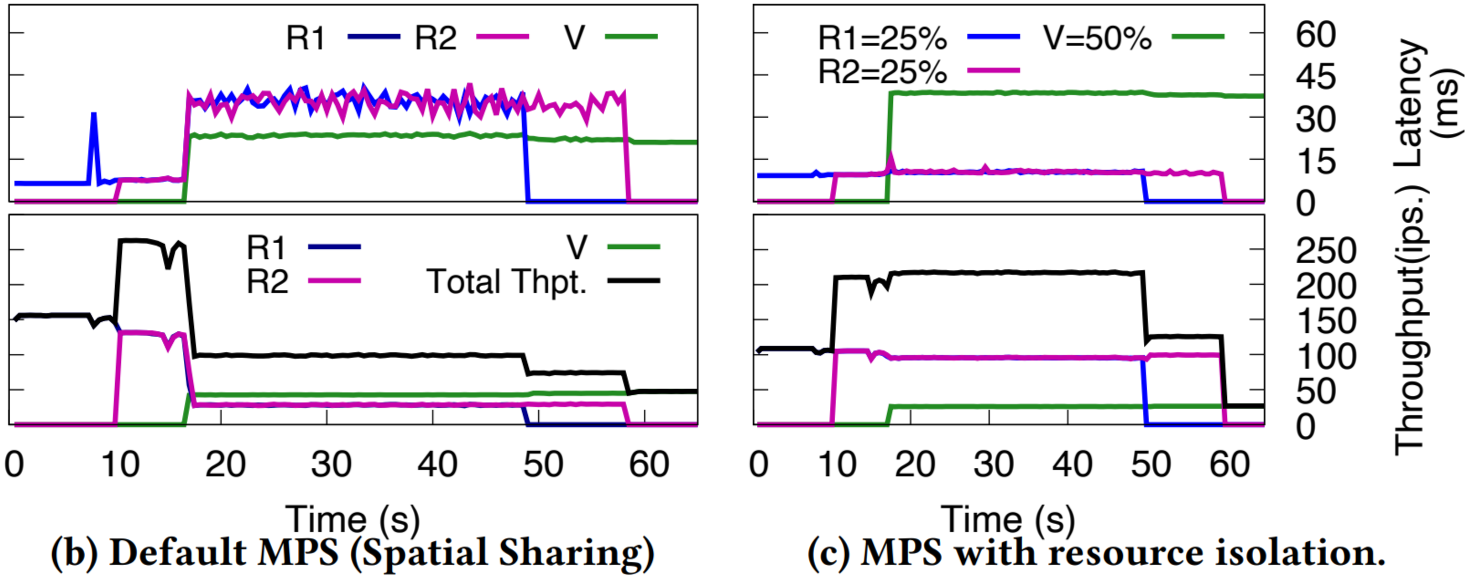
M. Han et al. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences (OSDI 22).
L. Ma et al. Rammer: Enabling holistic deep learning compiler optimizations with rtasks (OSDI 20).

Dynamic Resource Allocation is Important

- GPU utilization changes within a model
- Service level Objective(SLO)



Timeline of CU usage
of VGG on GPU



Inference Throughput & Latency of ResNet and VGG

A. Dhakal et al. GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference.(SoCC 20)

Problem of MPS&MIG: Lack of Dynamic

- If we want Model A scale up 60% -> 80% resource, the overhead of reset is huge (~10s).

M. Chow et al. KRISP: Enabling Kernel-wise Right-sizing for Spatial Partitioned GPU Inference Servers.(HPCA 23)

Outline

- GPU Sharing Basic
- Background
- MPS Improvement: GSLICE
- Kernel-wise Sharing: KRISP
- Summary

GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform

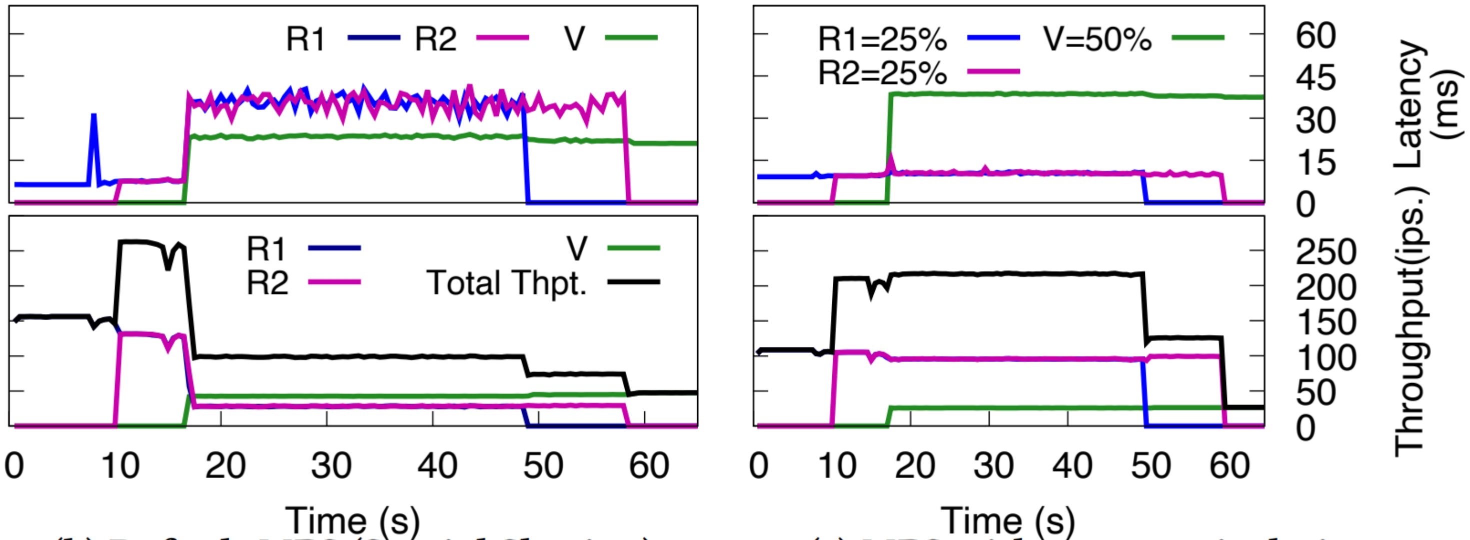
Aditya Dhakal
University of California, Riverside
adhak001@ucr.edu

