A GPU Kernel Transactionization Scheme for Preemptive Priority Scheduling

Hyeonsu Lee, Jaehun Roh, Euiseong Seo

School of Software

Sungkyunkwan University

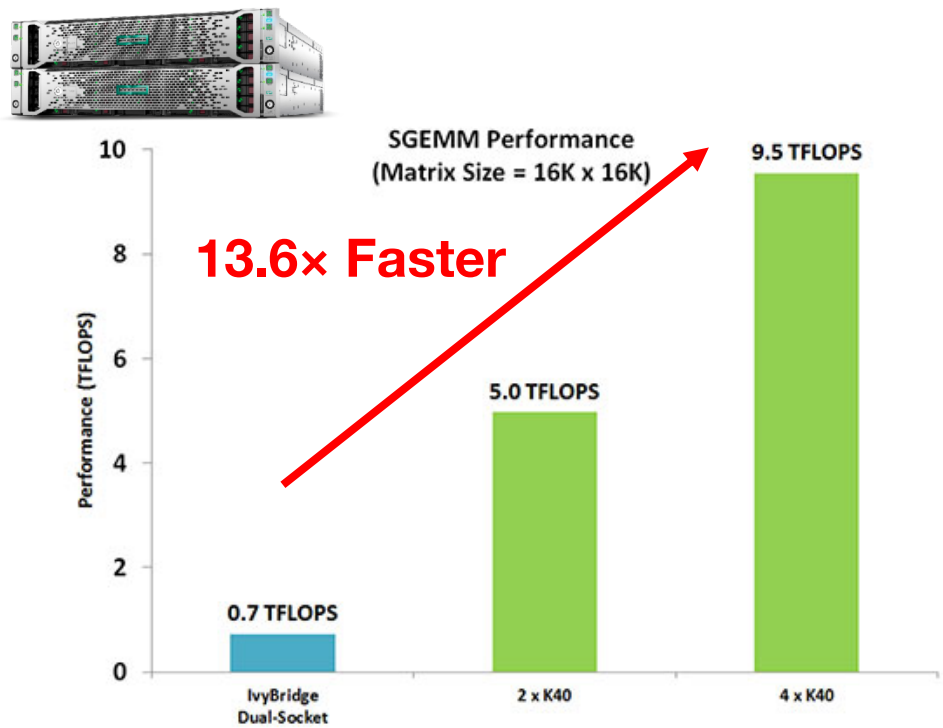
SUNG KYUN KWAN UNIVERSITY

Index

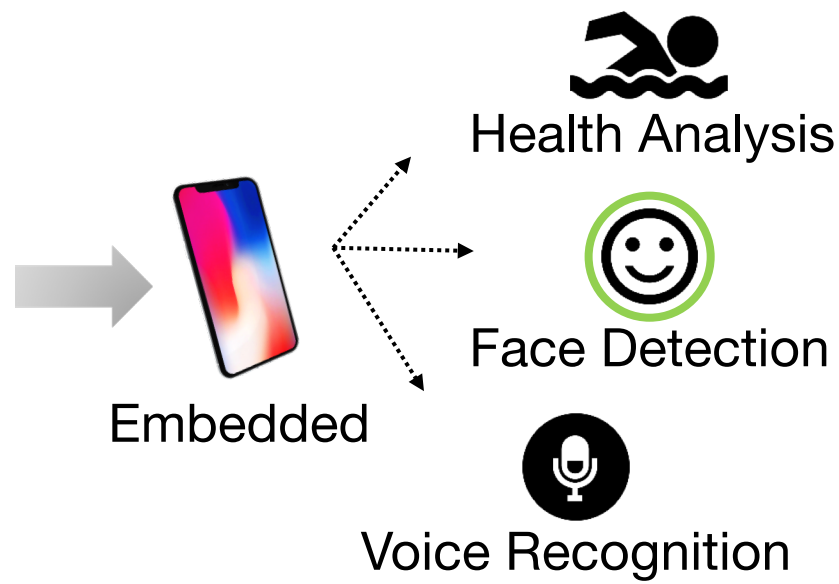
Contents

Introduction

- GPU is specialized in parallel computing workload
  - Embedded systems can earn substantial benefit from GPU computing



NVIDIA's Machine Learning Benchmark



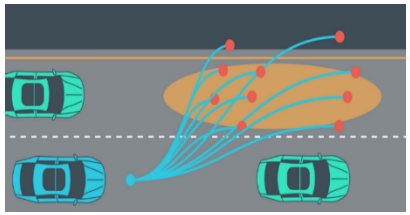
- Introduction
- **Motivation**
- Related Work
- Background
- Our Approach
  - Design
  - Implementation
- Evaluation
- Conclusion

Motivation

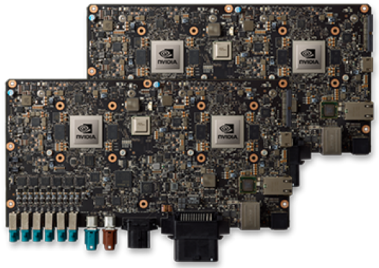
- Example: Autonomous vehicle control systems using GPU

