# Hardware Compute Partitioning on NVIDIA GPUs for Composable Systems

Joshua Bakita and James H. Anderson

Department of Computer Science
University of North Carolina, Chapel Hill

# How can we do more, with less?

# How can we do more, with less, on the CPU?

## Processor Cores

➔ Concurrency, with hardware partitioning

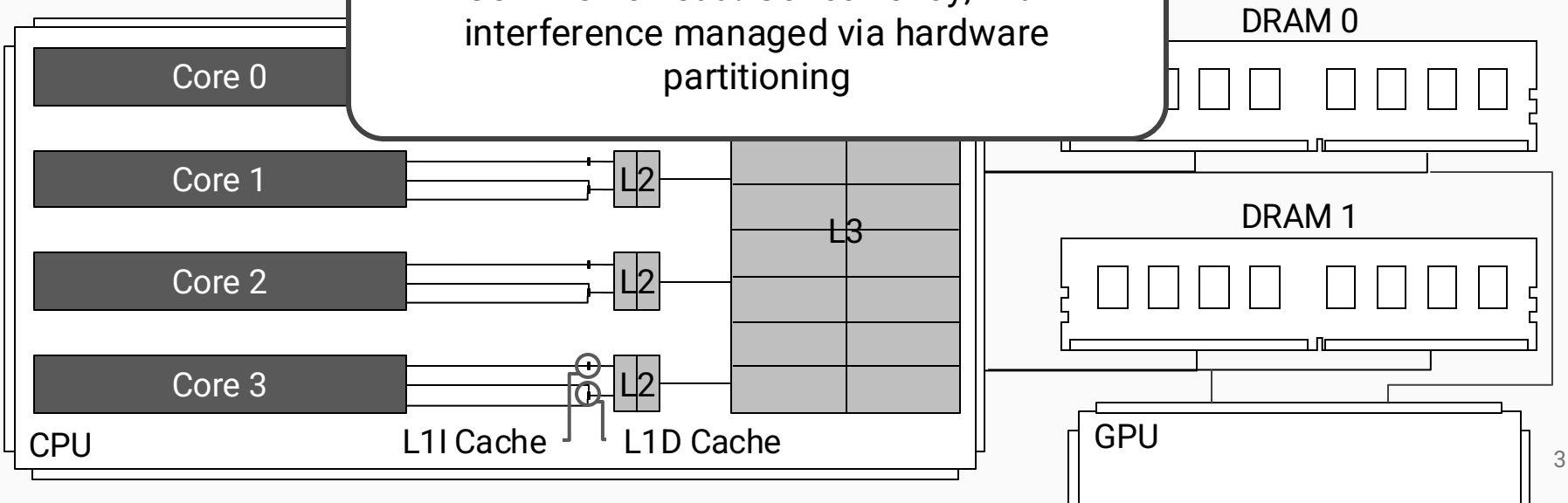
## Memory Caches and Interconnects

➔ Concurrency, with hardware partitioning

## Main Memory (DRAM)

➔ Concurrency, with hardware partitioning

**Common thread:** Concurrency, with interference managed via hardware partitioning



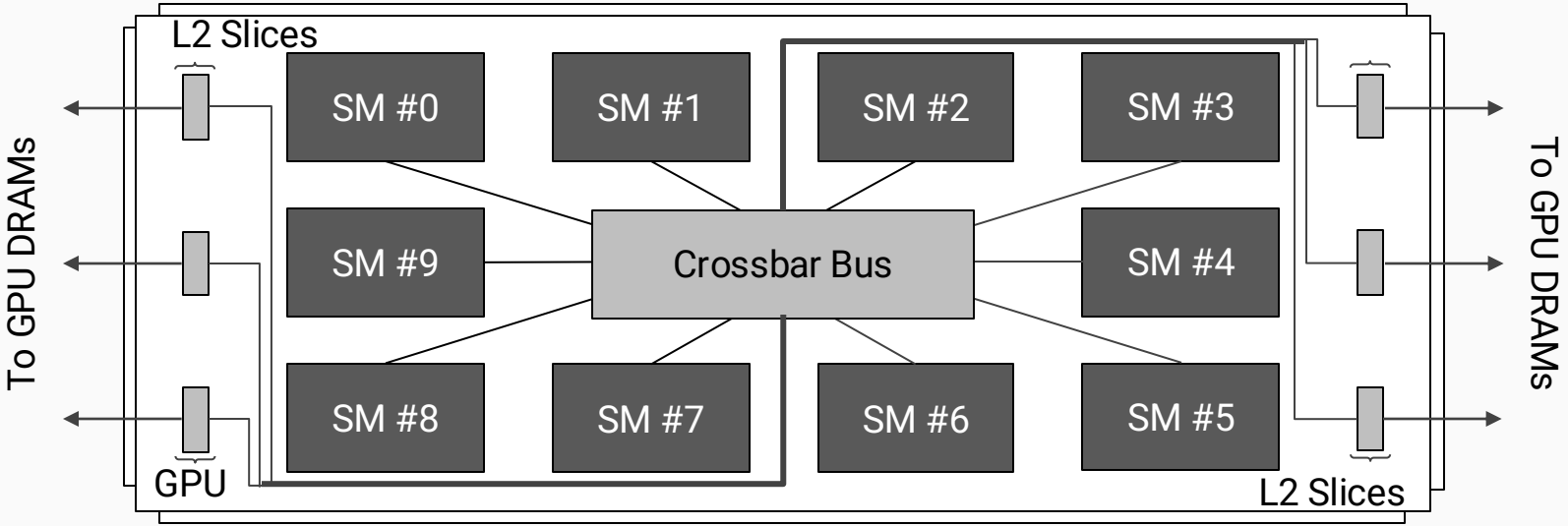
# How can we do more, with less, on the GPU?