Sameer G Kulkarni
IIT, Gandhinagar
sameergk@iitgn.ac.in

K. K. Ramakrishnan
University of California, Riverside
kk@cs.ucr.edu

GSLICE Scenario

- Spatial sharing: MPS
- Throughput: image per second

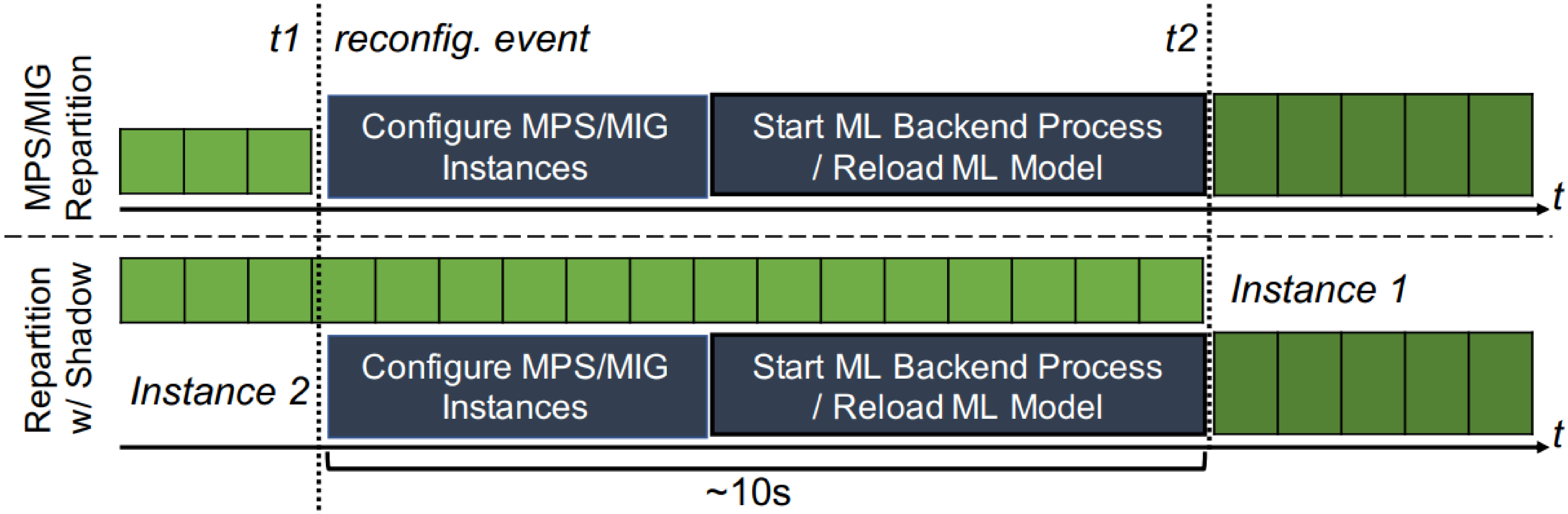


(b) Default MPS (Spatial Sharing)

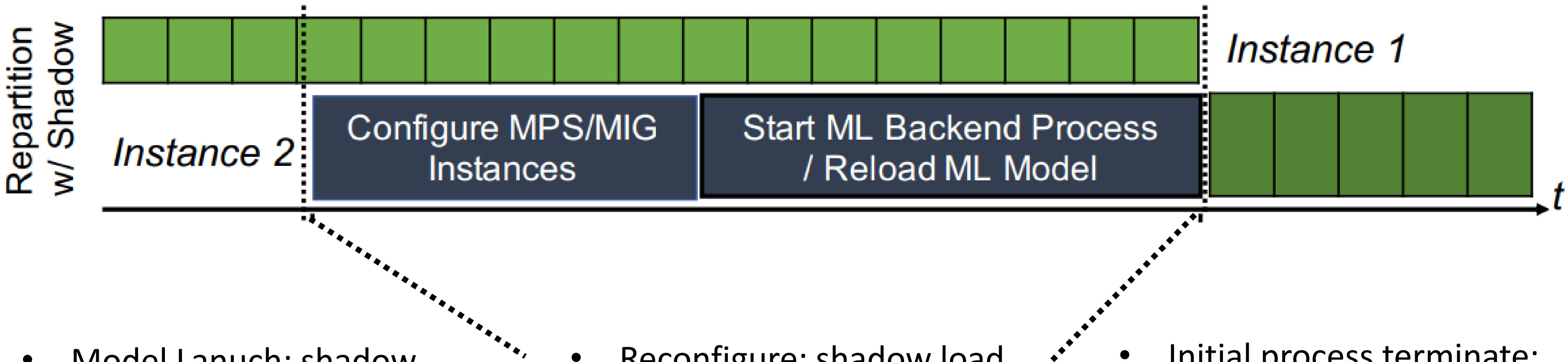
(c) MPS with resource isolation.

GSLICE: Overlap to hide new process overhead

- When changing process's GPU%, keep the current process and reconfigure in shadow process.