High Priority

Evasion Path Finder



Delayed



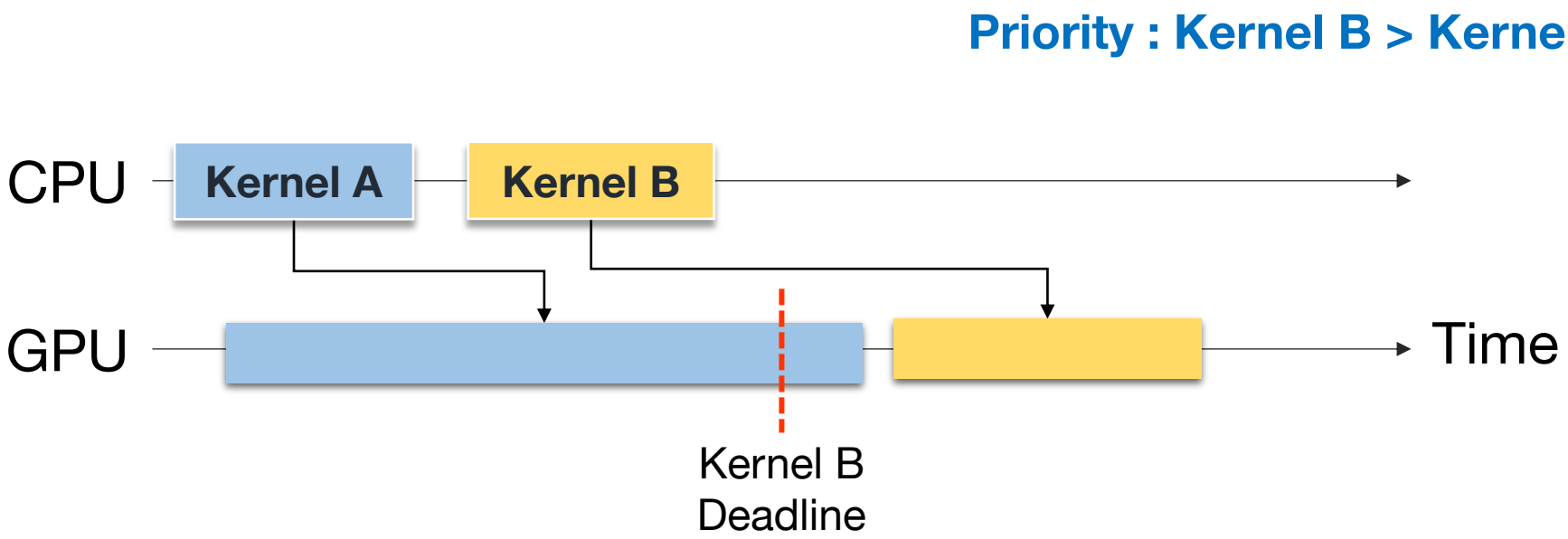
GPU Embedded System

Run

Object Detection

Motivation

- Priority inversion problem on GPU
  - No GPU Preemption HW and Interfaces

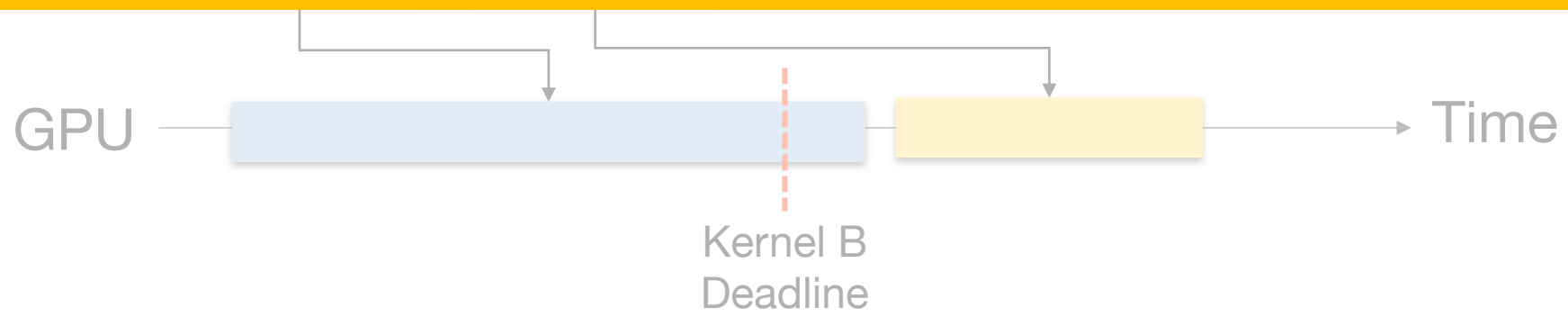


Motivation

- Priority inversion problem on GPU
  - No GPU Preemption HW and Interfaces

Priority : Kernel B > Kernel A

Embedded Systems Need
Preemptive Priority Scheduling of GPU Kernels



- Introduction
- Motivation
- **Related Work**
- Background
- Our Approach
  - Design
  - Implementation
- Evaluation
- Conclusion

Related Work - Hardware

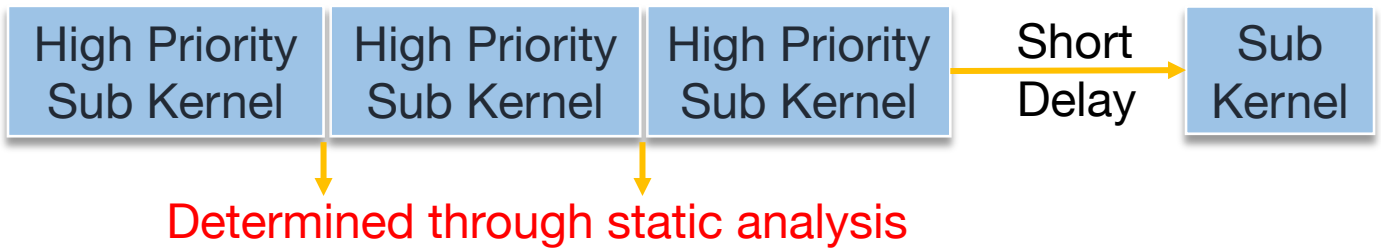
- HW extension for preemptive GPU scheduling
  - Experimental architecture was proposed and evaluated with simulation
  - Nvidia commercialized GPUs with HW preemption

    (But, they have not disclosed API for its, yet)

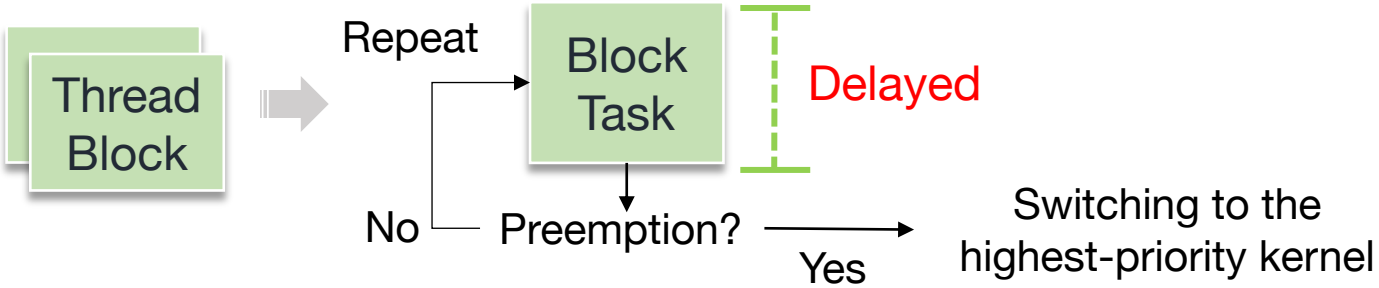
- Limitations of HW GPU preemption
  - Up to 100μs context switching time from massive amount of state saving/restoring
  - Increased complexity in HW design and thus cost to manufacture