## Compute Units

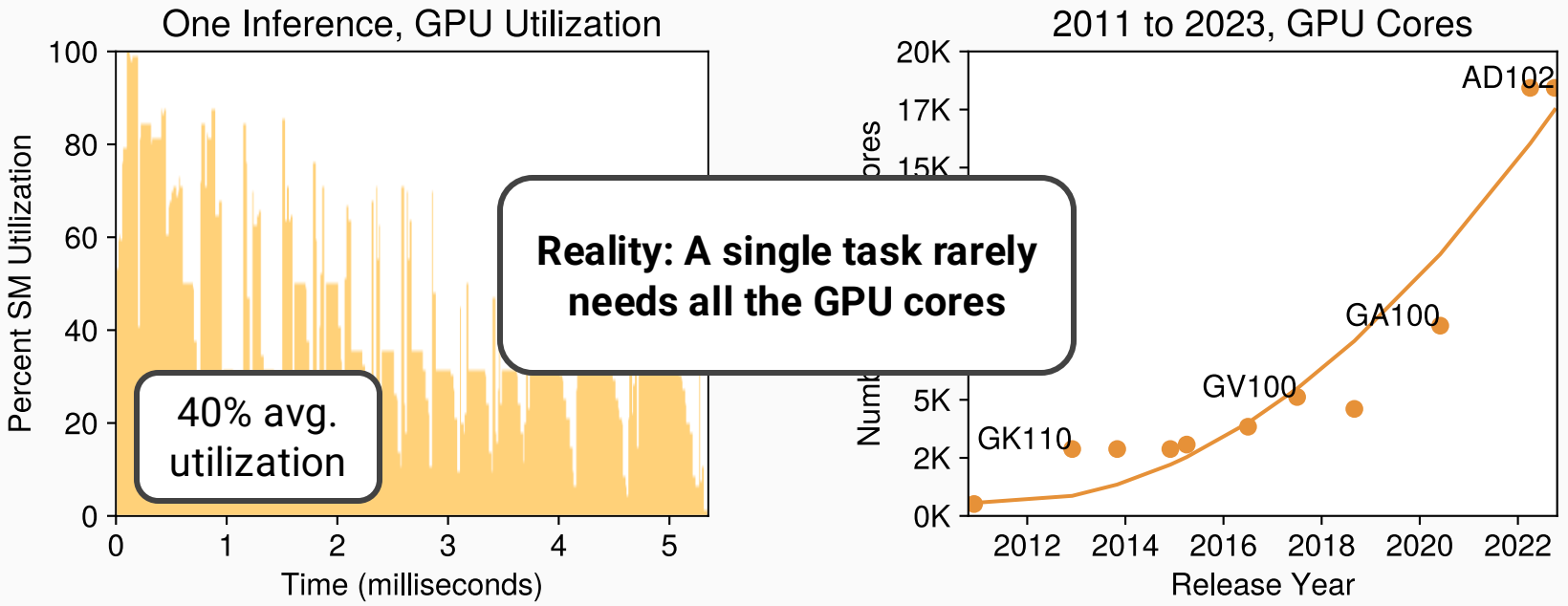
➔ **Concurrency, with hardware partitioning?**

## Memory Caches and Interconnects

➔ Concurrency, with hardware partitioning [15, 45]