Initial Process & Shadow Process



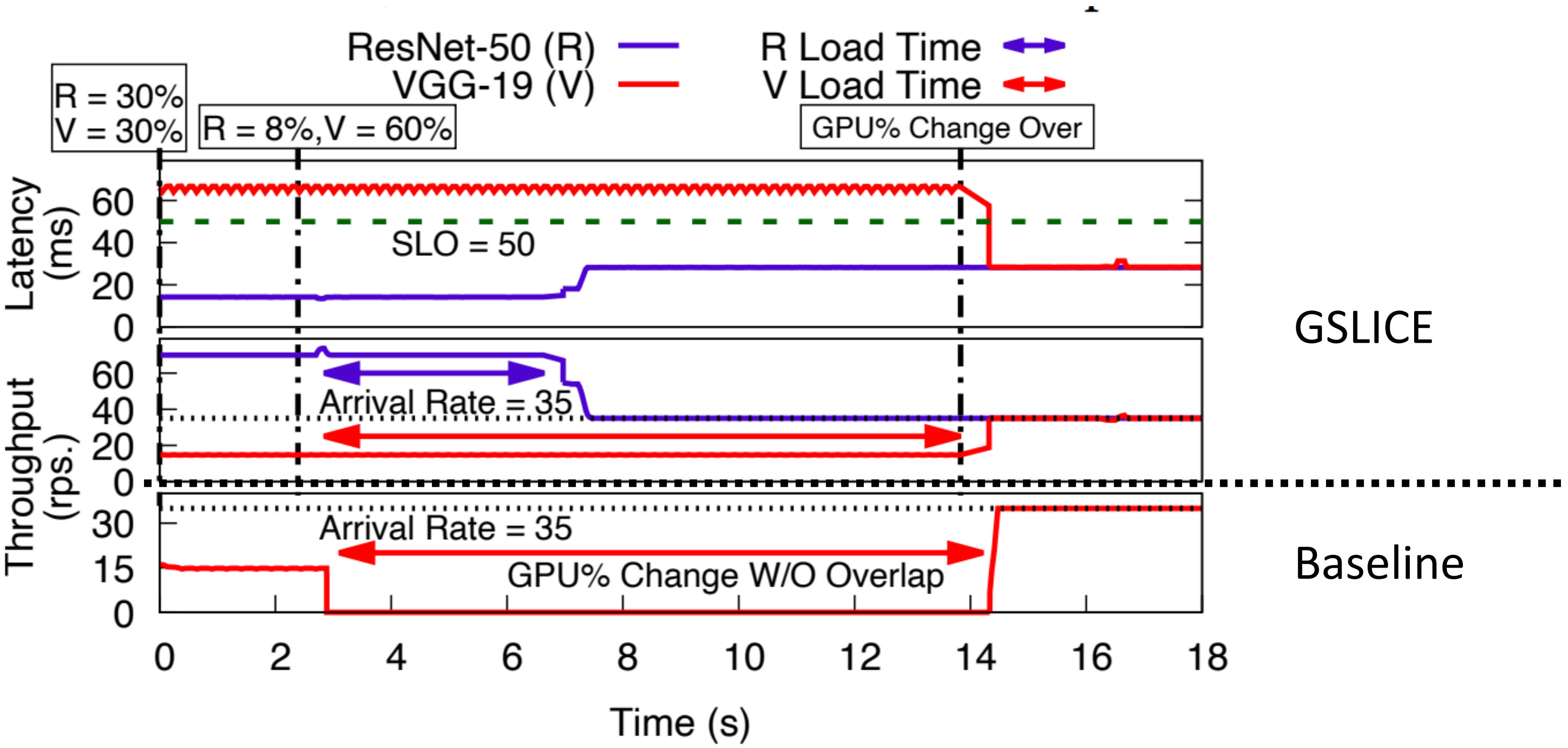
- Model Lanuch: shadow process also starts.
- Share same resources: buffer pool, CPU core
- Shadow does not access GPU

- Reconfigure: shadow load the model to GPU

- Initial process terminate: non-preemtion.

Evaluation on Process Resource Change

- Throughput does not waste when GPU% change
- Still need ~10s.



