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

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

GM

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

UNC REAL-TIME SYSTEMS GROUP

UNC COMPUTER VISION GROUP

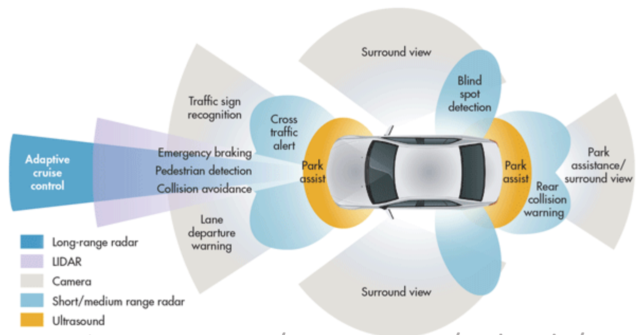
Motivation

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

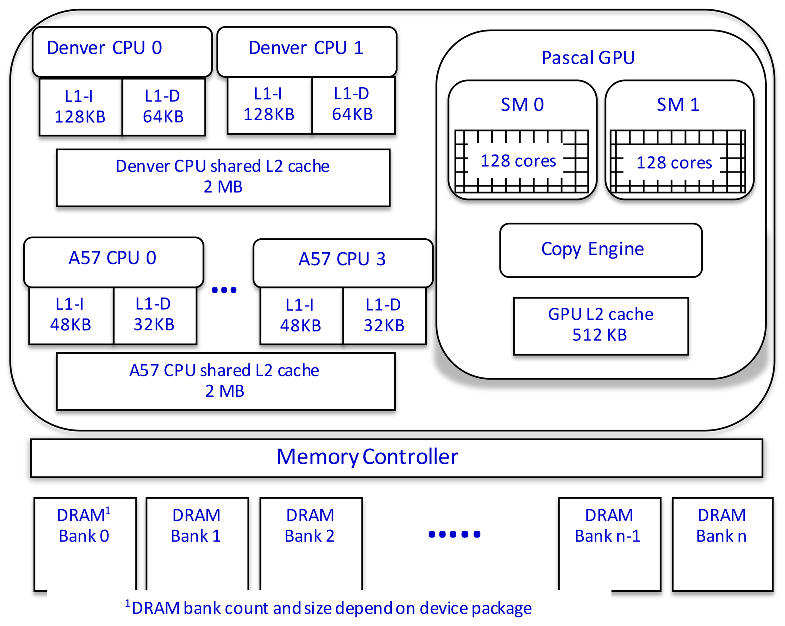
- Automotive Cyber-Physical Systems graduate-level course at UNC Chapel Hill. (http://www.cs.unc.edu/~anderson/teach/comp790a/)
- Autonomous Driving: Moving from Theory to Practice graduate-level course at UNC Chapel Hill. (http://need4speed.web.unc.edu, https://cs.unc.edu/~anderson/teach/comp790car/)

- 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, D. 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.
- M. Yang, N. Otterness, T. Amert, J. Bakita, J. Anderson, and F. D. Smith, "Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems", ECRTS 2018.
- M. Yang, T. Amert, K. Yang, N. Otterness, J. Anderson, F. D. Smith, and S. Wang, "Making OpenVX Really `Real Time'", RTSS 2018 (to appear).

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.

Inferring GPU Scheduling Behavior

Motivation:

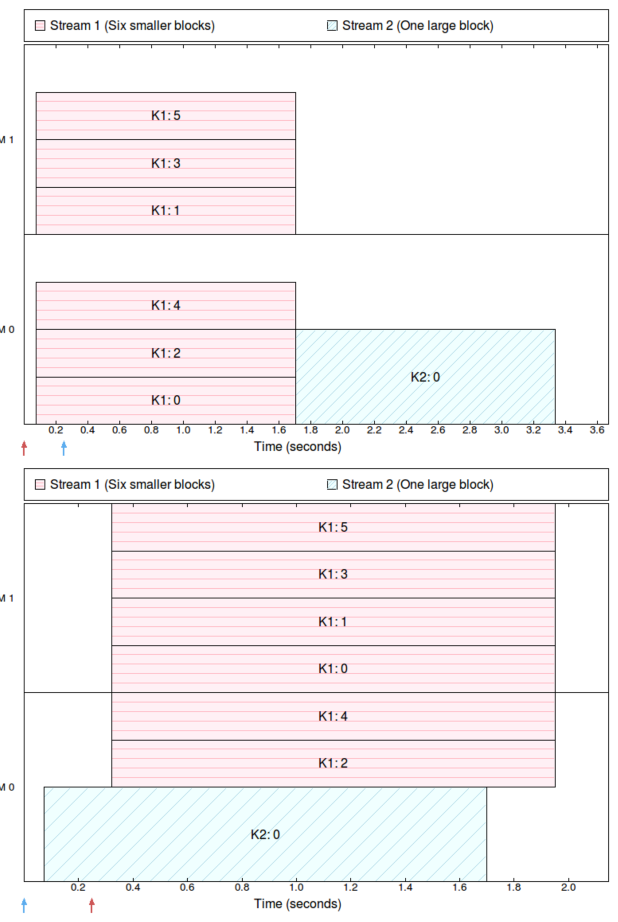
- Scheduling of GPU programs can result in wasted capacity.