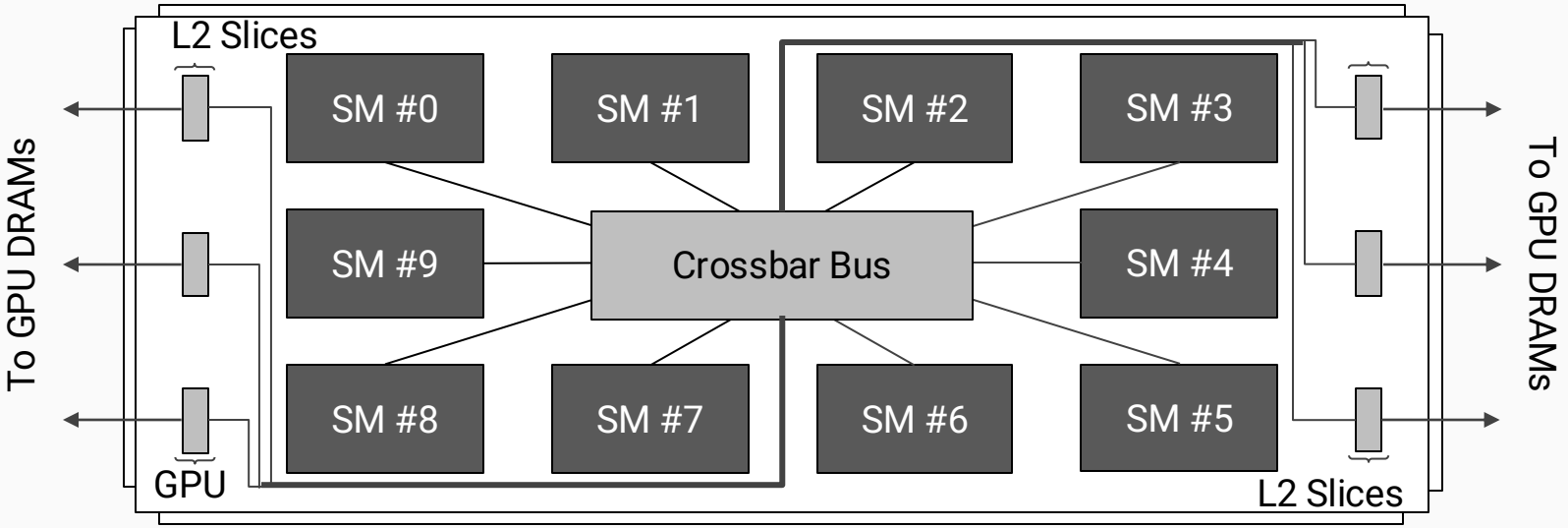
# Assumption: One task can saturate a GPU




**Reality: A single task rarely needs all the GPU cores**

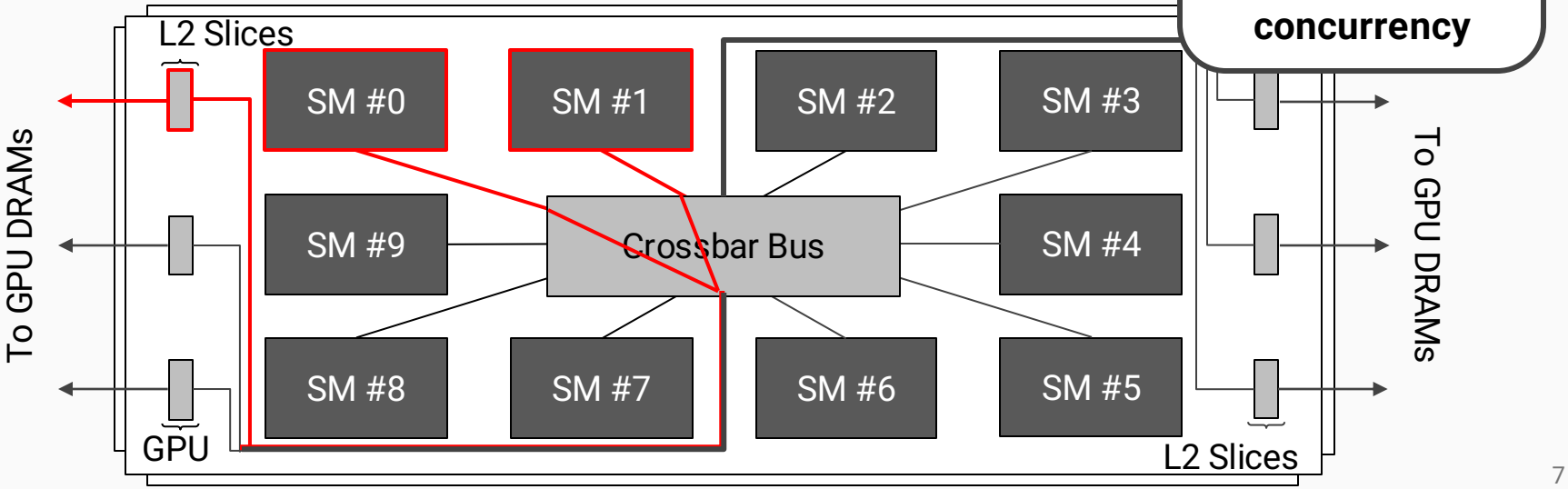
Utilization on RTX 2080 Ti (4352 CUDA cores) running YOLOv2 via Aleksei Bochkovskii's Darknet.