Related Work - Software

- Kernel slicing approach



- Thread-block-level scheduling approach



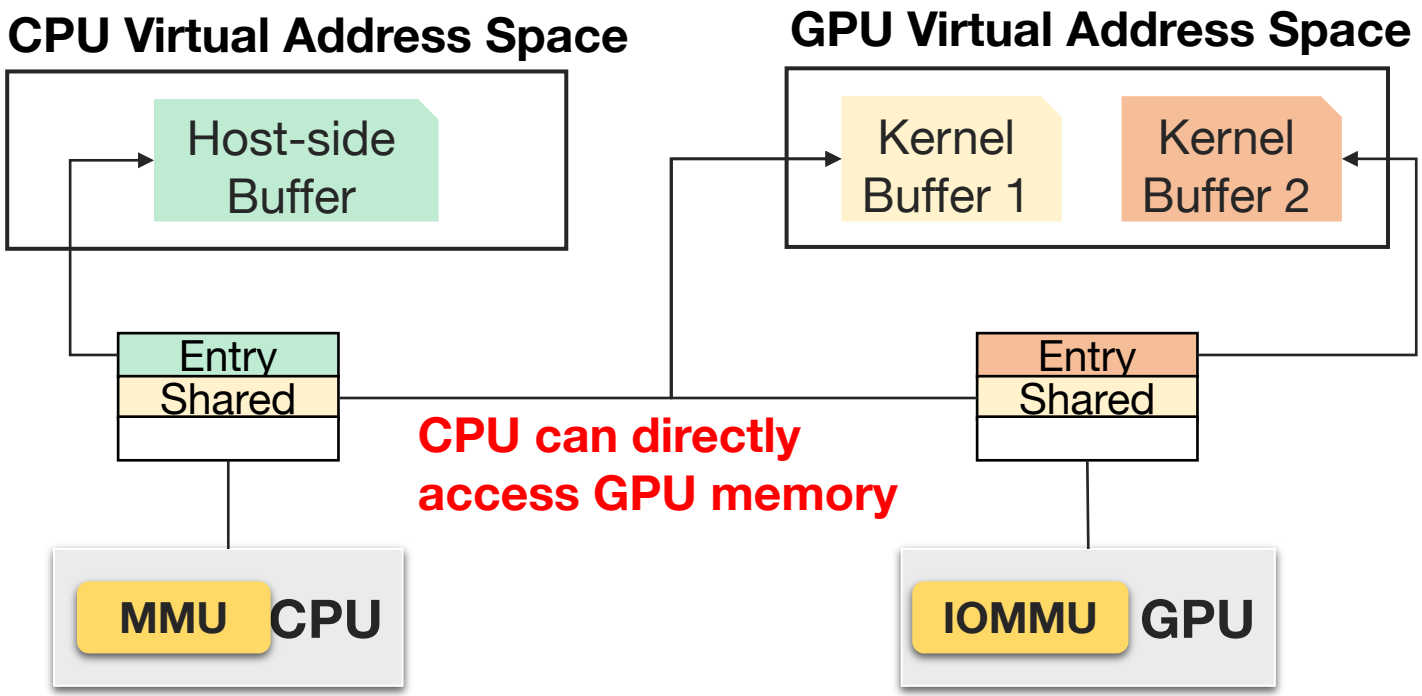
We propose a software preemption solution that

- **Immediately aborts currently running kernel**
- **Re-executes aborted kernels later**

- Introduction
- Motivation
- Related Work
- **Background**
- Our Approach
  - Design
  - Implementation
- Evaluation
- Conclusion

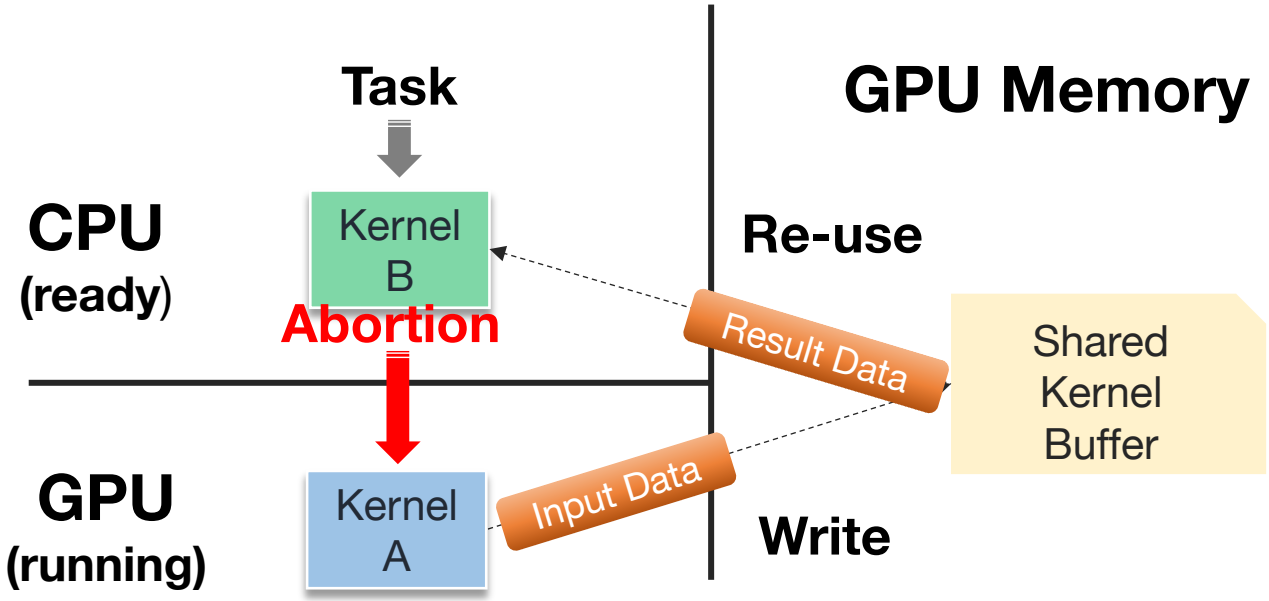
Background – Shared System Memory

- GPU memory management in embedded systems



Background – Shared Kernel Buffers

- Producer-consumer relationships through sharing kernel buffers
- Blind abortion and re-execution may corrupt data delivered from its producer



