Doing More with Less: Cost-Effective Infrastructure for Automotive Vision Capabilities

University of North Carolina Chapel Hill
PI: Prof. James Anderson, co-PIs: Prof. Sanjoy Baruah, Prof. Alexander Berg & Dr. Shige Wang
Students: Tanya Amert, Nathan Otterness, Ming Yang

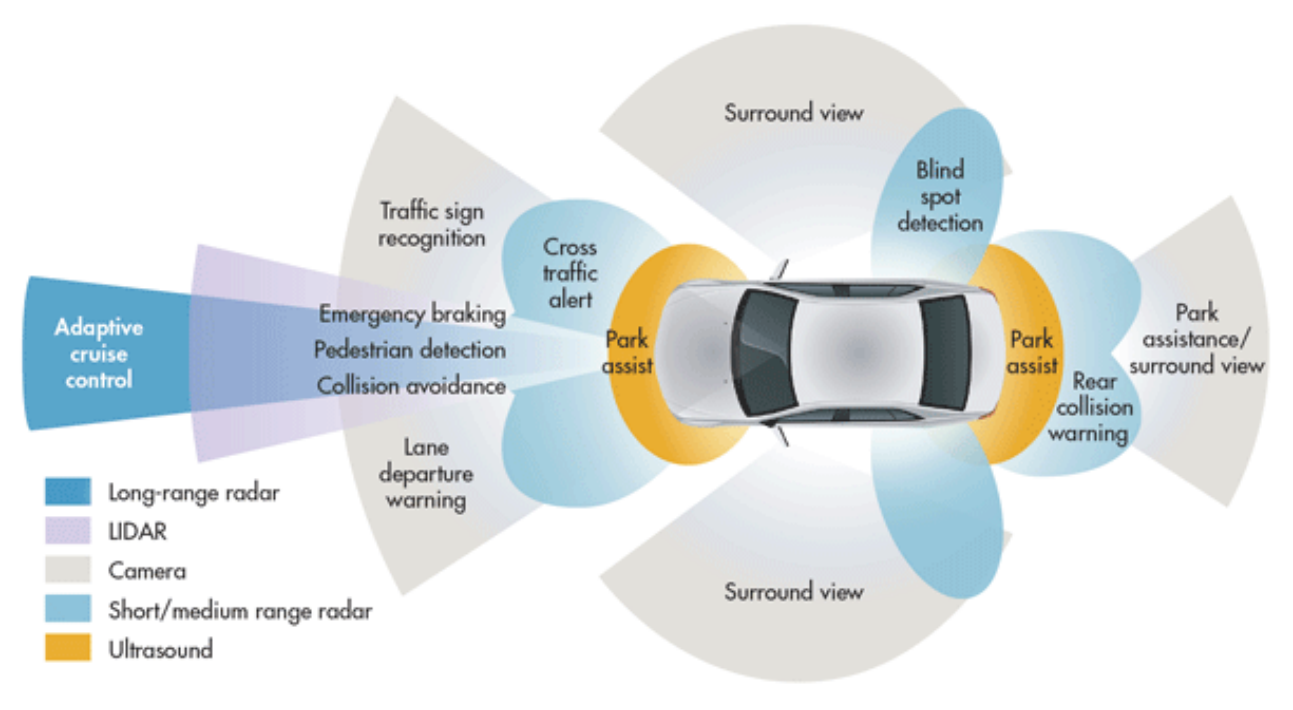
Motivation

Many safety-critical cyber-physical systems rely on advanced sensing capabilities to react to changing environmental conditions. However, cost-effective deployments of such capabilities have remained elusive. Such deployments will require software infrastructure that enables multiple sensor-processing streams to be multiplexed onto a common hardware platform at reasonable cost, as well as tools and methods for validating that required processing rates can be maintained.

Problem

Currently, advanced driver assistance system (ADAS) capabilities have only been implemented in prototype vehicles using hardware, software, and engineering infrastructure that is very expensive. Prototype hardware commonly includes multiple high-end CPU and GPU chips and expensive LIDAR sensors.

Focusing directly on judicious resource allocation, this project seeks to enable more economically viable implementations. Such implementations can reduce system cost by utilizing cameras in combination with low-cost embedded multicore CPU+GPU platforms.



http://roboticsandautomationnews.com/wp-content/uploads/2016/09/adas-illustration.gif

Objectives

This project focuses on three principal objectives:

- New implementation methods for multiplexing disparate image-processing streams on embedded multicore platforms augmented with GPUs.
- New analysis methods for certifying required stream-processing rates.
- New computer-vision methods for constructing image-processing pipelines.

