

# Response-time Bounds for Concurrent GPU Scheduling

**Ming Yang** and James H. Anderson

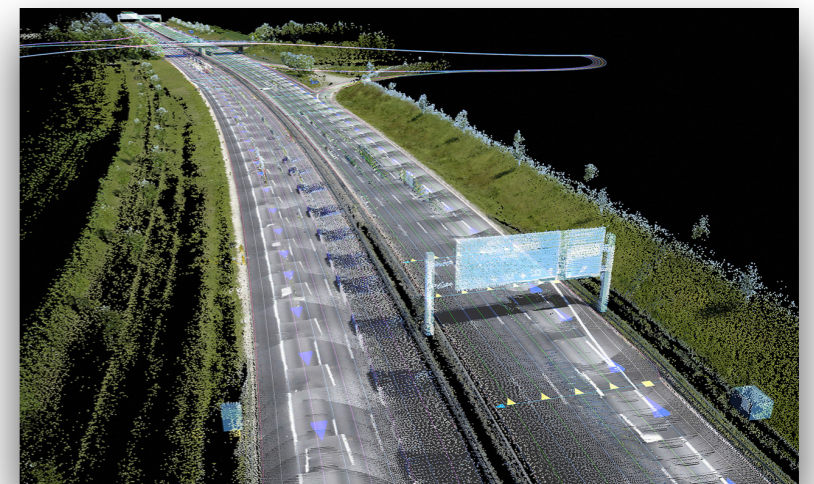
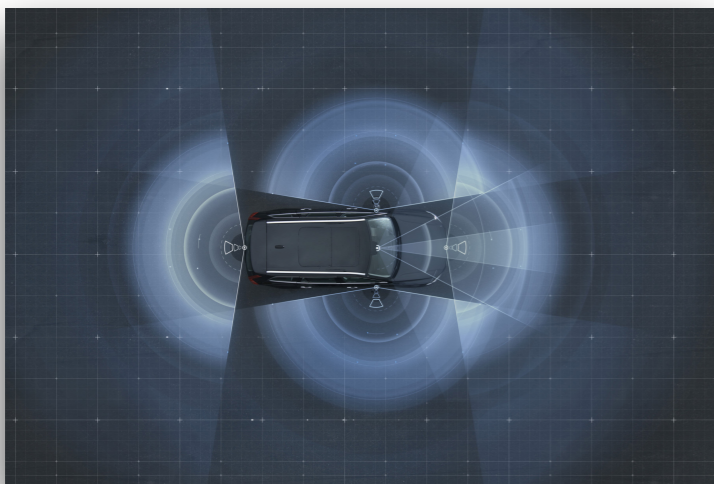