Comments of GSLICE

- Could not find the source code to confirm the method.
- Seems when changing GPU%, there would be 2× on-chip memory usage😨

Outline

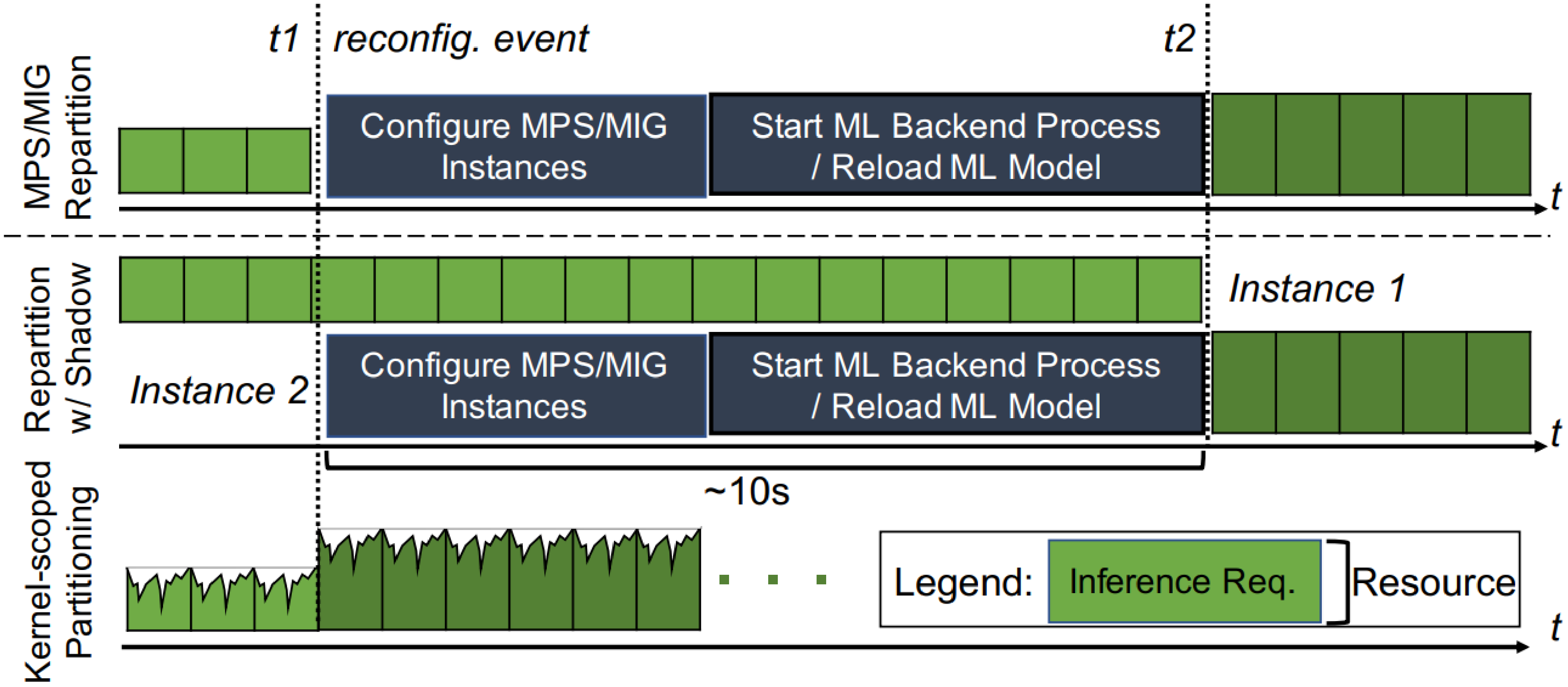
- GPU Sharing Basic
- Background
- MPS Improvement: GSLICE
- Kernel-wise Sharing: KRISP
- Summary

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

KRISP: Enabling Kernel-wise RIght-sizing for Spatial Partitioned GPU Inference Servers