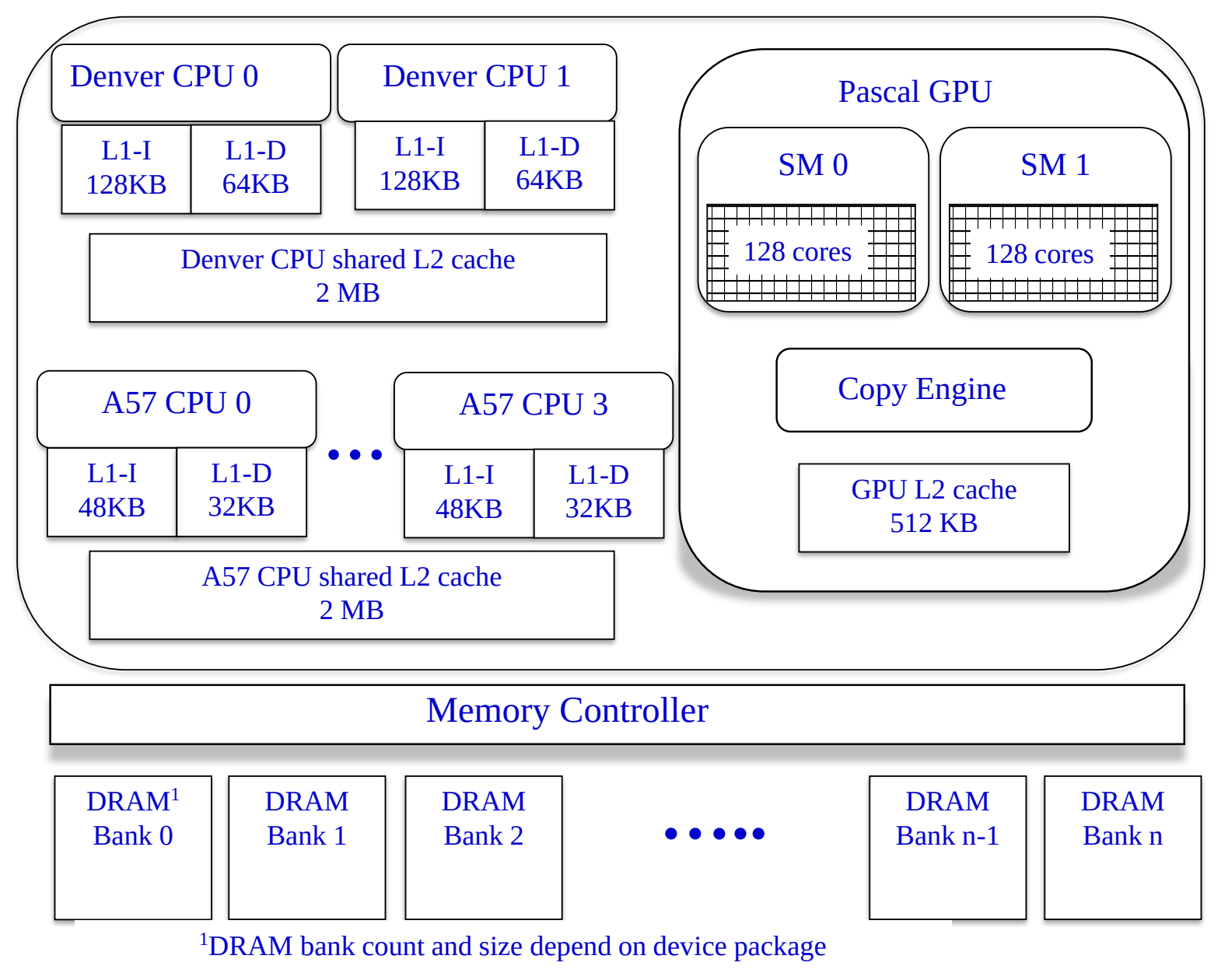
Activities

- Automotive Cyber-Physical Systems graduate-level course at UNC Chapel Hill. (http://www.cs.unc.edu/~anderson/teach/comp790a/)
- Autonomous Driving: Moving from Theory to Practice graduate-level course at UNC Chapel Hill. (http://need4speed.web.unc.edu, https://cs.unc.edu/~anderson/teach/comp790car/)
- G. Elliott, K. Yang, and J. Anderson, "Supporting Real-Time Computer Vision Workloads using OpenVX on Multicore+GPU Platforms", RTSS 2015.
- K. Yang, G. Elliott, and J. Anderson, "Analysis for Supporting Real-time Computer Vision Workloads using OpenVX on Multicore+GPU Platforms", RTNS 2015.
- W. Liu, D. Anguelov, E. Erhan, C. Szegedy, S. Reed, C. Fu, A. Berg, "*SSD: Single Shot MultiBox Detector*", ECCV 2016.
- N. Otterness, V. Miller, M. Yang, J. Anderson, and F.D. Smith, "GPU Sharing for Image Processing in Embedded Real-Time Systems", OSPERT 2016.
- N. Otterness, M. Yang, S. Rust, E. Park, J. Anderson, F.D. Smith, A. Berg, S. Wang, "*An Evaluation of the TX1 for Supporting Real-Time Computer-Vision Workloads*", RTAS 2017
- N. Otterness, M. Yang, T. Amert, J. Anderson, and F.D. Smith, "Inferring the Scheduling Policies of an Embedded CUDA GPU", OSPERT 2017.
- M. Yang and J. Anderson, "Response-Time Bounds for Concurrent GPU Scheduling", ECRTS-WiP 2017.
- T. Amert, N. Otterness, M. Yang, J. Anderson, and F.D. Smith , "GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed", RTSS 2017.
- N. Otterness, M. Yang, T. Amert, J. Bakita, J. Anderson, and F. D. Smith, "Implicit GPU Synchronization: A Barrier to Real-Time CUDA Workloads", RTAS 2018, in submission.

Supporting Real-Time Computer Vision Workloads

Platform

We are focusing on real-time systems where significant computing capacity must be provided with minimum monetary cost and size, weight, and power (SWaP). NVIDIA's Jetson TX2 fits these constraints.

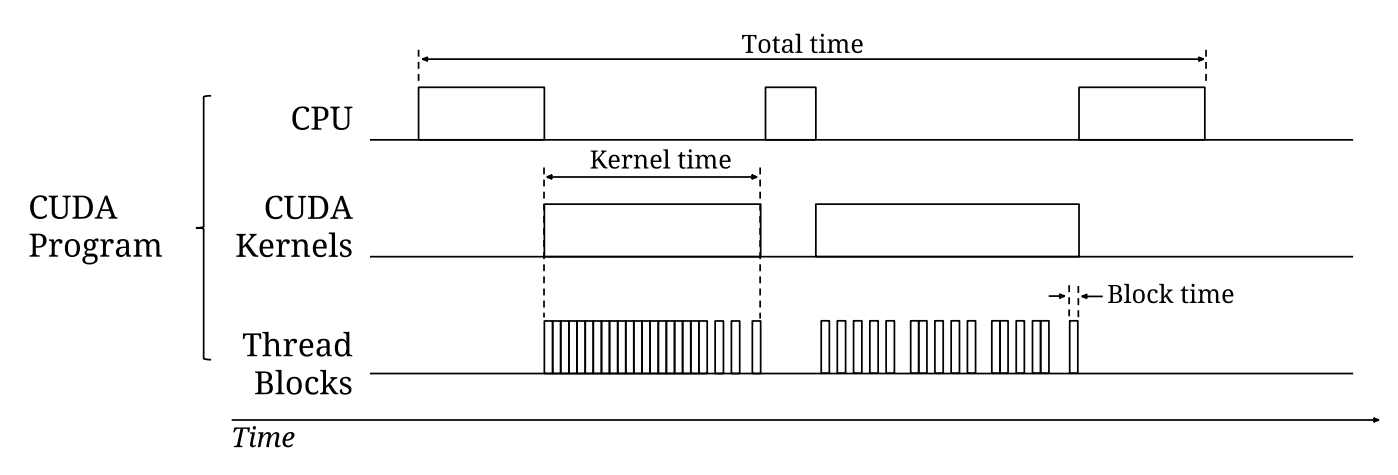


Jetson TX2:

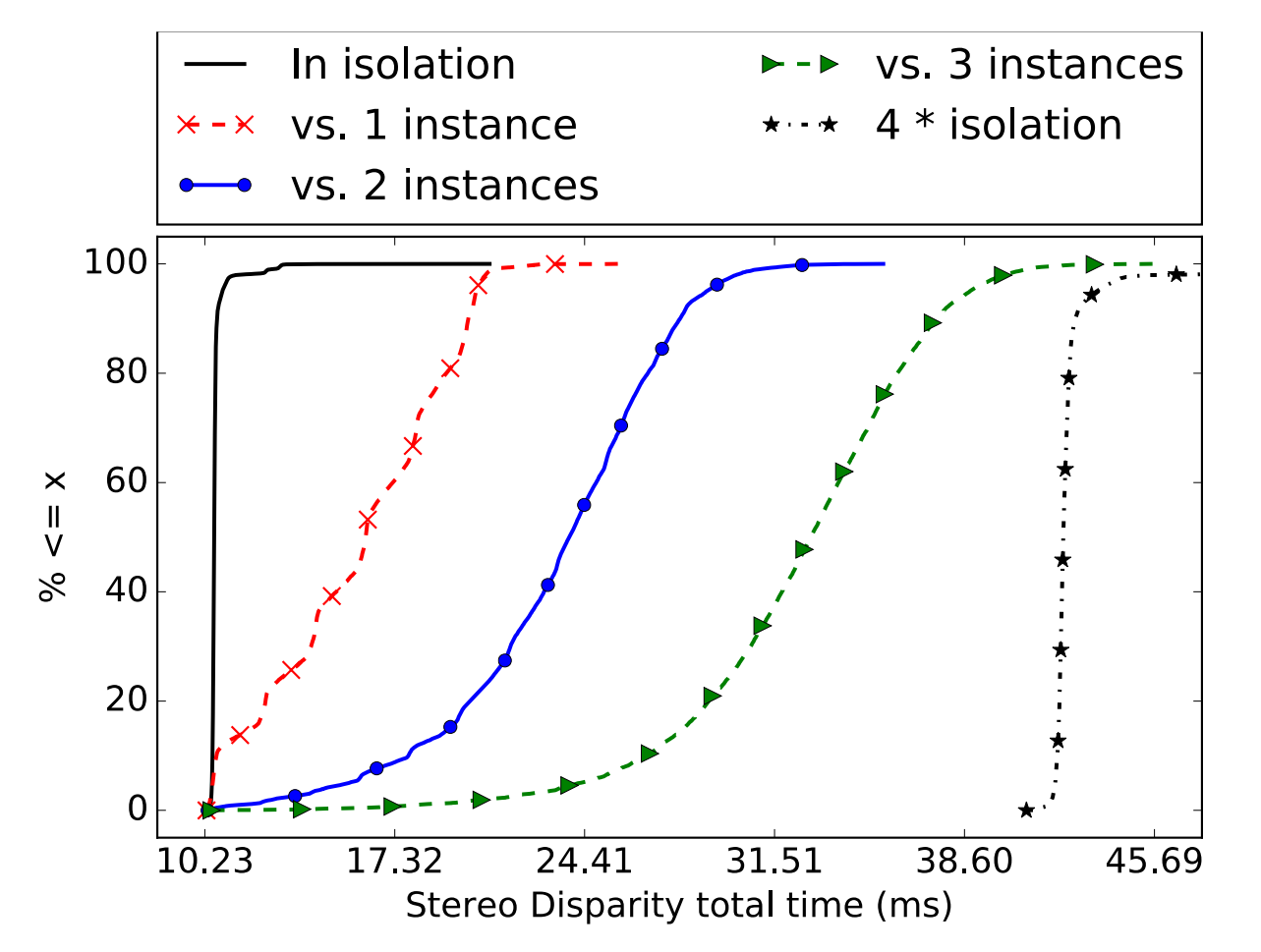
- 600 USD
- a leading multicore+GPU solution
- Marketed by NVIDIA as "The embedded platform for autonomous everything"
- A single-board computer containing:
  - quad-core 64-bit ARM CPU + dual-core Denver CPU
  - 8GB of DRAM,
  - an integrated GPU
- The DRAM is shared between the host CPU and GPU.