Producer Consumer Relationship

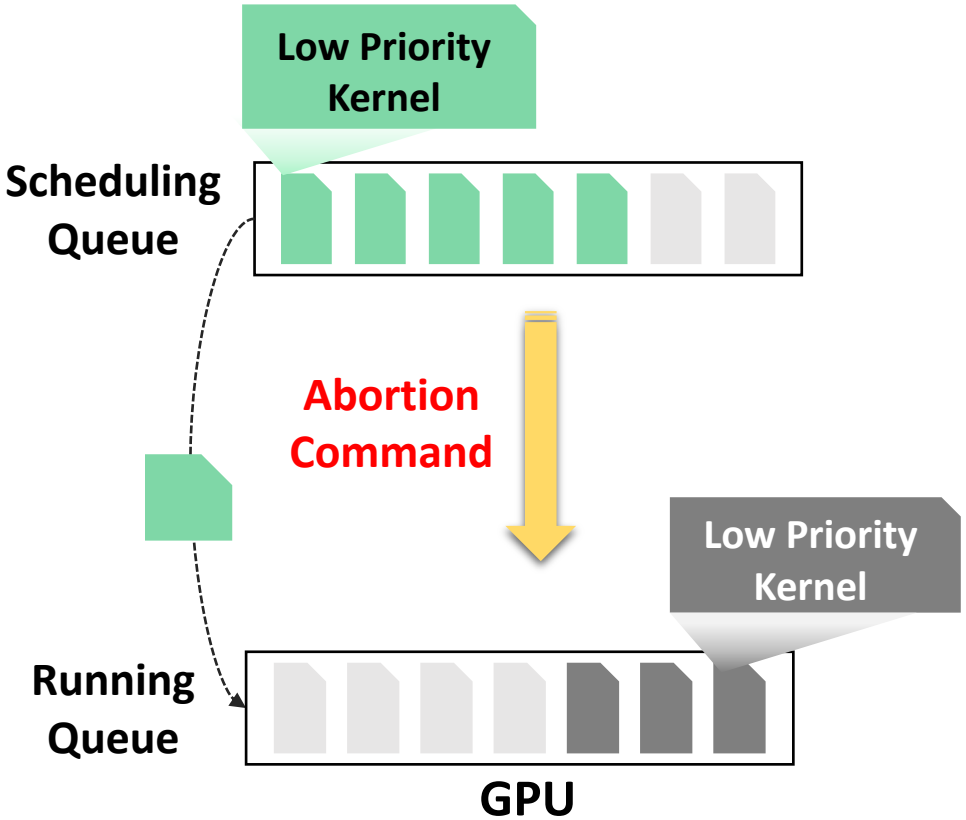
- Introduction
- Motivation
- Related Work
- Background
- **Our Approach**
  - **Design**
  - Implementation
- Evaluation
- Conclusion

Design - Preemption with Transactionized Kernels

- Our design is structured in

  GPU kernel abortion :

  To immediately abort the low priority kernel

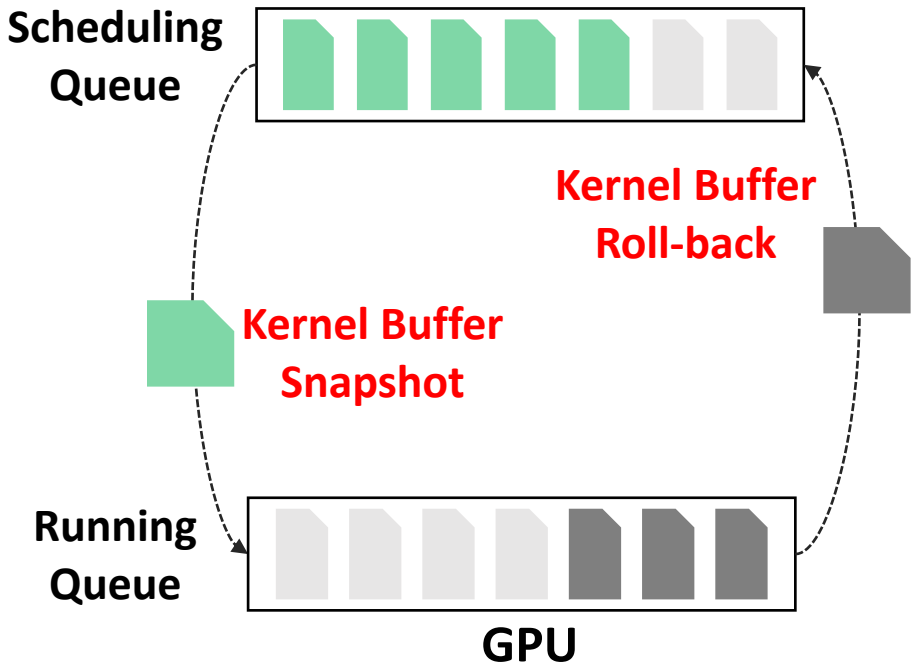


Design - Preemption with Transactionized Kernels

- Our design is structured in

  GPU memory context saving/restoring :