Methodology:

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

Future work:

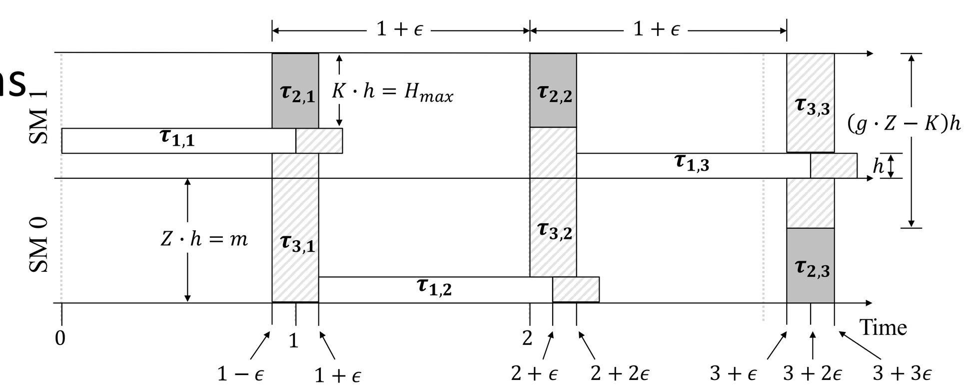
- Write middleware to reorder GPU work.



Schedulability Theory

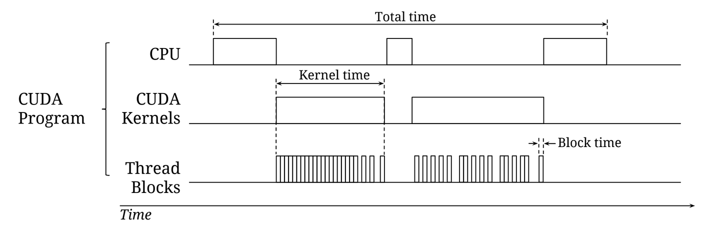
To produce response-time bound for concurrently-executed sporadic GPU-using tasks on NVIDIA GPUs, we:

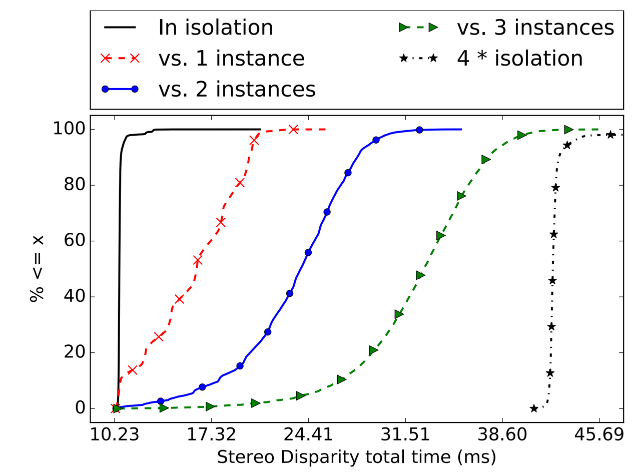
- Adapted our prior work on scheduling processing graphs and determining end-to-end graph response time bounds to apply to our fine-grained OpenVX graph, in which each node accesses either a CPU or a GPU.
- Provided new analysis for determining response-time bounds for GPU computations. We showed how to compute such bounds for recent NVIDIA GPUs by leveraging recent work by our group on the functioning of these GPUs.
- Showed that allowing invocations of the same graph node to execute in parallel is crucial in avoiding extreme capacity loss.

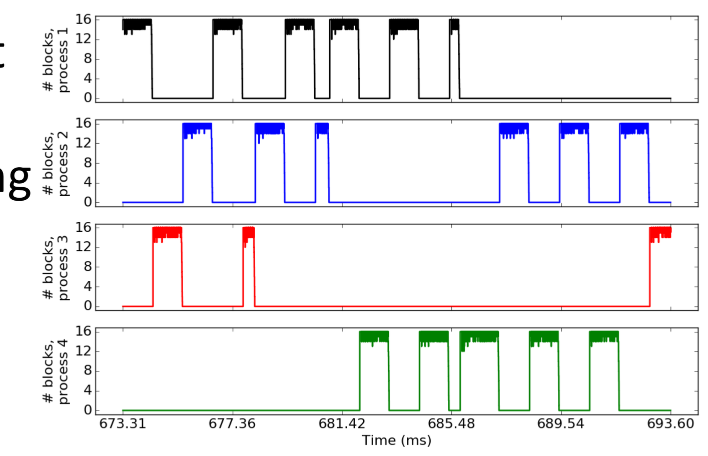


Multiprocess Co-Scheduling

- **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.