THE UNIVERSITY  
*of* NORTH CAROLINA  
*at* CHAPEL HILL



# GPU on Self-driving Vehicle

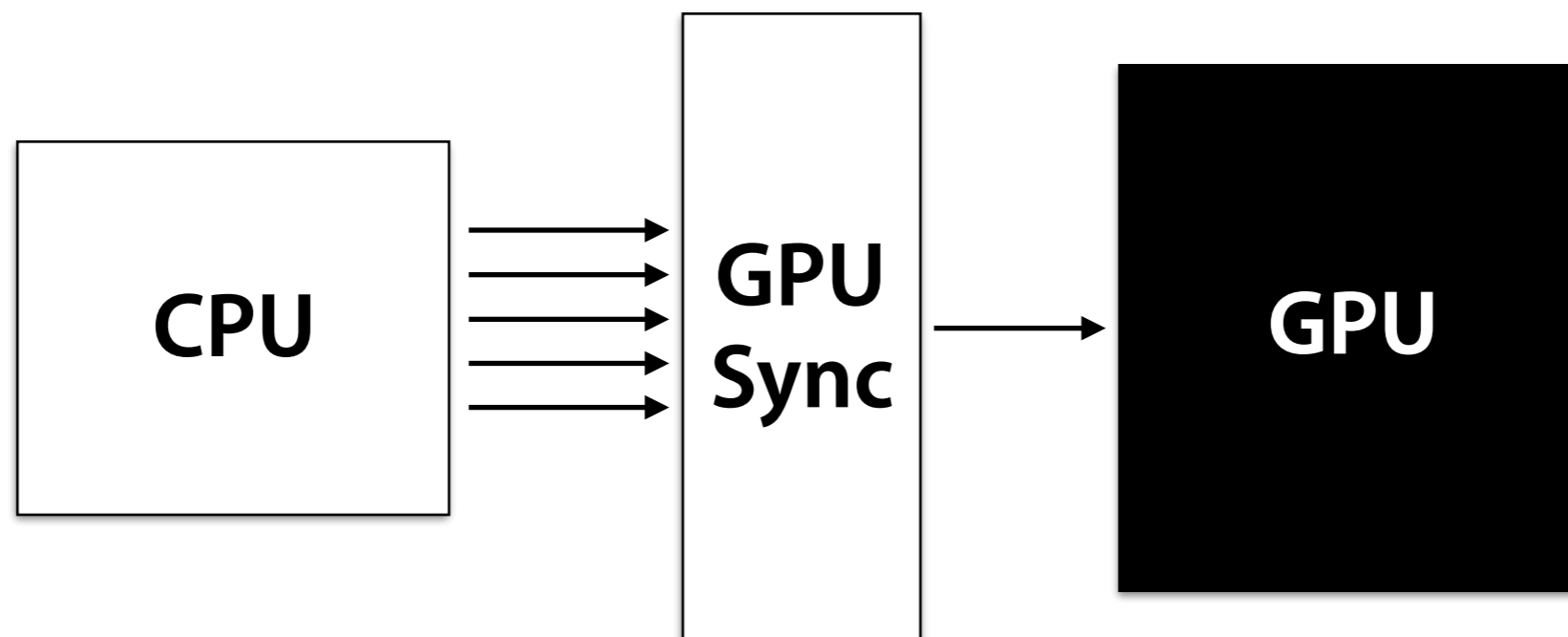
- Power efficient embedded system
- Massively parallel capability
  - sensor fusion, computer vision/DNN, HD mapping...





# Toward Real-time GPU

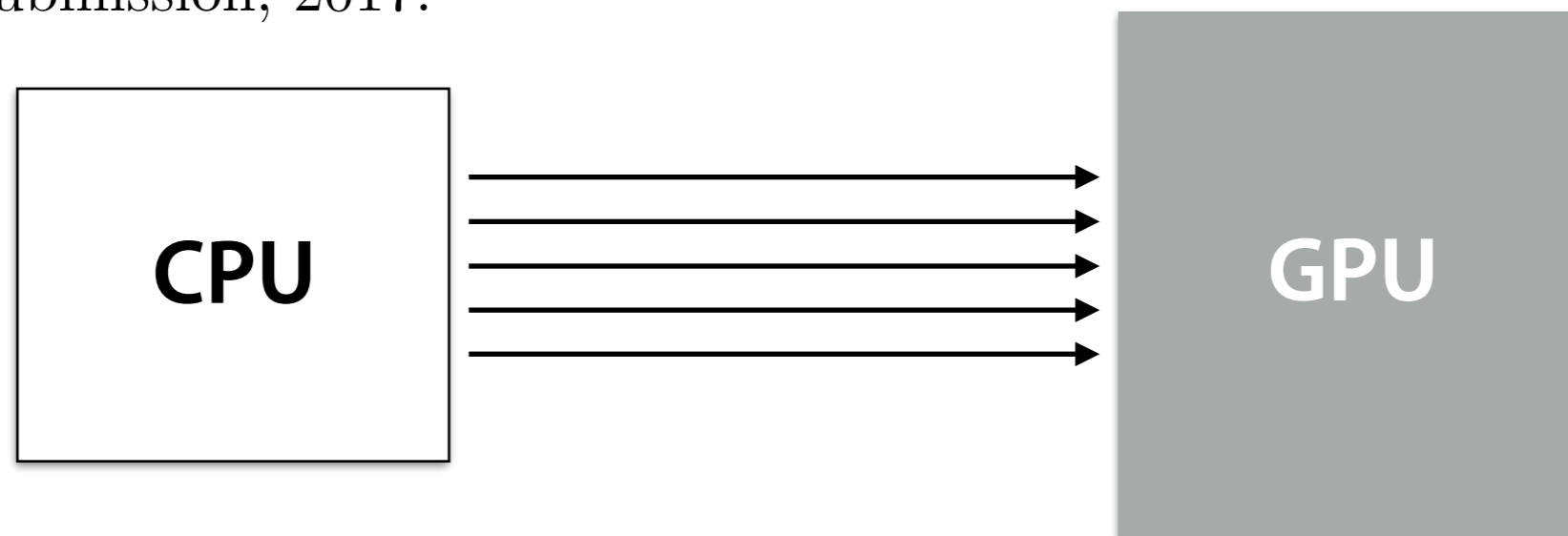
- **Proprietary** hardware, driver and library make it difficult
- Typically treated as a **single black-box resource**
  - One program accessing GPU at a time
  - Possible **capacity loss** when each individual program is incapable of occupying all GPU resources