## Assumption: Interference worse than on-CPU



Assumption: Interference worse than on-CPU

**Reality: GPUs are highly capable of interference-free concurrency**

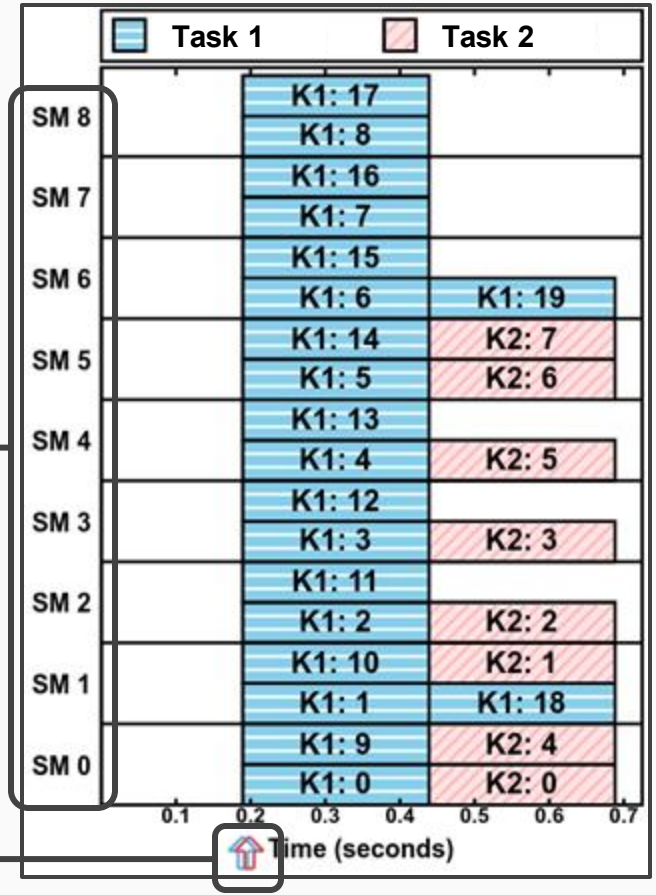


# Prior Work

A kernel is broken into *blocks*, and then the blocks are scheduled concurrently on the GPU

