LET: A Way Forward for Safe GPU Co-Scheduling

Nathan Otterness, James H. Anderson
Prerequisite: Timing Models

Real-time scheduling often assumes a *Bounded Execution Time* (BET) model:

Tasks must occupy a processor for a specific time interval before their deadline.

Tasks (implicit deadline):
- $\tau_1$: $(0.4, 2.0)$
- $\tau_2$: $(0.2, 1.0)$
Prerequisite: Timing Models

Tasks (implicit deadline):
- \( \tau_1: (0.4, 2.0), \ U = 0.2 \)
- \( \tau_2: (0.2, 1.0), \ U = 0.2 \)

**Logical Execution Time (LET):**

Tasks occupy a *proportion of resources* for their *entire* period.
Prerequisite: Timing Models

Tasks (implicit deadline):
- $\tau_1$: (0.4, 2.0), $U = 0.2$
- $\tau_2$: (0.2, 1.0), $U = 0.2$

**Logical Execution Time (LET):**
Tasks occupy a *proportion of resources* for their *entire* period.

**Total utilization:**
- 40% Utilization
- 20% Utilization
- 20% Utilization
Prerequisite: Timing Models

*Logical Execution Time (LET):*

Tasks occupy a *proportion of resources* for their *entire* period.

**Tasks (implicit deadline):**
- $\tau_1$: (0.4, 2.0), $U = 0.2$
- $\tau_2$: (0.2, 1.0), $U = 0.2$
- $\tau_3$: (0.35, 0.5), $U = 0.7$

**Total utilization:**
- $\tau_1$: 20%
- $\tau_2$: 20%
- $\tau_3$: 70%
- Total: 110%
Platforms augmented with graphics processing units (GPUs), such as the NVIDIA Jetson TX1, are increasingly prevalent in embedded systems.
Despite a lack of documentation needed for modeling and certification, work is underway to incorporate GPUs into safety-critical systems.
The problem with GPU co-scheduling is that a lack of information leads to a lack of predictability.
GPU Co-Scheduling

Earlier systems work around this problem by enforcing exclusive access to GPUs.
An ideal management system will enable both predictability and concurrency.
A Simplified GPU Model

GPU capacity

100%

time
A Simplified GPU Model

![Diagram showing GPU capacity over time with K1]
A Simplified GPU Model

GPU capacity

100%

time

K1

Starts Executing
Completes
A Simplified GPU Model

Uses 50% of available capacity

CPU capacity

100%

K1
A Simplified GPU Model

GPU capacity

100%

K1
A Simplified GPU Model

GPU capacity

100%

K1
Approaches to Co-Scheduling

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, using NVIDIA's MPS middleware.
Approaches to Co-Scheduling

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, using NVIDIA's MPS middleware.
Multi-process Co-Scheduling

Starting Example: Sequential Execution

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
Multi-process Co-Scheduling

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
Multi-process Co-Scheduling

Co-scheduled kernels using multiple CPU processes

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
Multi-process Co-Scheduling

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)

Co-scheduled kernels using multiple CPU processes
Multi-process Co-Scheduling

Multiple processes share the GPU using multiprogramming.

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, using NVIDIA's MPS middleware.
Multi-thread Co-Scheduling

Reported in:
- *Inferring the Scheduling Policies of an Embedded CUDA GPU* (OSPERT '17)
- *GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed* (RTSS'17)

Starting Example: Sequential Execution
Multi-thread Co-Scheduling

Reported in:
- Inferring the Scheduling Policies of an Embedded CUDA GPU (OSPERT '17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
Multi-thread Co-Scheduling

Reported in:

- *Inferring the Scheduling Policies of an Embedded CUDA GPU* (OSPERT '17)
- *GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed* (RTSS'17)
Multi-thread Co-Scheduling

Multiple threads share the GPU through a hierarchical FIFO queue structure.