# Toward Real-time GPU

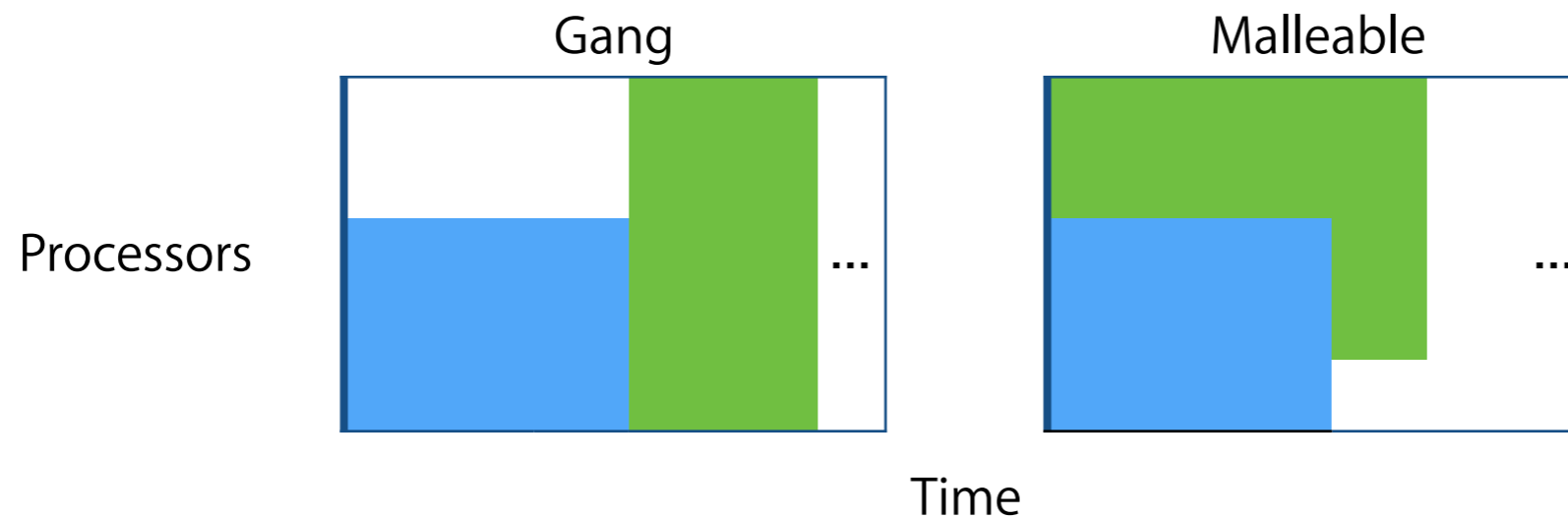
- On-going project: allow **multiple programs** to access the GPU concurrently
- inferring the **concurrent GPU scheduling rules**
  - N. Otterness, M. Yang, T. Amert, J. Anderson, and F.D. Smith, “Inferring the Scheduling Policies of an Embedded CUDA GPU”, OSPERT ’17.
  - T. Amert, N. Otterness, M. Yang, J. Anderson, and F.D. Smith , “GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed”, in submission, 2017.