  For re-execution of a preempted kernel

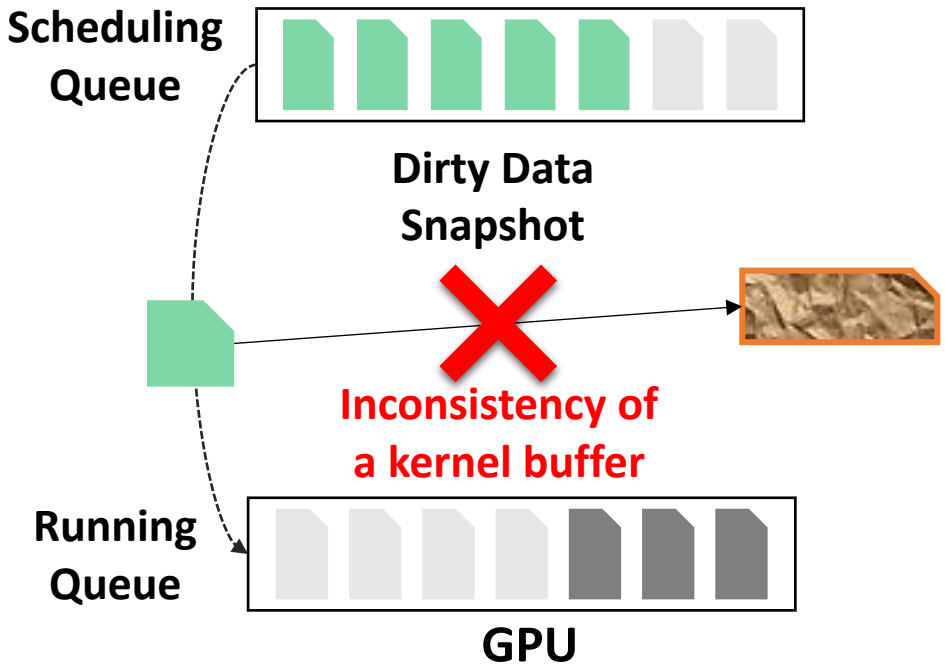


Design - Preemption with Transactionized Kernels

- Our design is structured in

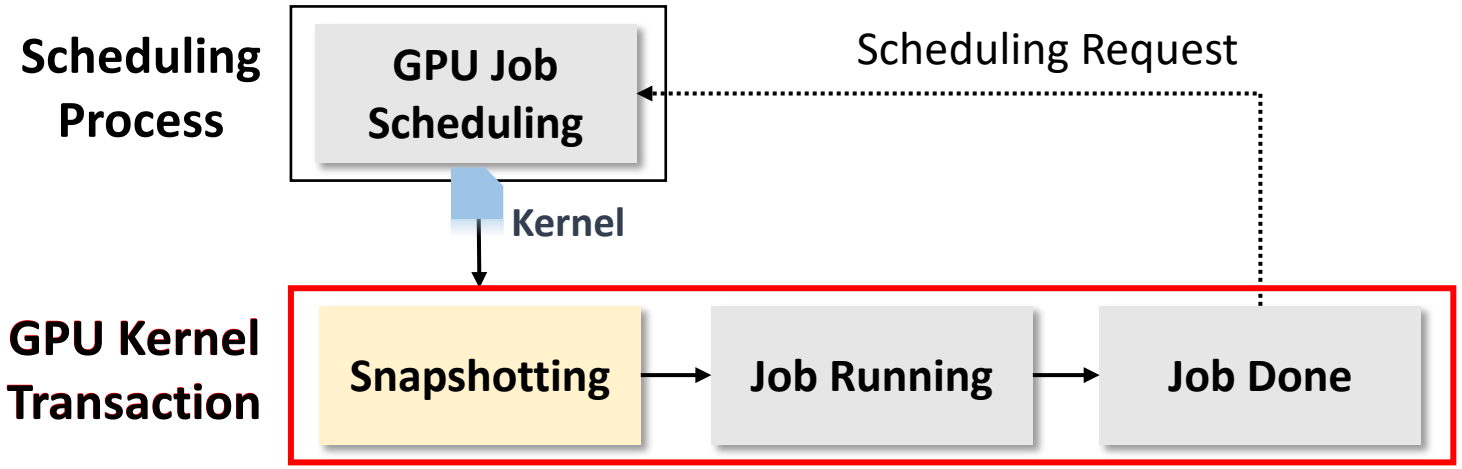
  Transaction of GPU kernel execution :

  To prevent simultaneously access shared kernel buffer



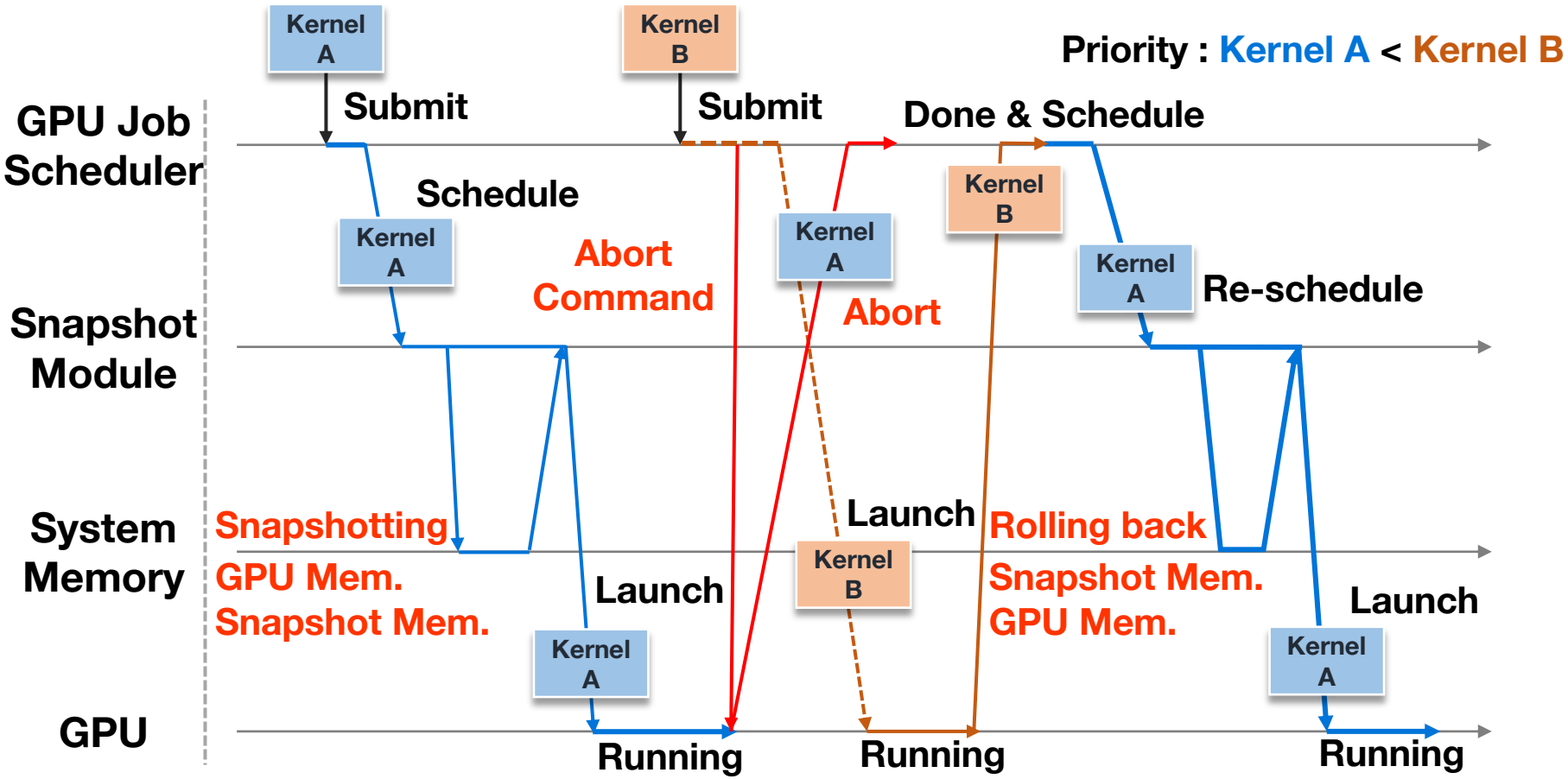
Design - Transactionization Scheme

- GPU kernel transaction structure for re-execution of kernel
  - Procedures from snapshotting to kernel finish