Each compute unit

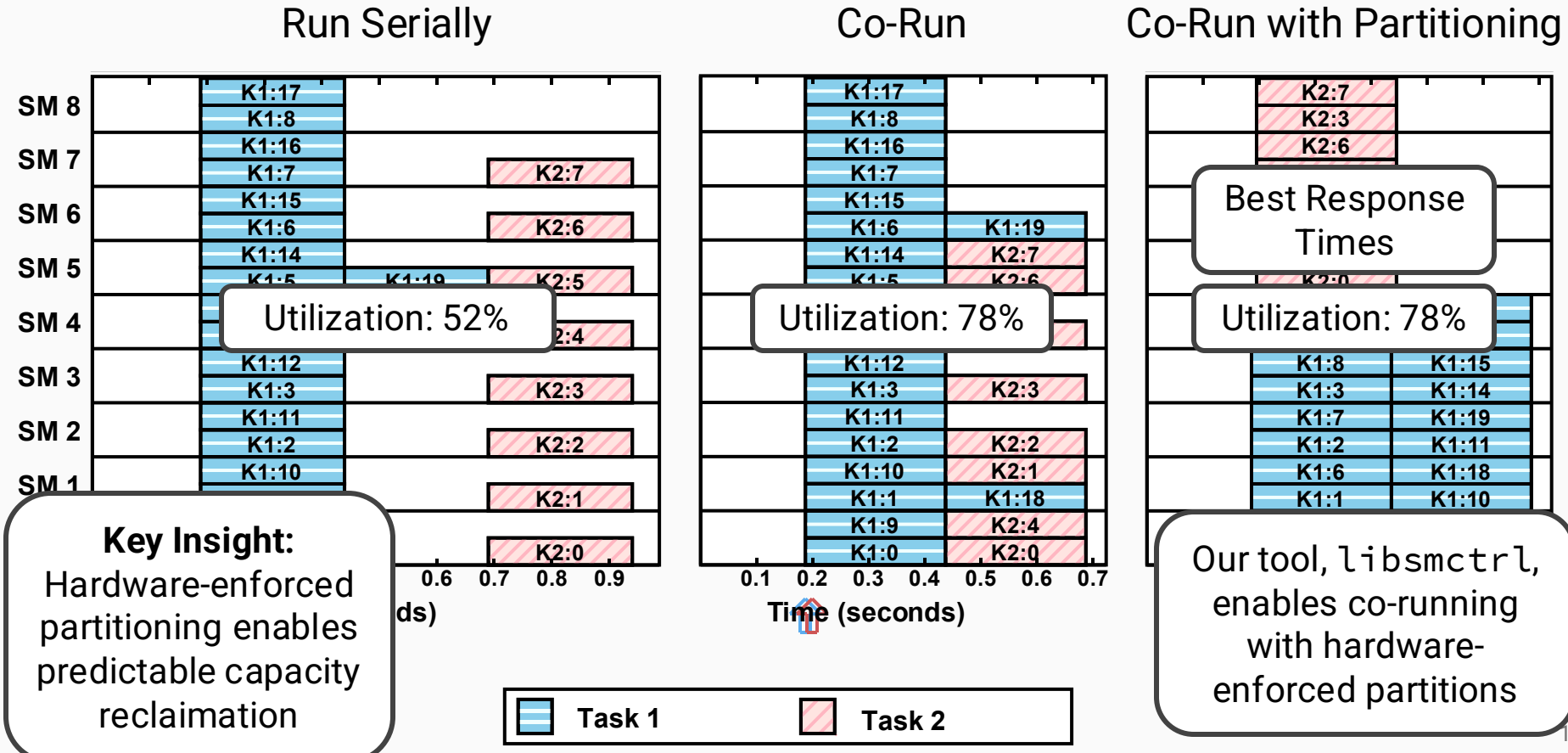
Kernel launch times




A timeline of what kernels (K_), execute which blocks (: __) on which GPU compute unit (SM)

# Prior Work

**But this has problems…**




**Key Insight:** Hardware-enforced partitioning enables predictable capacity reclaimation

Our tool, `libsmctrl`, enables co-running with hardware-enforced partitions

**Prior Work**

# Outstanding Problems with All

These issues apply to all prior academic work on GPU partitioning

This leaves few solutions for componentized systems

Does not work for **unmodified tasks**

**Compromises** address-space **isolation**

**NVIDIA's solutions,** the Multi-Process Service (MPS) and Multi-instance GPU (MiG), **do little better**

# Key Goals

Spatial partitioning for GPU compute units in composable systems that is:

**Predictable** **Efficient** **Easily Applicable**

By combining `libsmctrl` with **hardware capabilities we reveal,** we achieve all three for *any* NVIDIA GPU from the past 7 years.

# Predictable GPU Compute Partitioning

Goal 1 of 3

**Predictable Partitioning**

# If not `libsmctrl`, then what?

**NVIDIA MPS**

Can co-run unmodified tasks with “Execution Resource Provisioning.” Any GPU. **Implementation undocumented.**

**NVIDIA MiG**

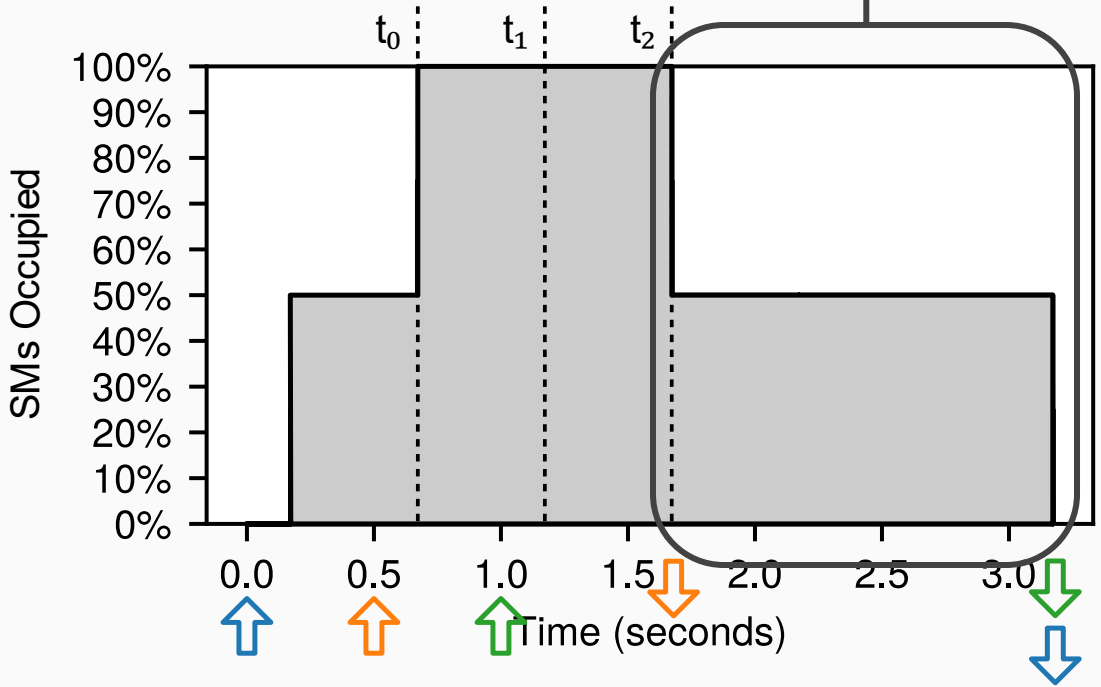
Can co-run unmodified tasks in partitions. **Datacenter-only.** Requires hardware augmentations.