# Work-in-Progress

- Having a **task model** defined for the GPU system, based on the summarized GPU scheduling rules.
- a hybrid of Gang task model and Malleable task model, at different scheduling levels



- Analyzing the **response-time bound**



# Further details in the poster session

- **Multilayered FIFO** scheduling rules
- A counter example showing the **necessary total utilization restriction** to ensure response-time bound
- And other details...

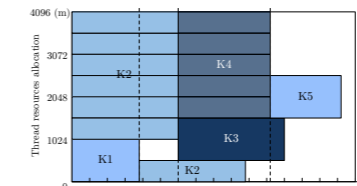
## Response-time Bounds for Concurrent GPU Scheduling

Ming Yang and James H. Anderson

### Introduction

Graphics processing units (GPUs) have been receiving increasing attention in industry as a potential solution for hosting workloads like those found in autonomous-driving use cases that require significant computational capacity. **Allowing multiple programs to access a GPU concurrently can enable the GPU to be more efficiently utilized**, if each individual program is incapable of occupying all GPU resources. In this work, we summarize the basic scheduling rules for concurrent GPU scheduling in NVIDIA GPUs. **We define a task model for GPU scheduling based on these scheduling rules. In ongoing work, we are attempting to obtain response-time bounds for tasks under this model.**

### Scheduling Rules



Each kernel (GPU program) is issued as a **group of threads** that execute on a GPU. Threads of one kernel are arranged into **multiple same-size blocks**. Each kernel is a **grid of such blocks**. The number of threads per block (*block size*) and the number of blocks per grid (*grid size*) of a kernel are defined via parameters passed to the CUDA call that launches that kernel. The basic GPU scheduling rules are as follows:

**R1. A block of a kernel in the primary queue is assigned to the GPU for execution if:**

- that kernel is at the head of the primary queue, and
- the number of unoccupied threads on the GPU is at least the kernel's block size.

**R2. A kernel is enqueued on the primary queue when it becomes the head of its stream queue.**

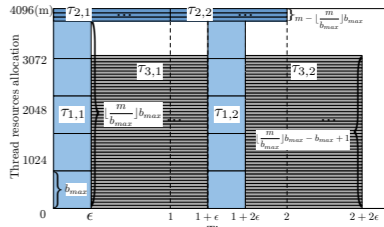
### System Model

$$\tau_i = (C_i, T_i, g_i, b_i)$$

$g_i$	# of blocks per task (grid size)
$b_i$	# of threads per block (block size)

When  $g_i = 1$ , it's the Gang task model. When  $b_i = 1$ , it's the Malleable task model.

### System Utilization Restriction



We constructed a task system  $\tau = \{\tau_1, \tau_2, \tau_3\}$  to show that if the total utilization  $U_{sum} \geq m - b_{max} + 1$ , where  $b_{max}$  is the maximum # of threads per block, then response times may be unbounded. Here are the details of this task system: Let  $k = \lfloor \frac{m}{b_{max}} \rfloor b_{max}$  for convenience, we have

$$\tau_1 = (\epsilon \cdot k, 1, \frac{k}{b_{max}}, b_{max})$$

$$\tau_2 = (m - k, 1, m - k, 1)$$

$$\tau_3 = (k - b_{max} + 1, 1, k - b_{max} + 1, 1)$$

Although the  $\lim_{\epsilon \rightarrow 0^+} U_{sum} = m - b_{max} + 1$ , job  $\tau_3$ 's response time is  $R_{3,j} = 1 + \epsilon \cdot j$ , where  $j \geq 1$ .

### Conclusions

In this paper, we defined a task model for GPU scheduling and showed that response times can be unbounded in this model if total utilization exceeds a certain limit. In ongoing work, we are attempting to prove that response times are indeed bounded in this model if total utilization is at most this limit.

N. Otterness, M. Yang, T. Amert, J. Anderson, and F.D. Smith. Inferring the scheduling policies of an embedded CUDA GPU. In OSPERT '17.  
T. Amert, N. Otterness, M. Yang, J. Anderson, and F.D. Smith. GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed, in submission, 2017.

Ming Yang