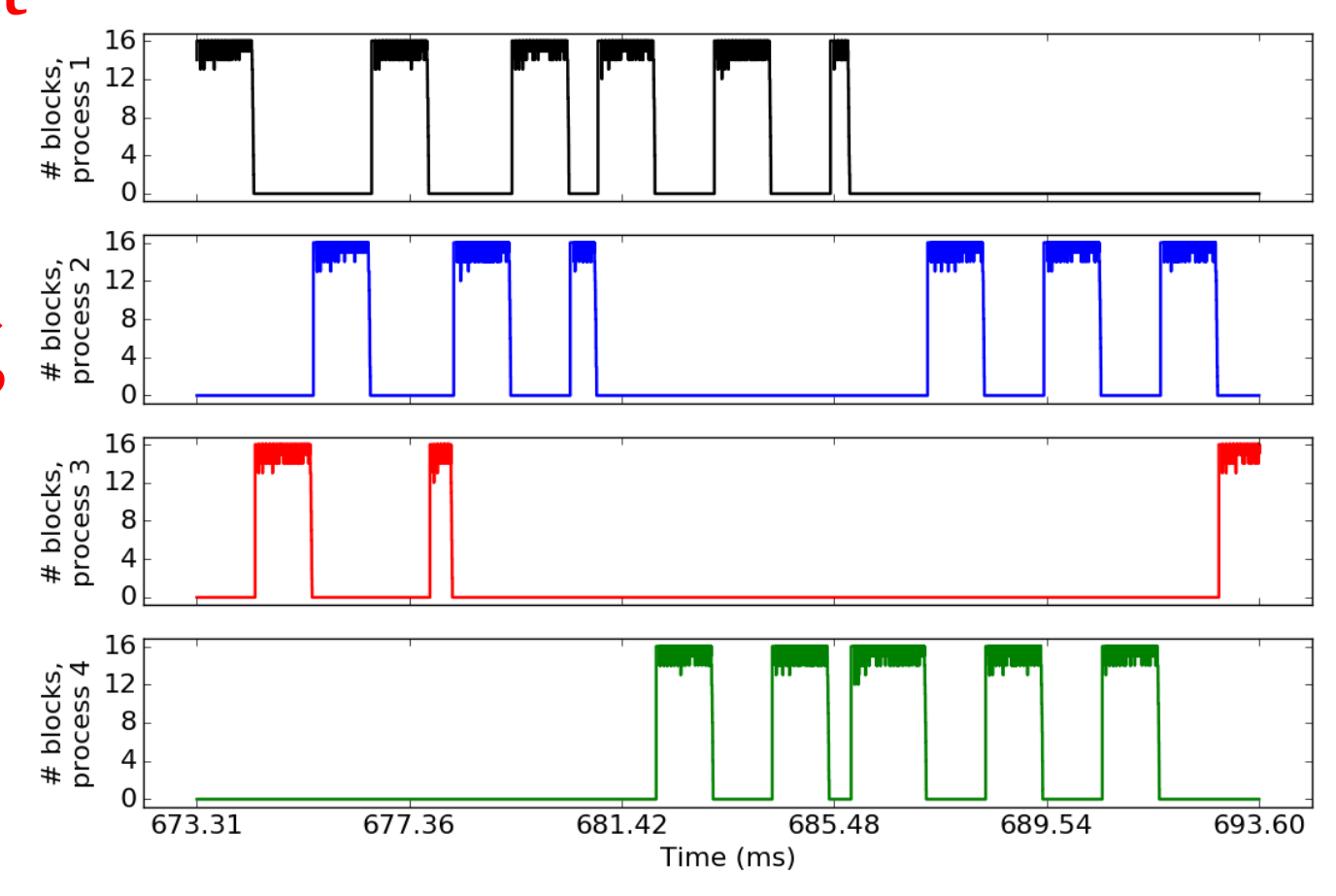
Multiprocess Co-Scheduling

- **Methodology**: Run GPU-using programs in separate processes, record start and end times of thread blocks.



- **Observations**:
  - GPU coscheduling can reduce total time compared to sequential execution.
  - Block times are minimally affected by coscheduling in this case.
  - Coscheduled processes do not truly share the GPU, but are multiprogrammed.
- Our observations imply that using multiple threads within a single process have more potential to improve utilization.





Inferring GPU Scheduling Behavior

Motivation:

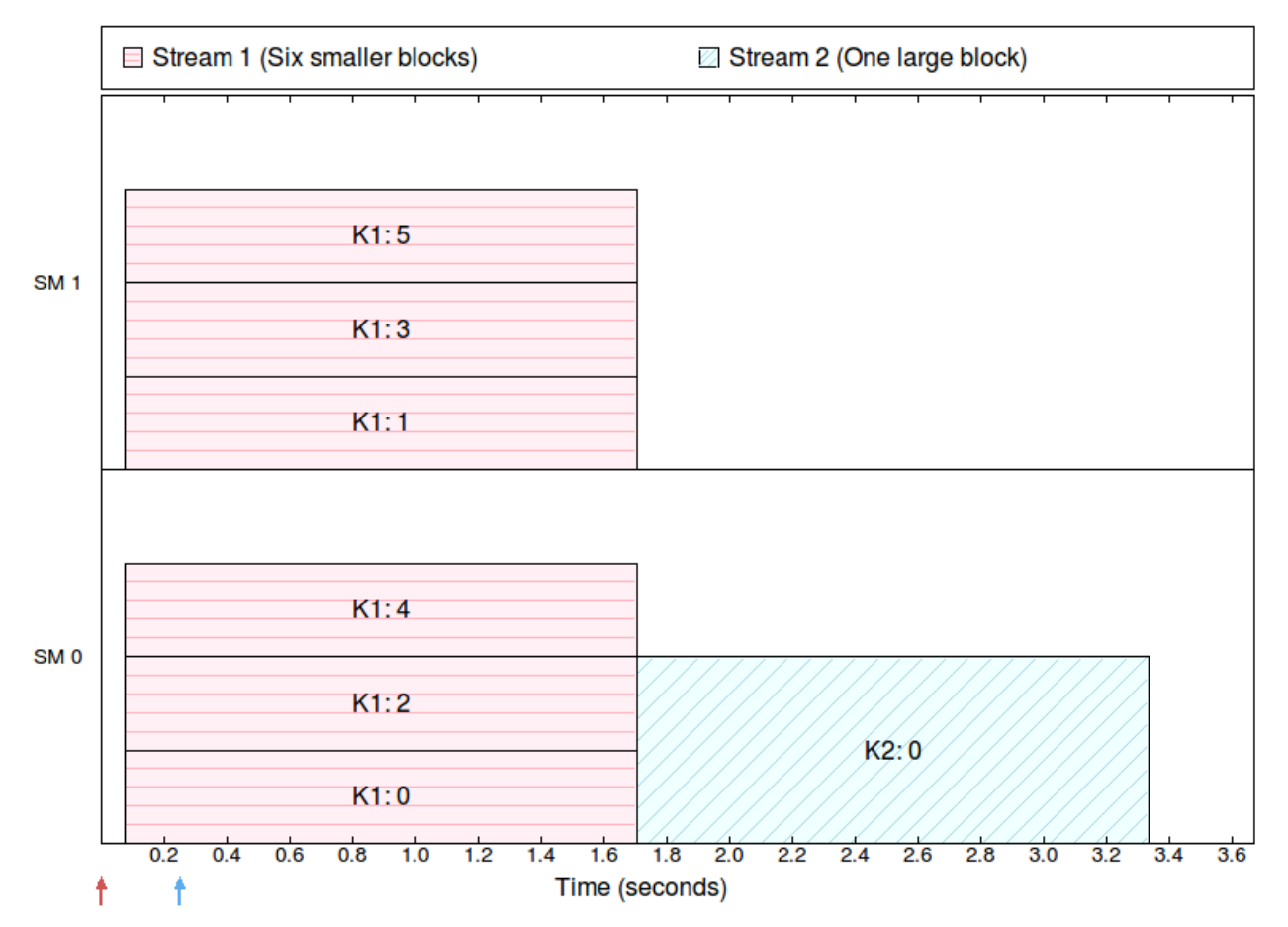
- Scheduling of GPU programs can result in wasted capacity