Co-scheduled kernels using multiple CPU threads

Reported in:
- *Inferring the Scheduling Policies of an Embedded CUDA GPU (OSPERT '17)*
- *GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)*
Multi-thread Co-Scheduling

Blocking can be common when multiple tasks share a CUDA context.

Reported in:
- *Inferring the Scheduling Policies of an Embedded CUDA GPU* (OSPERT '17)
- *GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed* (RTSS'17)
Approaches to Co-Scheduling

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, using NVIDIA's MPS middleware.
Multi-process Co-Scheduling with MPS

Co-scheduled kernels using MPS

GPU capacity

100%

time
Multi-process Co-Scheduling with MPS

Co-scheduled kernels using MPS
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.

The best system:
- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.

The best system:
- Is easy to schedule (e.g. supports preemption). 
- Does not require modifying existing CUDA programs. 
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.

The best system:

- Is easy to schedule (e.g. supports preemption). ✗
- Does not require modifying existing CUDA programs. ✔
- Fully utilizes the GPU when possible. ✔
Sporadic kernels under MPS (or threads)

Kernel K3, with the closest deadline, is blocked by two kernels with less-urgent deadlines.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.

The best system:
- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Sporadic kernels using Multiprogramming

100% GPU capacity

K1 K2 K3

(Starting example)
Multiprogrammed kernels

Adjusted "capacity": 300%

Response time may be better, but still not good enough to meet deadline (runs at ~1/3rd speed + overhead)
Multiprogrammed kernels

Adjusted "capacity": 225%

Reducing resource requirements from competitors won't help with multiprogramming overhead.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware.

The best system:

- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

● Every task is a separate CPU process (no GPU middleware).
● Every task is a separate CPU thread in a single process.
● Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

● Is easy to schedule (e.g. supports preemption).
● Does not require modifying existing CUDA programs.
● Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

- Is easy to schedule (e.g. supports preemption).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

- Is easy to schedule (*approximates fluid scheduling*). ✔
- Does not require modifying existing CUDA programs. ✔
- Fully utilizes the GPU when possible.
Which one is best for safety-critical systems?

Different ways to Co-Schedule GPU Tasks:

- Every task is a separate CPU process (no GPU middleware).
- Every task is a separate CPU thread in a single process.
- Every task is a separate CPU process, but using NVIDIA's MPS middleware on a Volta-architecture GPU to limit per-task computing resources.

The best system:

- Is easy to schedule (approximates fluid scheduling).
- Does not require modifying existing CUDA programs.
- Fully utilizes the GPU when necessary.
Applying LET to Volta MPS
Applying LET to Volta MPS

Make K1 and K2 use the minimum capacity necessary-- All kernels meet their deadlines.
What needs to be done?

This is still ongoing work. The next steps include:

1. Determining formulas relating GPU utilization to execution time.  
   (This can actually be measured per-task rather than per-kernel.)
2. Write a management system that dynamically sets utilization limits based on the formulas and tasks' deadlines.
Potential Problems

- Volta GPUs are currently expensive and in short supply.
- Embedded GPUs (so far) do not support MPS, regardless of GPU architecture.
- There's no guarantee that future GPU architectures will support setting resource limits.
Conclusion

Safe, predictable real-time scheduling seems possible, when applying the principles of LET to GPU resource partitioning on Volta-architecture GPUs.
GPU Co-Scheduling with Processes

When GPU tasks are launched from separate CPU processes (CUDA contexts), co-scheduling is achieved via multiprogramming.

Reported in:
- An Evaluation of the NVIDIA TX1 for Supporting Real-time Computer-Vision Workloads (RTAS'17)
- GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed (RTSS'17)
GPU Co-Scheduling with Processes

Small, short-lived competing workload

Small, long-lived competing workload

Multiprogramming on Maxwell GPUs leads to blocking.
GPU Co-Scheduling with Processes

Multiprogramming on Pascal GPUs leads to disproportionate performance loss.