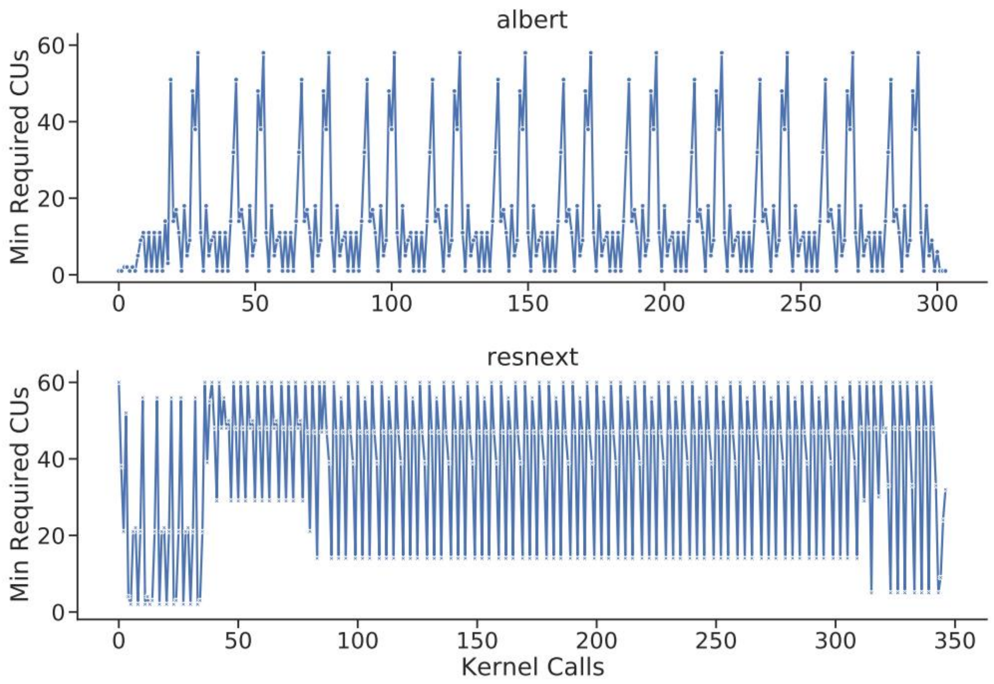
Motivation of KRISP

- Kernel-level spatial sharing: lightweight dynamic resource control



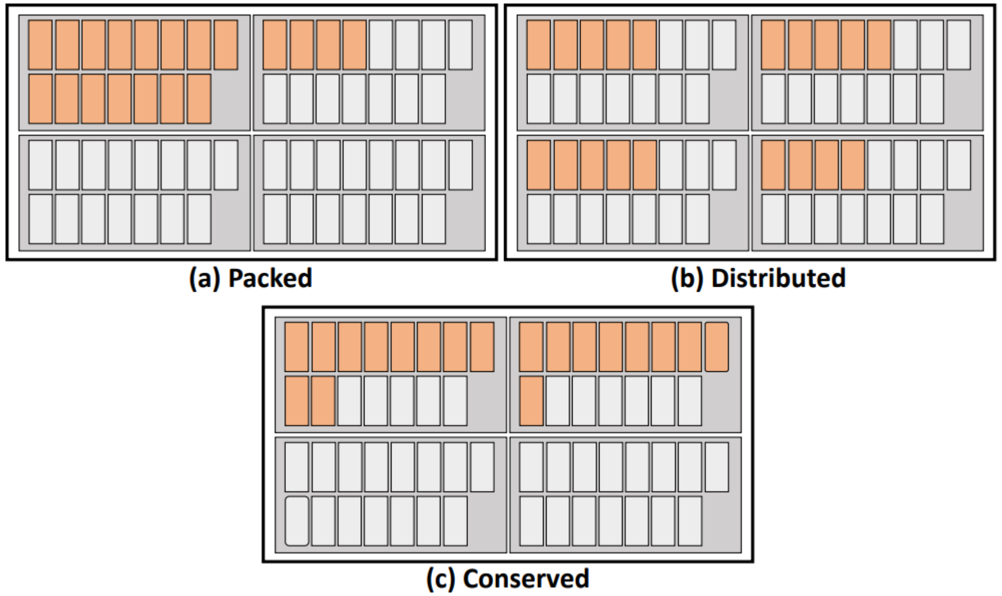
Basis of Kernel-wise Spatial Sharing

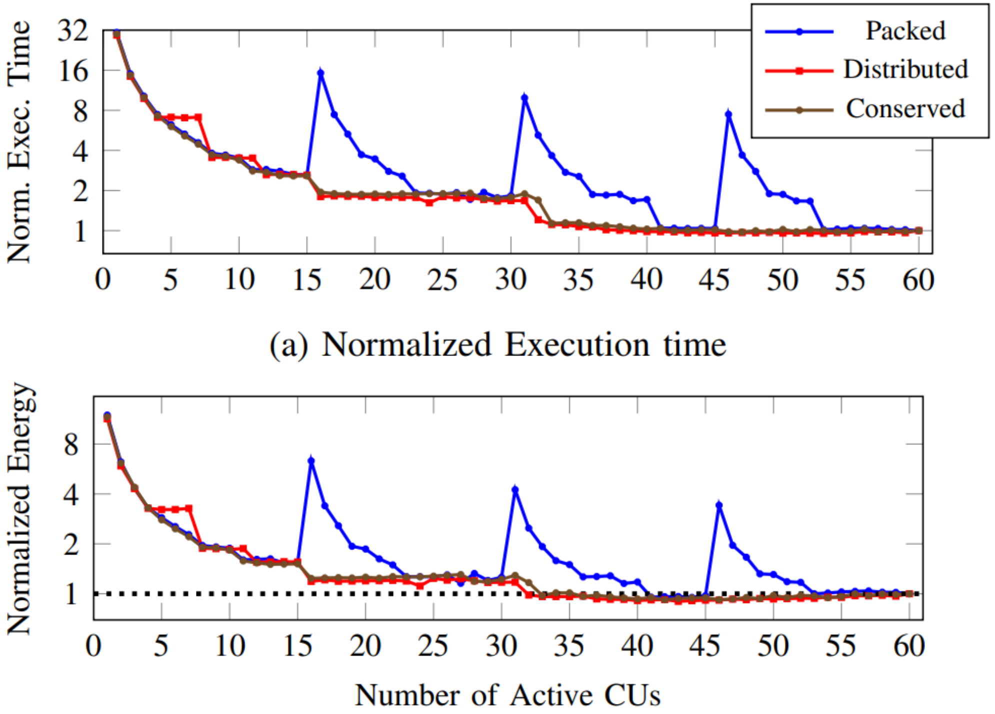
- Minimum required CUs: least number of CUs that have the same latency