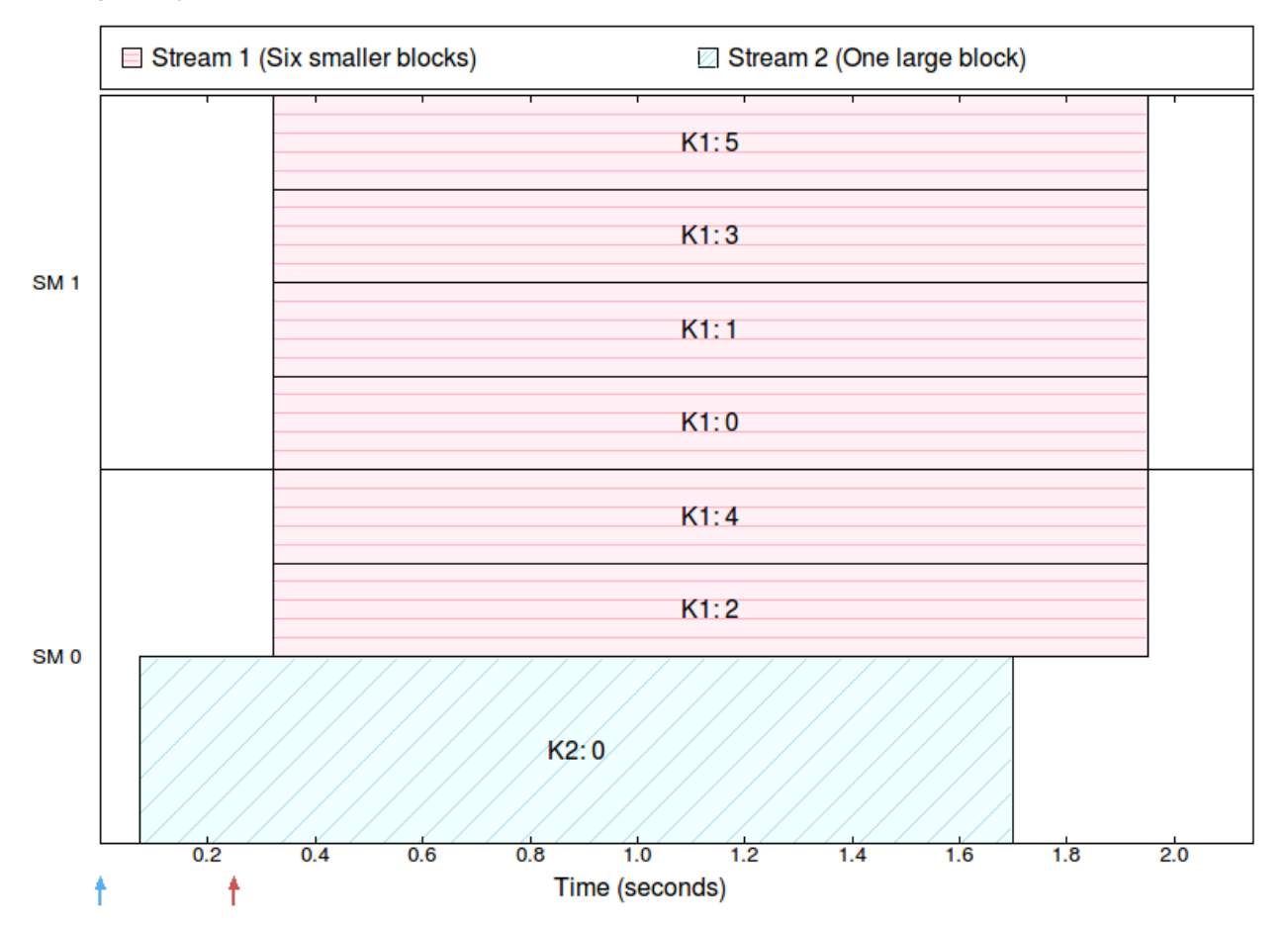
Methodology:

- Designed an experimentation framework to infer GPU scheduling behavior
- Developed rules to describe scheduling behavior seen in experiments

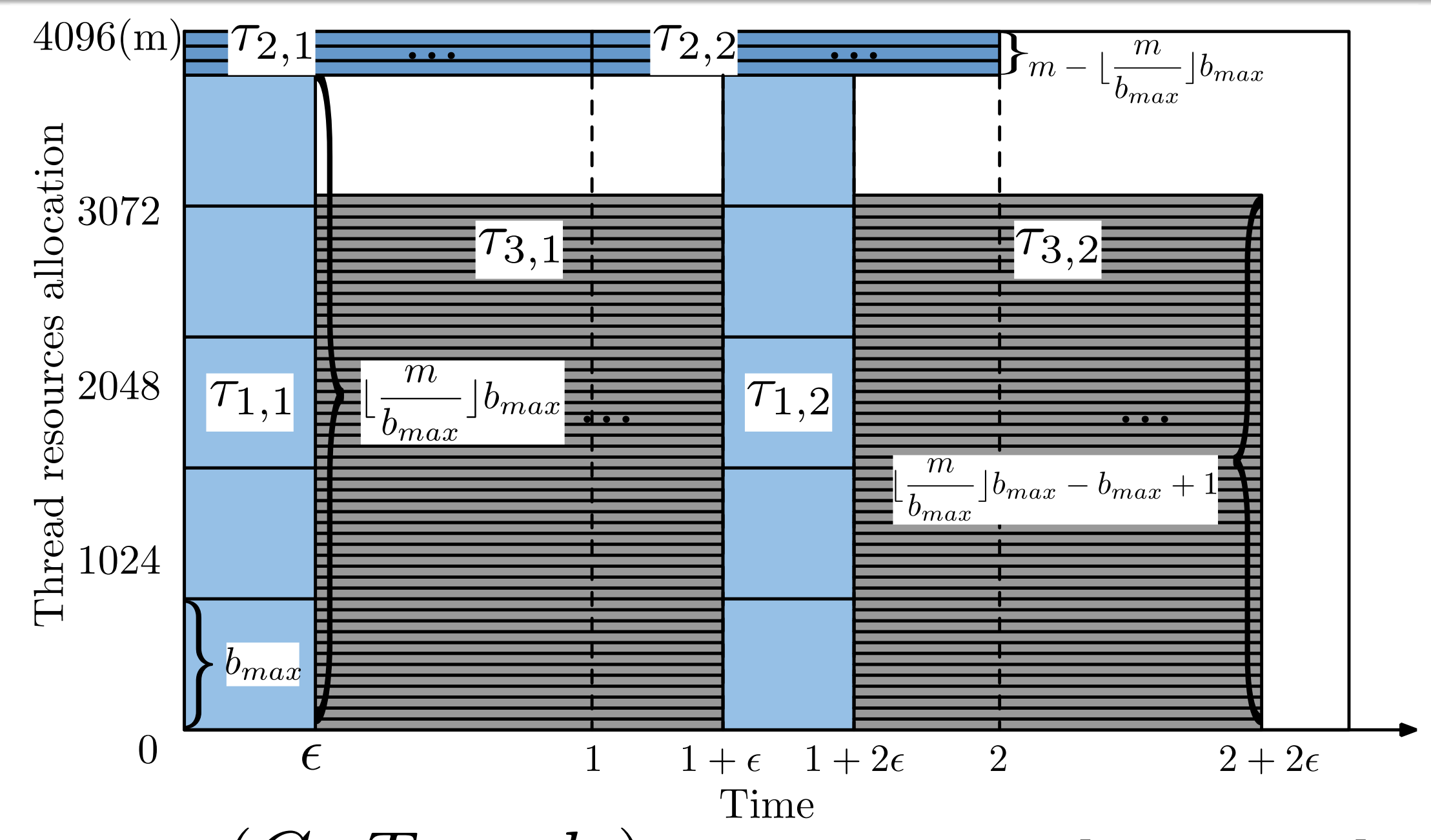
Future work:

- Write middleware to reorder GPU work
- Develop schedulability theory





Total Utilization Restriction


4096(m)
3072
2048
1024
0
Thread resources allocation
$\tau_{2,1}$
$\tau_{2,2}$
$m - \lfloor \frac{m}{b_{max}} \rfloor b_{max}$
$\tau_{3,1}$
$\tau_{3,2}$
$\tau_{1,1}$
$\tau_{1,2}$
$\lfloor \frac{m}{b_{max}} \rfloor b_{max}$
$\lfloor \frac{m}{b_{max}} \rfloor b_{max} - b_{max} + 1$
b_{max}
ϵ
1
$1+\epsilon$
$1+2\epsilon$
2
$2+2\epsilon$
Time

We use $\tau_i = (C_i, T_i, g_i, b_i)$ to represent each GPU task, where C_i indicates the job execution time, T_i indicates the task period, g_i indicates the # of blocks, and b_i indicating # of threads per block. 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:

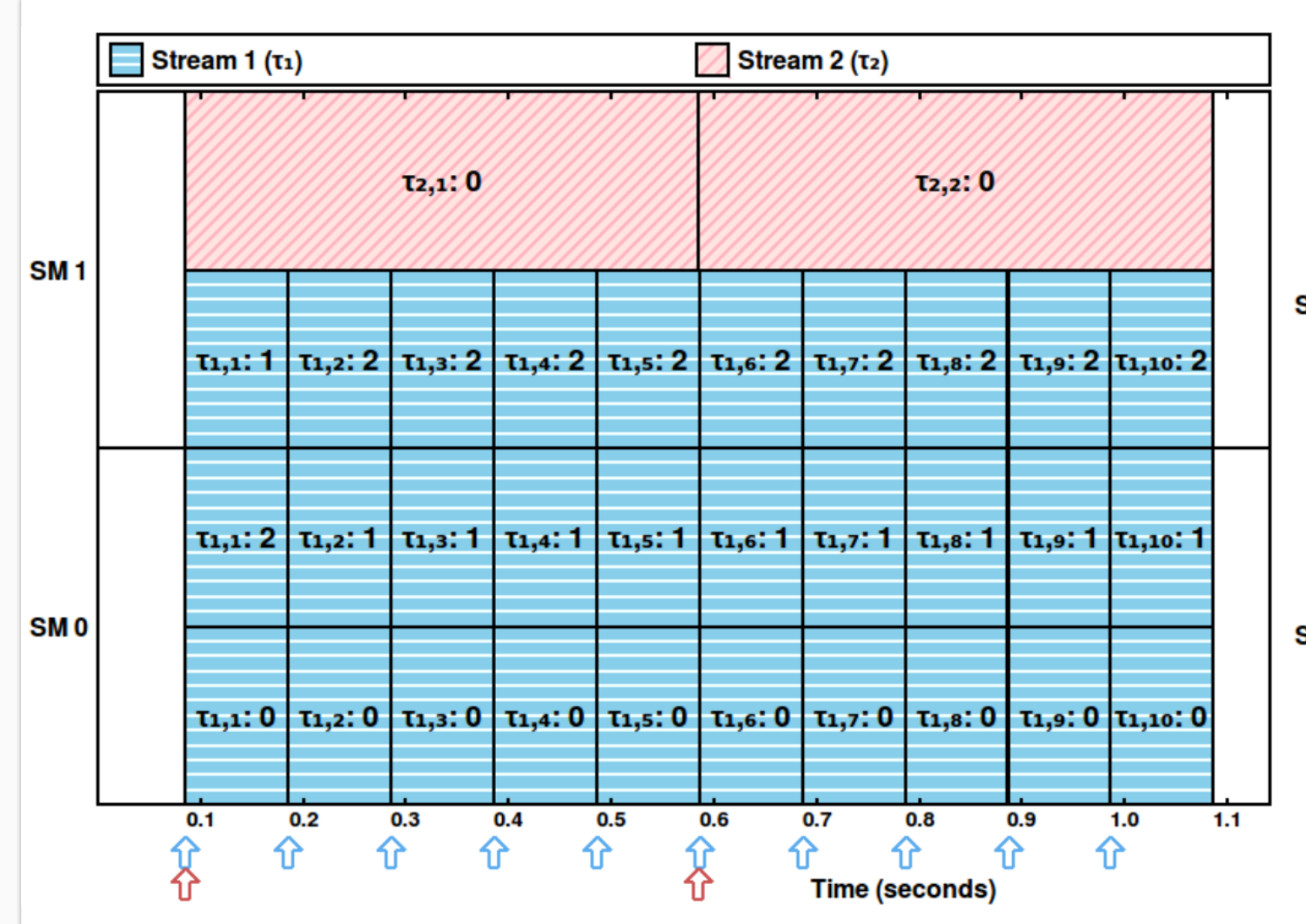
Let $k = \lfloor \frac{m}{b_{max}} \rfloor b_{max}$ for convenience, we have

