Work in Progress: Increasing Schedulability via on-GPU Scheduling

Joshua Bakita (jbakita@cs.unc.edu) and James H. Anderson (anderson@cs.unc.edu)
University of North Carolina at Chapel Hill

The Pitch

When scheduling multiple tasks on one GPU, by executing the GPU scheduler on the GPU rather than on the CPU, we can reduce absolute overhead, and eliminate capacity loss that occurs on the CPU.

We do this by developing a new scheduling framework for discrete NVIDIA GPUs that runs entirely on the GPU itself.

Problem

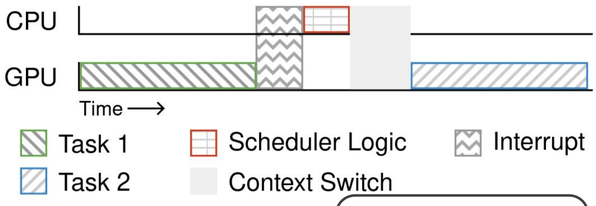

Fig. 1. Illustration of execution steps for prior work on GPU scheduling; *e.g.*, GCAPS or TimeGraph.

GPU interrupts to the CPU to run scheduler ①

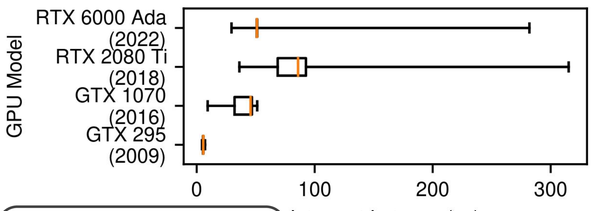

Fig. 2. 0, 25, 50, 75, and 100th percentile time between triggering an interrupt on the GPU and receiving it on the CPU (100 samples).

But interrupt latencies can be in the hundreds of microseconds ②

And dedicating a CPU core for spin-based synchronization is expensive ③

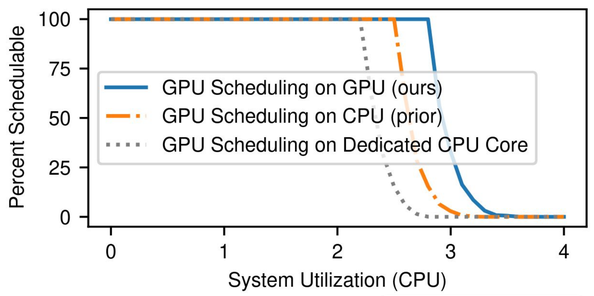

Fig. 3. Schedulability under G-EDF on the CPU for tasks of medium periods and utilizations after the overhead of interrupt-driven GPU scheduling (165 µs every 2 ms in this experiment), or of dedicating a CPU core to GPU scheduling.

Leading to reduced schedulability, esp. when the scheduler must be run frequently ④

Our Solution

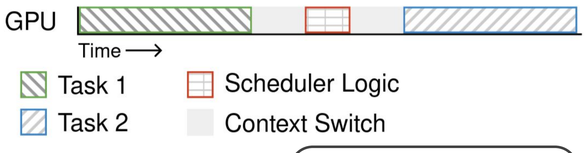

Fig. 4. Illustration of execution steps with our approach to GPU scheduling.

We run GPU scheduling synchronously on the GPU Ⓐ

OVERHEAD OF ON-GPU SCHEDULING ON RTX 6000 ADA

# Tasks	Time w/ Built-in (baseline) (ms)	Time w/ on-GPU (ours) (ms)	Overhead (per invocation)
2	94,218	98,578	95 μs
3	141,334	147,922	95 μs
4	188,450	198,021	104 μs
5	235,540	248,463	112 μs

Tbl. 1. Absolute overhead of our approach (vs. 421–783 µs typical with GCAPS)

Yielding lower absolute overheads Ⓑ

And eliminating any CPU time cost Ⓒ

Framework Features

Our GPU scheduling framework:

- Works **on many** NVIDIA **GPUs** (2018+)
- Supports **arbitrary schedulers** via a plugin-based interface
- **No task modifications** required
- **No hardware modifications** required
- Works for **any task type** (CUDA, OpenGL, Vulkan, etc.)
- **Preserves logical isolation** between tasks
- Uses **hardware-enforced** time budgeting

Limiting Preemption Overhead

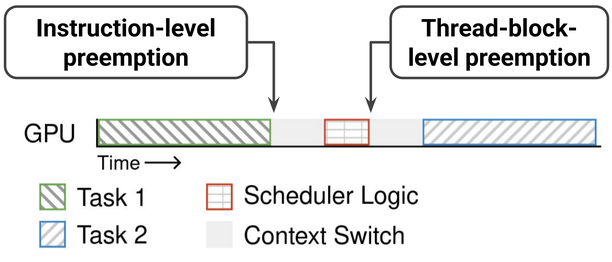

Fig 5. We use low-overhead predefined preemption points (thread-block boundaries) for our scheduler, and arbitrary ("instruction-level") preemption for tasks. This reduces the amount of state that must be saved or restored for our scheduler, while still ensuring that budgets are accurately enforced for tasks.

RTAS 2025
UNC REAL-TIME SYSTEMS GROUP
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Full implementation of our work is under submission to appear at an operating systems conference later in 2025.