## Predictable Partitioning

# Partitioning in MPS

- Called *"Execution Resource Provisioning"*
- Supports a per-task limit on proportion of resources.
- **Does not guarantee mutual exclusion.**

Two tasks, each with a 50% partition, **co-run on the same 50% of the GPU**, leaving half idle




Three tasks run, each allowed up to 50% of the GPU.

Predictable Partitioning

# Our Solution

**Key Insight:**
NVIDIA MPS, combined with `libsmctrl`-like partitioning, can predictably co-run tasks

1. Disable MPS's partitioning system.
2. Verify that MPS *does not* modify the kernel-dispatch critical path.
3. Use the same mechanism from `libsmctrl` to intercept kernels during dispatch and configure partitions of TPCs.
4. This works solely via interactions with the CUDA library, and can be done **without task modification** and **while allowing MPS to co-run tasks**.

# Efficient GPU Compute Partitioning

Goal 2 of 3

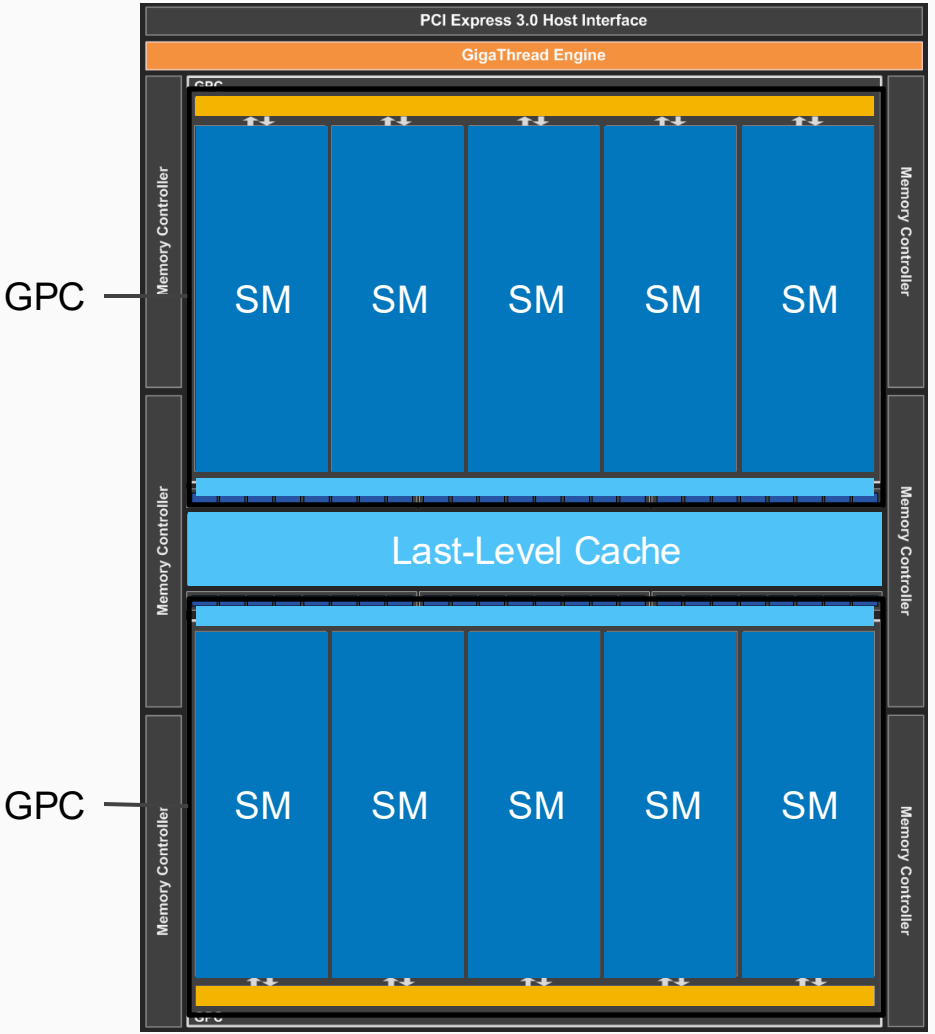
## Efficient Partitioning

# Hardware-Aware Partitioning

- Each set of five SMs is grouped into a GPC, and shares a cache with other SMs in its GPC