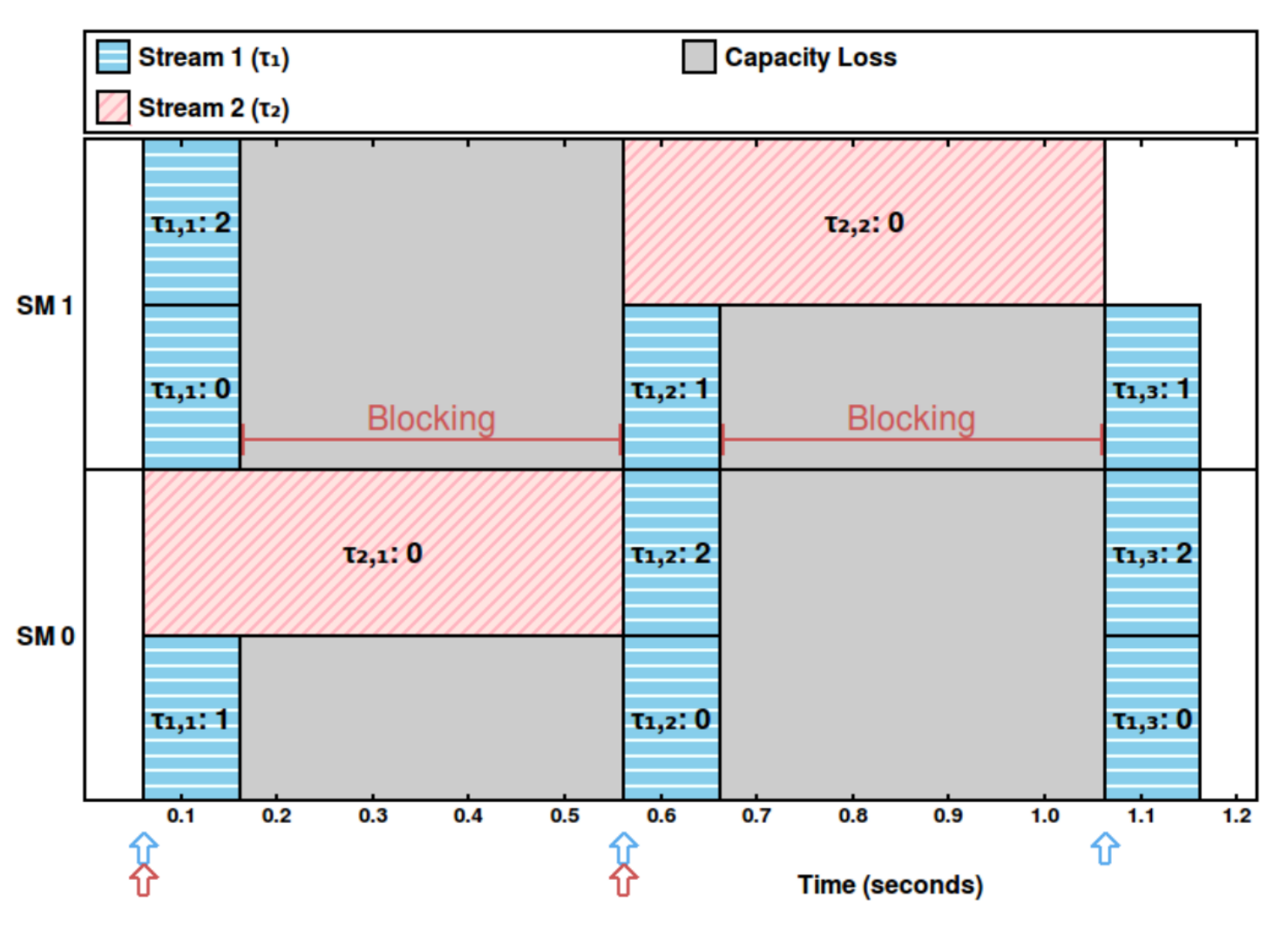
$$\tau_1 = (\epsilon \cdot k, 1, \frac{k}{b_{max}}, b_{max}) \qquad \tau_2 = (m-k, 1, m-k, 1)$$

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

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

Implicit Synchronization





- GPU synchronization blocks GPU operations, causing capacity loss.
- The CUDA API can cause unexpected implicit synchronization.
- Future middleware may reduce blocking by re-scheduling some implicit-sync API calls.