- GPU-accelerated libraries: rocBLAS, MIOpen
- Offline Profiling.



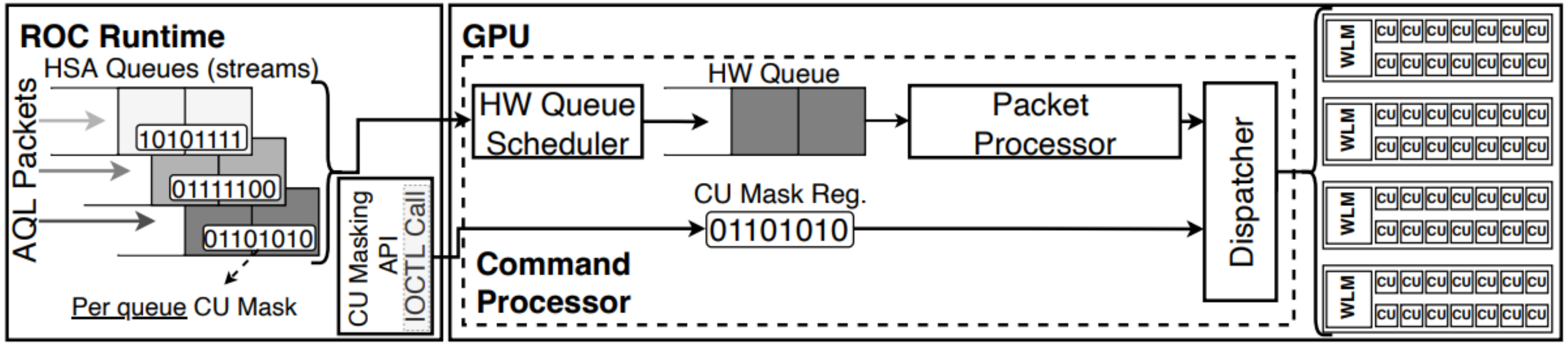
Allocate kernel to specific CU

- AMD MI50 GPU: 4 shader engine, 60 CUs
- Add mask to each kernel to locate on specific CU
- Choose conserved policy



