Scaling Up: The Validation of Empirically Derived Scheduling Rules on NVIDIA GPUs

Joshua Bakita
Department of Computer Science, University of North Carolina at Chapel Hill
14th Annual Workshop on Operating Systems Platforms for Embedded Real-Time (OSPERT), Session III
July 3rd 2018
Certifying Autonomy

- GPUs best fit size, weight, and power requirements
- Users want safety guarantees, but millions, or even billions of hours of road testing would be needed to achieve statistical meaning
- Formal (mathematical) guarantees cannot be made without understanding the hardware
- Central role of GPUs demands a solid understanding of them
Anatomy of Autonomy

- Computer Vision
- Sensor Fusion
- Vehicle Routing
- Actuation and Control
- Scheduler
- Hardware GPU and CPU
- Traditional Automotive Real-time Platform
Anatomy of Autonomy

- Computer Vision
- Sensor Fusion
- Vehicle Routing
- Scheduler
- Hardware GPU and CPU
- Actuation and Control
- Traditional Automotive Real-time Platform
How do we enable GPU certification?

- Determine rules of behavior
- Rigorously validate rules
How do we enable GPU certification?

- Determine rules of behavior ✔ - postulated in past research at UNC
- Rigorously validate rules
How do we enable GPU certification?

- Determine rules of behavior ✔ - postulated in past research at UNC
- Rigorously validate rules ✗ - focus of my paper
Definitions

CUDA Thread Block: A group of GPU threads executing the same set of user-defined instructions in lockstep. This is the lowest-level GPU scheduling unit considered in the paper.

CUDA Kernel: A combination of instruction code and CUDA thread block specifications. Dispatched asynchronously by a user-space process.

CUDA Stream: A first-in-first-out (FIFO) work queue into which processes on the CPU can dispatch kernels.
Definitions

**SM:** A subdivision of an NVIDIA GPU. Single thread blocks cannot be split across multiple SMs.

**EE (Execution Engine) Queue:** A special internal queue of kernels that our past work has defined to exist between CUDA stream queues and the actual GPU (explained in later figure).
Limits of Empirical Observation

Previously Published Test [1]

Limits of Empirical Observation
Superhuman Scale

Autonomous validation of scheduling rules via state machine

FLOWCHART

KERNEL_LAUNCH_START

GPU Idles

KERNEL_LAUNCH_END

KERNEL_END

BLOCK_START

BLOCK_END

GPU Executes

ee_queue.enqueue(kernel)

stream_queues[s].enqueue(kernel)

Is kernel at stream queue head?

Yes

No

Is parent kernel still executing?

Yes

No

Does kernel have more blocks to dispatch?

Yes

No

Does SM have more threads available than possible?

Yes

No

Are any other kernels queued?

Yes

No

threads_avail[SM] >= done_threads

ee_queue.dequeue()

threads_avail[SM] >= start_threads

Validate Fail

Kernel at EE queue head, SM exist, and are threads available?

Yes

No

Is kernel fully dispatched and at stream queue head?

Yes

No

stream_queues[s].dequeue()

ee_queue.enqueue(stream_queues[s].head())
Considered Events

Timestamps included in traces from GPU tests:

- Kernel launch start
- Kernel launch end
- Kernel end¹
- Thread block start
- Thread block end

¹ Pseudo-event; sometimes it is undesirable for a benchmark to perform a `cudaStreamSynchronize` to retrieve the actual end time. In those cases the tokenizer uses the end time of the last thread block of the kernel as a substitute.
Results

- Postulated rules apply in simple tests on recent GPUs
- Older GPUs follow different rules
- Rules do not strictly apply in complex tests on recent GPUs
  - Clock jitter?

Validating: BLOCK_END (SM8/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM3/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM4/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM0/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM11/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM6/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_END (SM3/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM7/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_END (SM4/770) (K1/pri-0/stream-25524) (Multi-kernel)
Validating: BLOCK_START (SM8/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM8/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM11/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM11/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM0/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM11/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel)
Validating: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel)
Validation failed at timestamp 2.784003778: Block starting for kernel
Different Rules in Effect

Previously Published Test [1]

Same Test, Different GPU Generation

Improper Ordering (Kepler)

Relevant Rules:

**G2:** “A kernel is enqueued on the EE queue when it reaches the head of its [CUDA] stream queue.” [2]

**G4:** “A kernel is dequeued from its [CUDA] stream queue once all of its blocks complete execution.” [2]

Improper Ordering (Kepler)

New Rule:

G2 (Kepler): “A kernel is dequeued from its stream queue and enqueued on the EE queue when it reaches the head of its stream queue.”

Kepler dates from 2012
Ordering Jitter (Newer GPUs)

Relevant Rules:

**G3:** “A kernel at the head of the EE queue is dequeued from that queue once it becomes fully dispatched.” [2, p. 5]

**X1:** “Only blocks of the kernel at the head of the EE queue are eligible to be assigned.” [2, p. 6]

Ordering Jitter (Newer GPUs)

2.784003586: BLOCK_START (SM11/376) (K2/pri-0/stream-25523) (Multi-kernel submission: Stream 2)
2.784003618: BLOCK_START (SM5/376) (K2/pri-0/stream-25523) (Multi-kernel submission: Stream 2) <- Blocks of S2/K2 stop dispatch
2.784003778: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4) <- Blocks of S4/K3 begin dispatch
2.784003778: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003778: BLOCK_START (SM4/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003778: BLOCK_START (SM0/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003778: BLOCK_START (SM11/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM0/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM8/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM8/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM11/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003810: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM0/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM8/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM8/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM11/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003842: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM0/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM4/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM4/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4)
2.784003874: BLOCK_START (SM3/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4) <- Blocks of S4/K3 finish dispatch
2.784003874: BLOCK_START (SM7/323) (K3/pri-0/stream-25525) (Multi-kernel submission: Stream 4) <- Blocks of S4/K3 finish dispatch
2.784004962: BLOCK_START (SM6/376) (K2/pri-0/stream-25523) (Multi-kernel submission: Stream 2) <- Block of S2/K2 dispatched
Future Work

- Investigate specific source of EE queue ordering jitter
  - Wall-clock distribution latency? (%globaltimer)
  - Propagation latency?
  - Resource blocking?
  - Multiple EE queues?
- Expand framework to validate more rules
  - Only validates six of the sixteen rules [2] at present
- Automate random workload execution and validation cycles

Impacts

- Will eventually allow GM Research and other autonomous vehicle developers to more confidently build on our theoretical rules
- Allows quick validation of different NVIDIA GPUs, yielding more flexibility to developers and creating the ability to take real-time learnings from one generation to the next
Questions?

Works cited and thanks to:

1. N. Otterness, M. Yang, T. Amert, J. Anderson, and F.D. Smith. Inferring the scheduling policies of an embedded CUDA GPU. In OSPERT ’17.