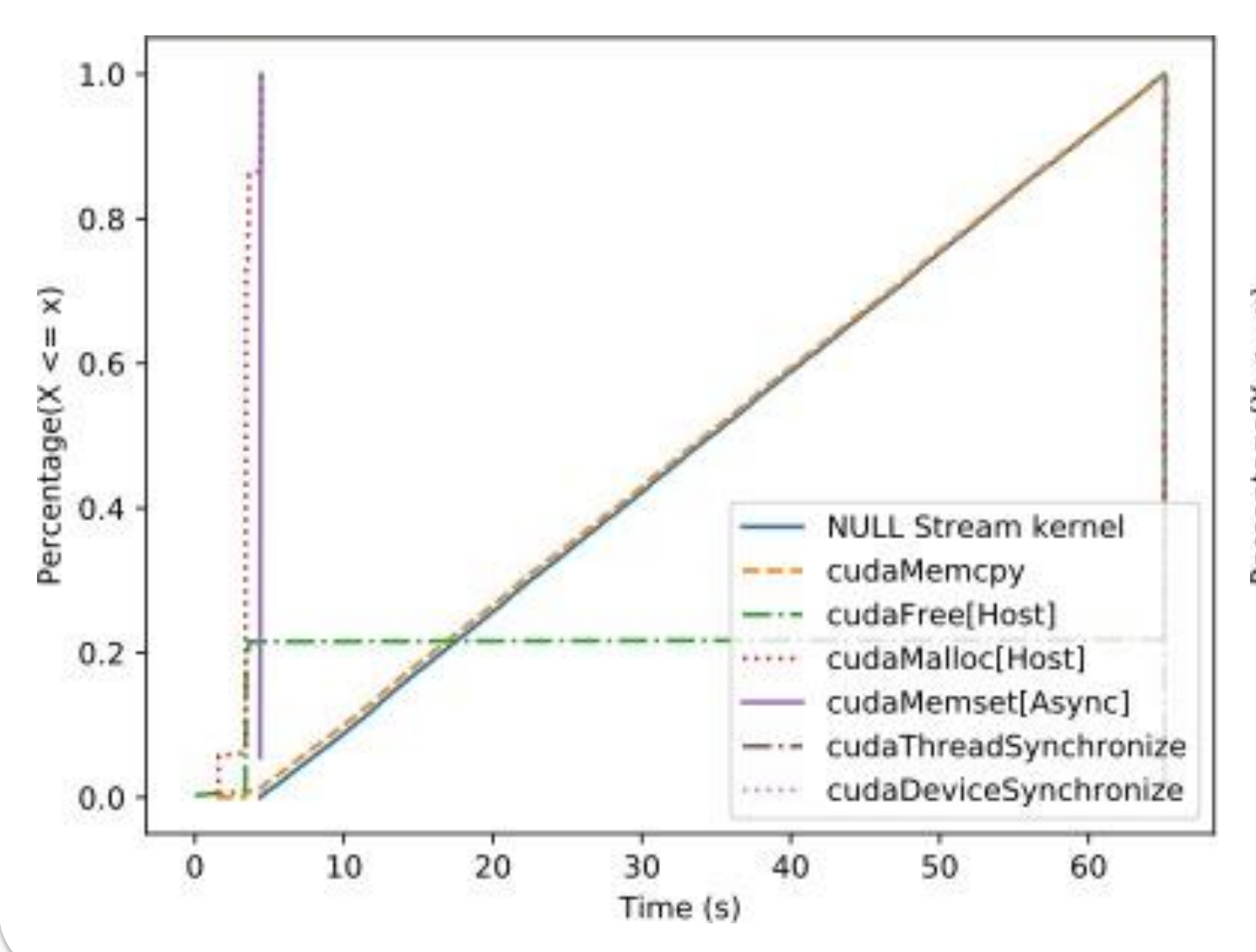
Case Study: Image-Processing Tasks

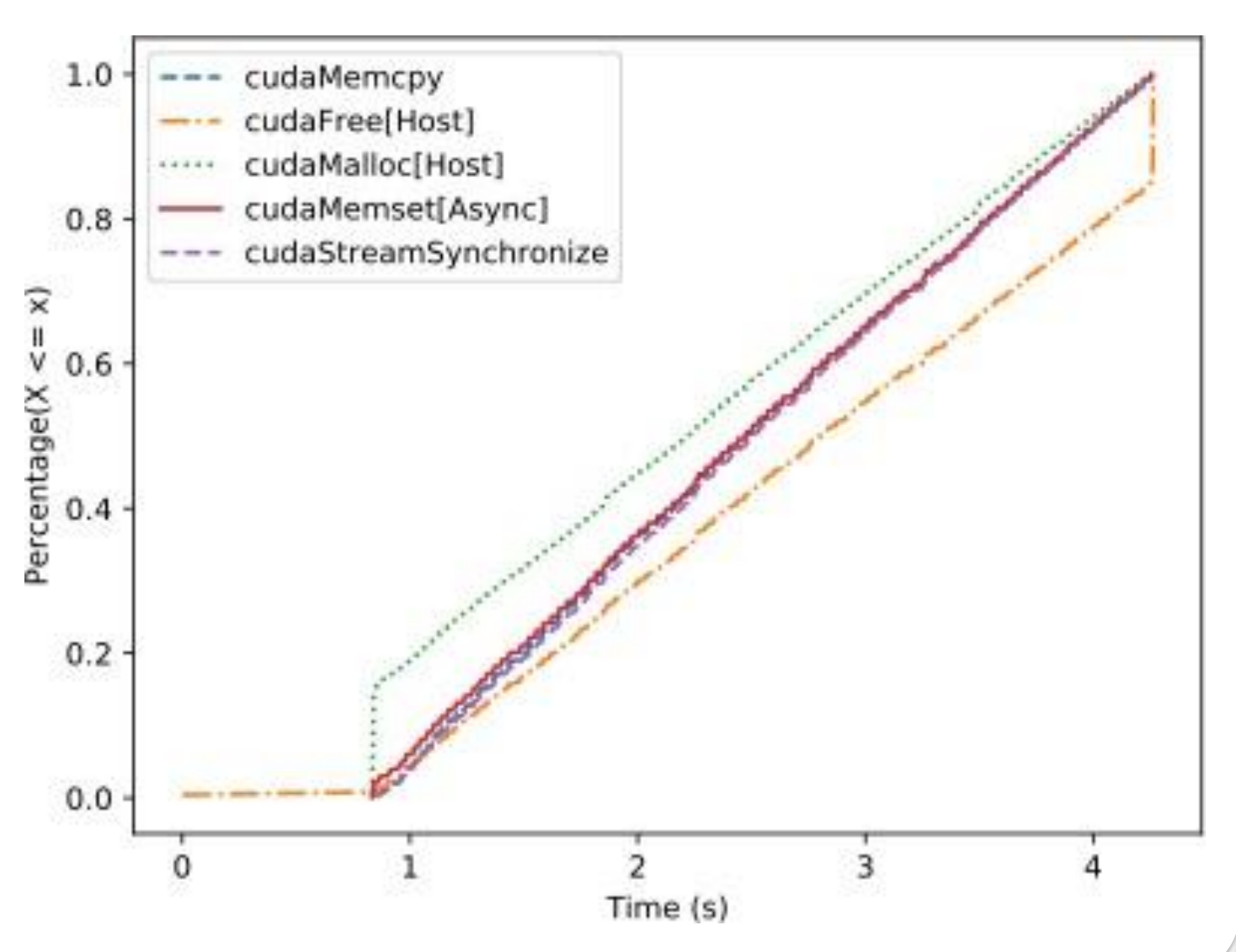
Choice of Software: CaffeNet, Hough, etc.

Methodology: We used NVIDIA's nvprof CUDA profiling tool to record known implicit-synchronization triggers of Hough and CaffeNet.

Observations:

- Hough invokes implicit-synchronization calls, including cudaFree, across a wide span within its overall runtime.
- The NULL stream is partially used in CaffeNet to intentionally trigger implicit synchronization .


Percentage(X <= x)
NULL Stream kernel
cudaMemcpy
cudaFree[Host]
cudaMalloc[Host]
cudaMemset[Async]
cudaThreadSynchronize
cudaDeviceSynchronize
Time (s)


cudaMemcpy
cudaFree[Host]
cudaMalloc[Host]
cudaMemset[Async]
cudaStreamSynchronize
Percentage(X <= x)
Time (s)