We want to partition on GPC boundaries to avoid slowdowns due to interference

Image from NVIDIA

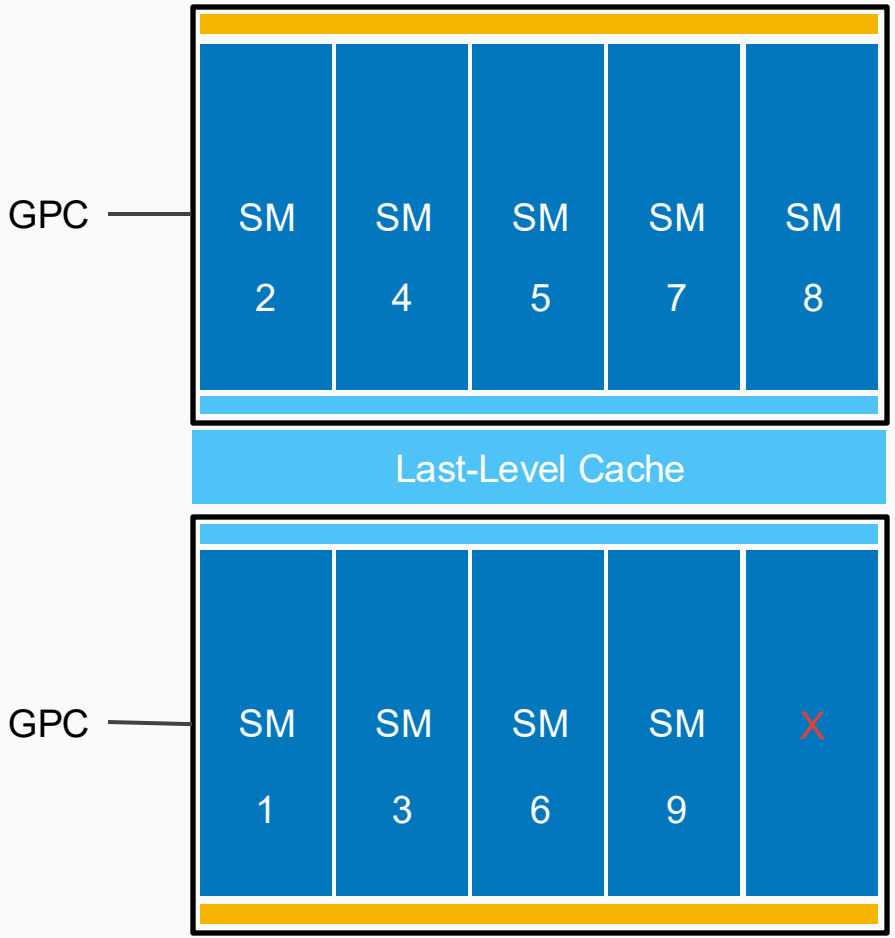


## Efficient Partitioning

# Aligning on GPC Boundaries

- SM IDs are programmed by the driver, and effectively random
- We build a tool, `nvdebug`, to extract them

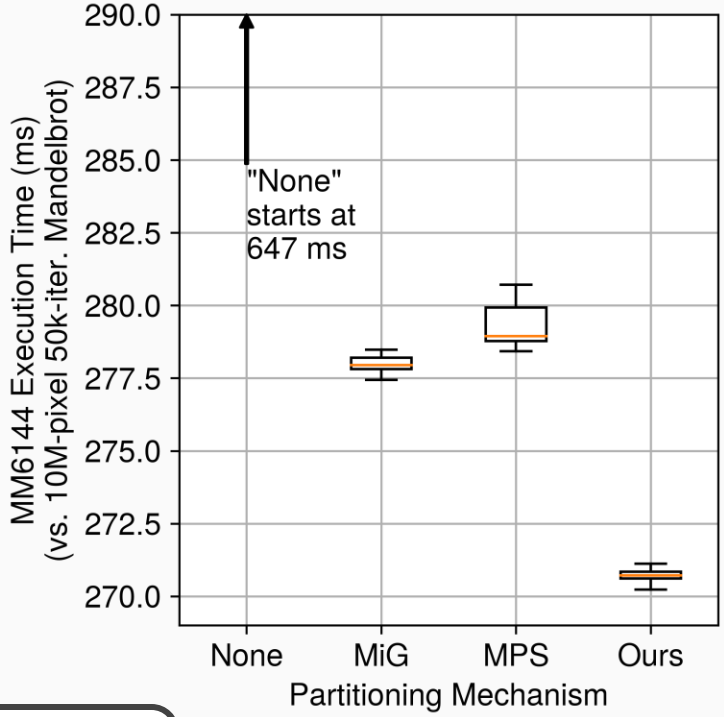
Ours in the first academic work that takes this into account