Design – Example of Transactionization Scheme

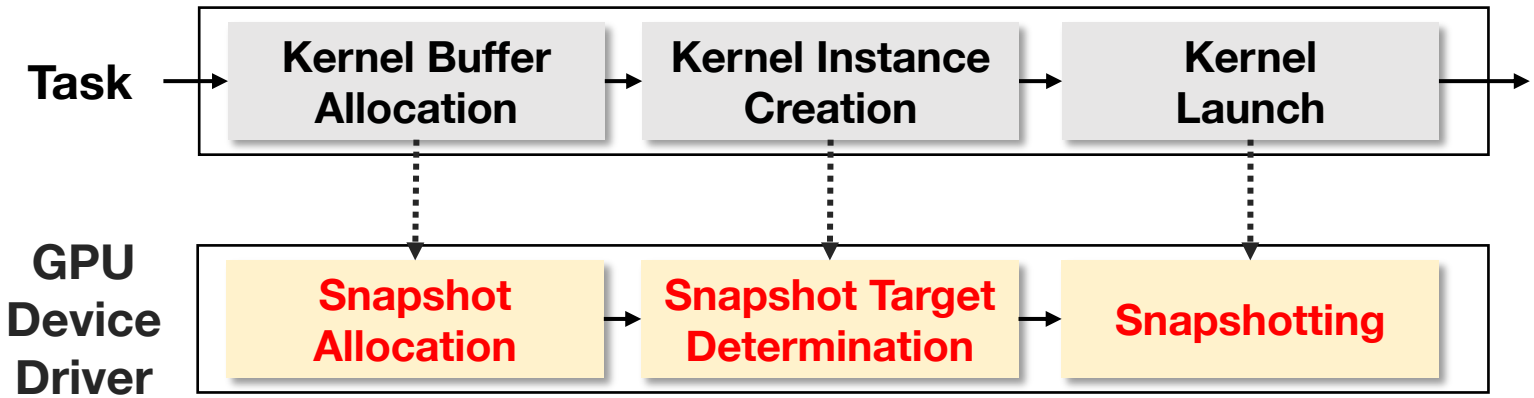
- Pseudo-preemptive priority scheduling flow of Transactionized GPU Kernels



- Introduction
- Motivation
- Related Work
- Background
- **Our Approach**
  - Design
  - **Implementation**
- Evaluation
- Conclusion