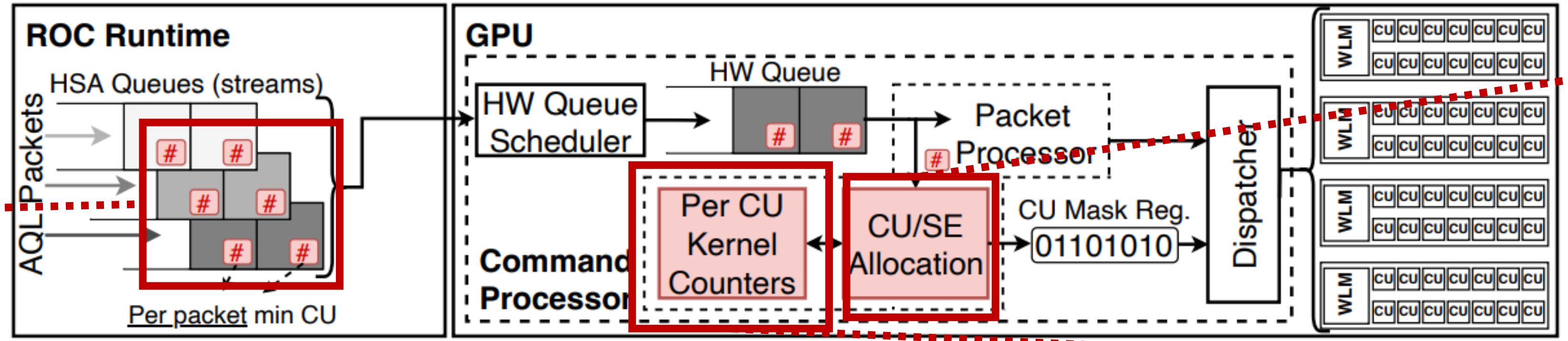


Architecture Support: GPU Driver



(a) Baseline AMD GPU architecture with Stream-scoped CU Masking API support.

Minimum CU info. & Mask



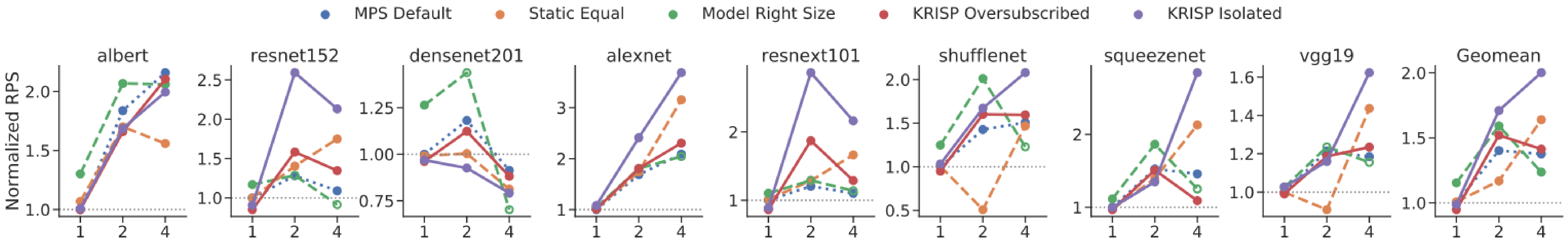
Kernel mask generation algo.

CU resource counters.

(b) Modifications for enabling Kernel-scoped Partition Instance.

Evaluation

- Model Throughput: 2× on average
- Up to 3.5× over MPS Default(1 worker)
- Overhead: offline-profiling; CPU Storage



(a) Model Throughput

Summary

	Spatial Partitioning	Granularity	Resize Overhead	Resize Overhead Masking	Reload Model?
GSLICE	MPS	Model	~10s	Overlap	Yes
KRISP	Kernel	Kernel	Low(ms)	--	No

Thanks!