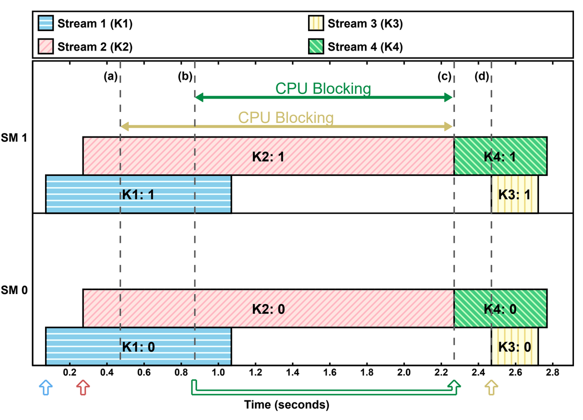


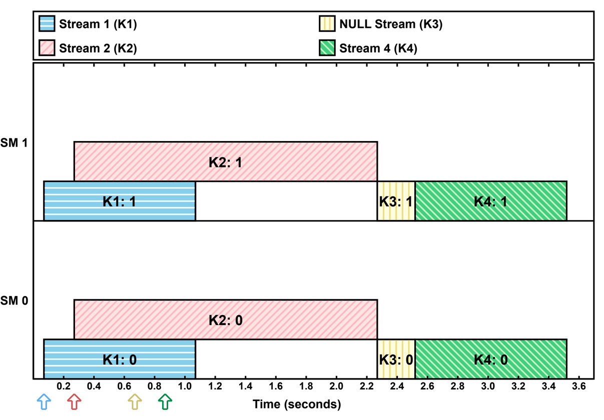




Implicit Synchronization

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





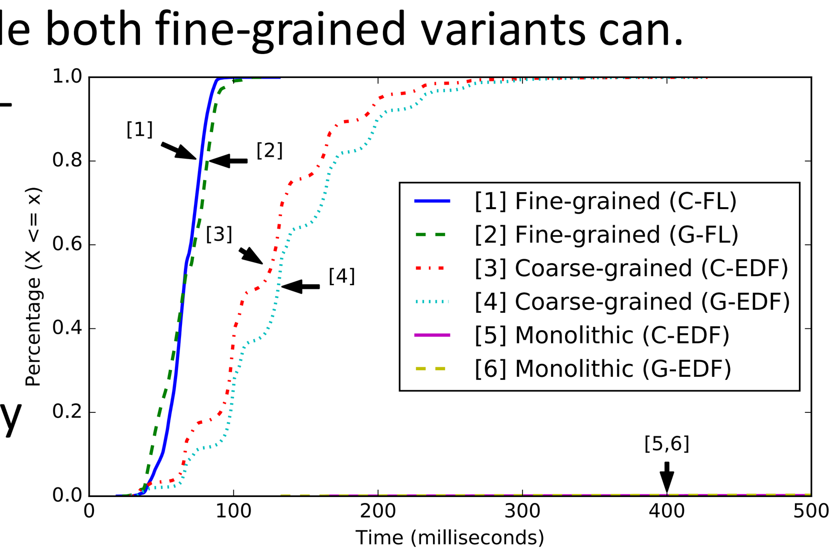
Case Study: Pedestrian-Detection Tasks

Choice of Software: Histogram of oriented gradients (HOG)

Methodology: We transformed HOG in OpenCV into a DAG. We compared the response times of successively finer-grained notions of DAG scheduling, corresponding to monolithic, coarse-grained, and fine-grained HOG DAGs, while supporting six cameras.

Observations:

- With respect to schedulablity, the monolithic and coarse-grained variants could not even come close to supporting all six cameras (i.e., DAGs), while both fine-grained variants can.
- The average (resp., worst-case) observed response time under the fine-grained variants was around 66 ms (resp., 130 ms), which is substantially lower than the non-fine grained variants.



	Monolithic G-EDF	Monolithic C-EDF	Coarse-Grained G-EDF	Coarse-Grained C-EDF	Fine-Grained G-FL	Fine-Grained C-FL
Analytical Bound (ms)	N/A	N/A	N/A	N/A	542.39	477.25
Observed Maximum Response Time (ms)	170091.06	243745.21	427.07	428.50	125.66	131.43
Observed Average Response Time (ms)	84669.47	121748.05	136.57	121.52	65.99	66.06