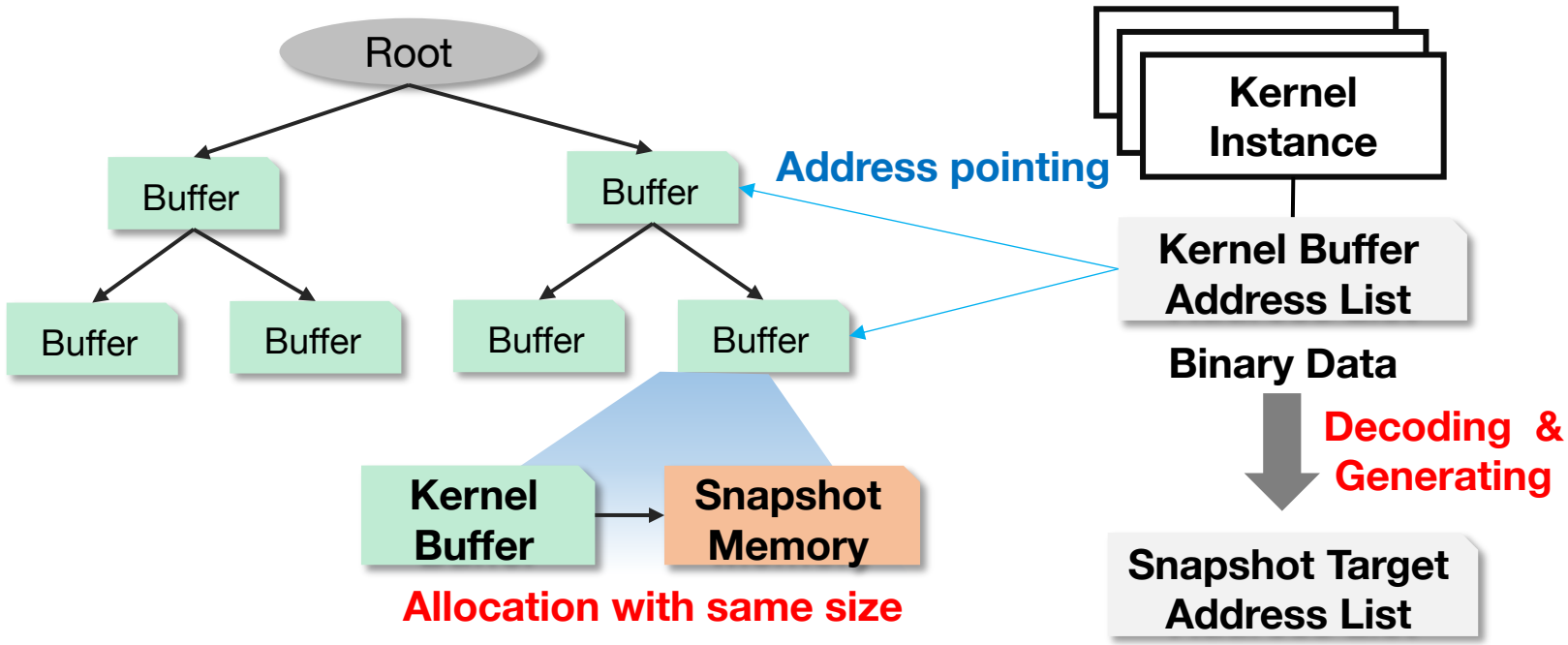
Implementation – Snapshot Mechanism

- Implementing snapshot mechanism through GPGPU programming model



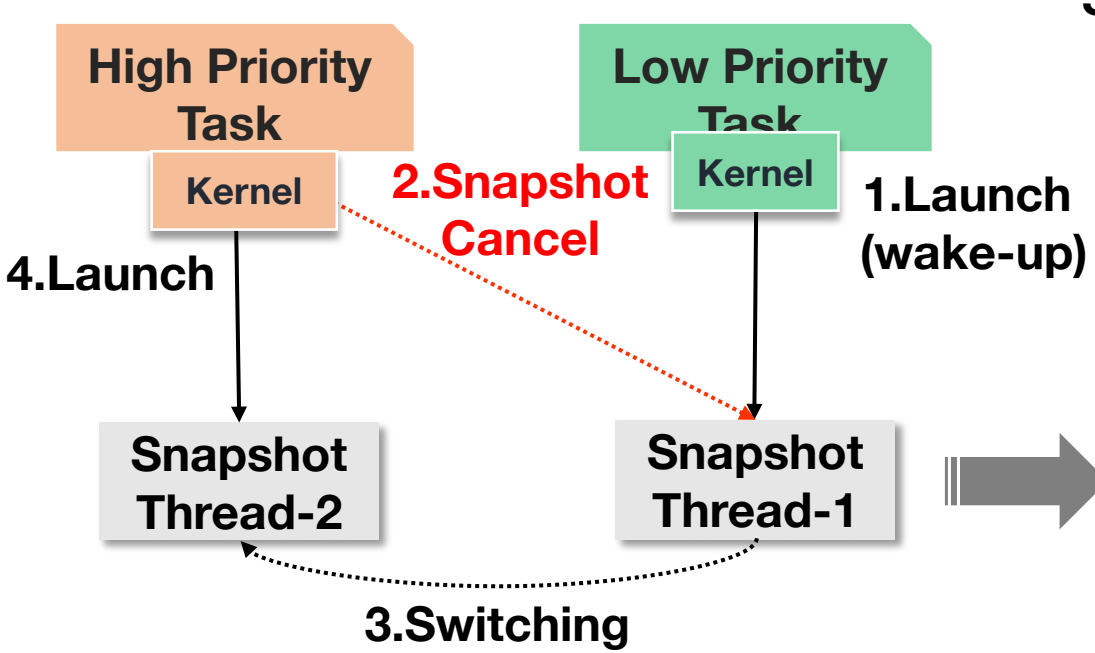
Implementation – Snapshot Memory Management

- Allocating snapshot memory when allocating a kernel buffer
- Tracking kernel buffer addresses thorough metadata of kernel when launching kernel



Implementation – Snapshot Process

- Snapshot process is implemented by kernel threads



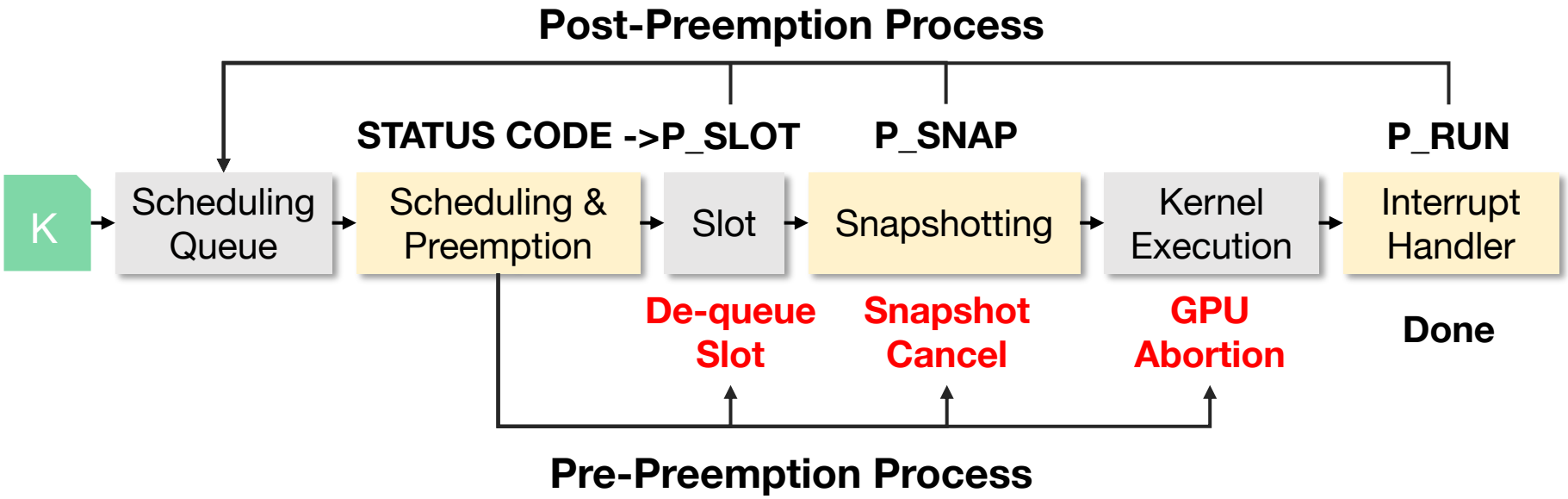
Snapshot Thread Pseudo Code

```
while(1){
    if( ! Is_next_kernel )
        interruptible_sleep()

    if( re-execution )
        roll-back();
    else
        snapshot();

    submit_to_GPU();
}
```

Snapshot Cancel Range

Implementation – Preemption Mechanism

- Pre-preemption process
  - Minimizing process of aborting scheduled kernels
- Post-preemption process
  - Recovering aborted kernels to the scheduling queue
  - Running a kernel launch and a post-preemption process



- Introduction
- Motivation
- Related Work
- Background
- Our Approach
  - Design
  - Implementation
- **Evaluation**
- Conclusion

Evaluation Setup (1)

- Target device environment

Board	Odroid-XU3 Rev. 2	SoC	Exynos 5422
CPU	4×Cortex A15@2.0Ghz 4×Cortex A7@1.4Ghz	GPU	Mali-T628 MP6 @600Mhz
	2MB Shared L2 Cache		256KB L2 Cache
Memory	2GB LPDDR3 RAM @ 933MHz		
Storage	32GB eMMC 5.0 HS400 Flash Storage		
OS	Linux 3.10.72 with Mali r5p0-06rel0 driver		