## Efficient Partitioning

# Comparison to Prior Work

- Only two prior works can enforce partitions in hardware, both from NVIDIA




Lower is better

**Key Insight:**
Awareness to hardware topology allows for greater efficiency

# Easily Applicable GPU Compute Partitioning

Goal 3 of 3

## Easily Applicable Part.

# Works in Many Situations

Works for **unmodified tasks**

**Overhead** of **<0.5 µs**

**Supports any** CUDA, Linux, or NVIDIA **driver** on x86_64 or aarch64

**Works** for any NVIDIA GPU **since 2018** (`sm_70+`)

## Easily Applicable Partitioning

# Easy to Use, `taskset`-like Tool

**Key insight:** GPU scheduling hardware changes little generation-to-generation

On Linux:

1. Download `libsmctrl`, and install CUDA + `nvdebug.ko`*
2. `make install`
3. `./nvtaskset --gpc-list 0-2 ./my-cuda-task`
4. `./nvtaskset --gpc-list 3-5 ./my-other-cuda-task`
5. Done! Both tasks will run concurrently on mutually-exclusive sets of GPCs.
6. Dynamic change for PID 1205: `./nvtaskset --gpc-list 0-1 1205`

No kernel or driver configuration, or superuser permissions needed to use.

*`nvdebug.ko` is *currently* required for `--gpu-list`, but not for `--tpc-list`

# Conclusions

**We build spatial partitioning for GPU compute units in composable systems that is:**

| **Predictable** | **Efficient** | **Easily Applicable** |
| --- | --- | --- |
| Can we co-run tasks with predictability? | Does leveraging hardware topology help? | Can we make it easy to use? |
| Yes, by **co-running via MPS** and **partitioning via libsmctrl** | Yes, allowing us to **beat even NVIDIA**'s solutions | Yes, on almost any NVIDIA GPU, **without task modification** |

# What you have to read the paper for…

NVIDIA MPS:

- How NVIDIA MPS works.
- Details on hardware implementation of MPS's Execution Resource Provisioning feature.
- Eight detailed pitfalls of MPS and suggested mitigations.
- Easy hardware-improvement opportunities for NVIDIA.

Evaluation:

- Full details on our system setup and configuration.
- Task startup overheads.
- Kernel launch overheads.
- Partitioning granularity comparison.
- Efficiency problems of NVIDIA MiG.

Regarding `nvtaskset`:

- How it applies to all tasks without modification.
- Support for partitioning tasks not originally started via `nvtaskset`.
- Full list of limitations.
- Usage examples.

\+ More details and background on everything covered in this presentation

# Thank you! Questions?

Future work:

- ➔ Support multi-GPU systems
- ➔ Provide partitioning strategies and response-time analysis
- ➔ Better Hopper support

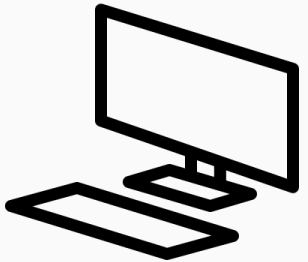
Contact:
Email: jbakita@cs.unc.edu
X: @JJBakita
Web: https://jbakita.me

Old Well, University of North Carolina at Chapel Hill, Winter 2017

# Backup Slides

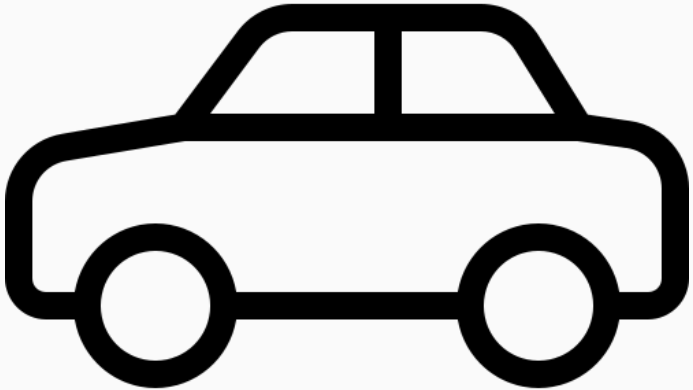
# Context

**Personal Computing**



Game + LLM

**Autonomous Driving**



Perceive Plan Control

**Cloud**

Inference for Multiple Users

Why NVIDIA GPUs?

# Processor Company Sales

**Example:**
Every self-driving car licensed in California is based on NVIDIA GPUs

| $39 Billion | $22 Billion | $31 Billion |
| --- | --- | --- |
| NVIDIA | AMD + Intel Combined | ARM + NXP + Broadcom + Qualcomm Combined |

(Revenue numbers are GAAP figures from latest fiscal quarter as of Mar 2025, and include all segments.)