ARMMALI
Visual Technology

- Rodinia benchmark suite 3.1

Program	Abbr.	Description	Num. of Kernel Inst.	Snapshot Size per Kernel (Avg.)	Snapshot Memory Size (at Peak)	Avg. Kernel Exec. Time	Turnaround Time
k-Nearest Neighbors	NN	Dense Linear Algebra	1	60 MB	60.35 MB	10.38 ms	0.68 s
Back Propeagation	BP	Unstructured Grid Algebra	2	6.79 MB	9.42 MB	15.74 ms	0.39 s
PathFinder	PF	Dynamic Programming	5	38.84 MB	39.25 MB	609 ms	3.50 s
Breadth-First Search	BF	Graph Traversal	24	21.94 MB	37.55 MB	4.57 ms	0.91 s
Kmeans	KM	Dense Linear Algebra	38	67.6 MB	130.38 MB	115.01 ms	8.40 s
Hotspot3D	3D	Structured Grid	100	24 MB	24.3 MB	10.78 ms	11.82 s
LU Decomposition	LUD	Dense Linear Algebra	190	4.15 MB	26.92 MB	2.35 ms	10.18 s
Needleman-Wunsch	NW	Dynamic Programming	255	32.47 MB	104.45 MB	0.29 ms	0.67 s
Gaussian Elimination	GE	Data Mining	510	0.50 MB	1.18 MB	0.26 ms	0.40 s
Myocyte	MC	Biological Simulation	3,913	1.76 MB	2.12 MB	0.13 ms	8.63 s
CFD Solver	CF	Fluid Dynamics	14,004	7.24 MB	19.3 MB	5.53 ms	99.99 s

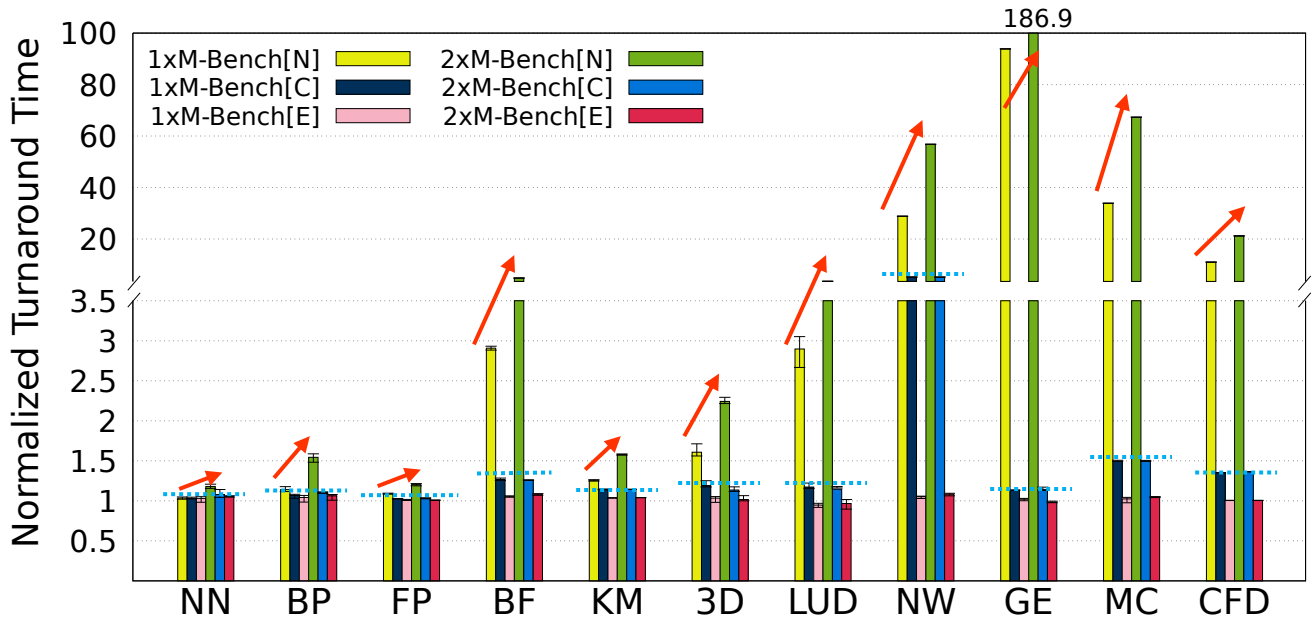
Evaluation Setup (2)

- Micro benchmark (M-Bench)
  - We made simple GPGPU workload to be used for lower priority workload than Rodinia benchmark's workloads
  - M-Bench running in background disturb the high priority task

- Priority configuration
  - [E] : Emergency priority
    - This priority is the highest priority and does not require snapshotting
  - [C] : Common priority
    - This priority is a higher priority than M-Bench
  - [N] : Normal environment
    - This is result of evaluation in vanilla kernel

Performance of High Priority Task

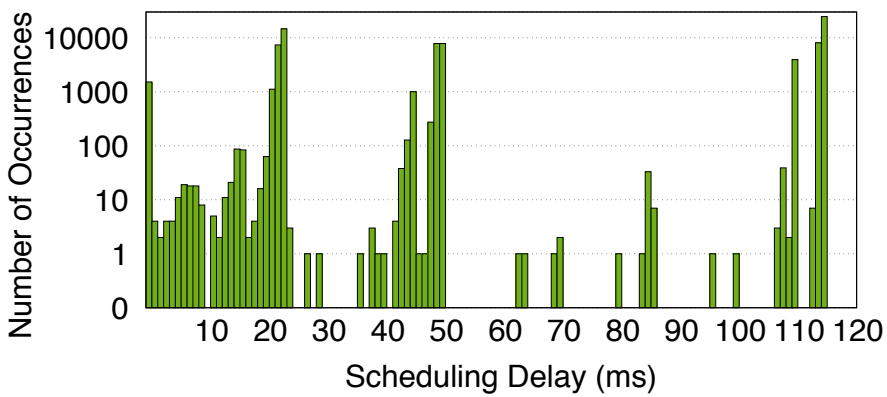
- Performance of a high priority task
  - Degrading performance up to 186.9× depending on low priority tasks on normal environment
  - Guaranteeing consistent performance regardless of background load on Transactionization scheme



Scheduling Delay on Original Environment

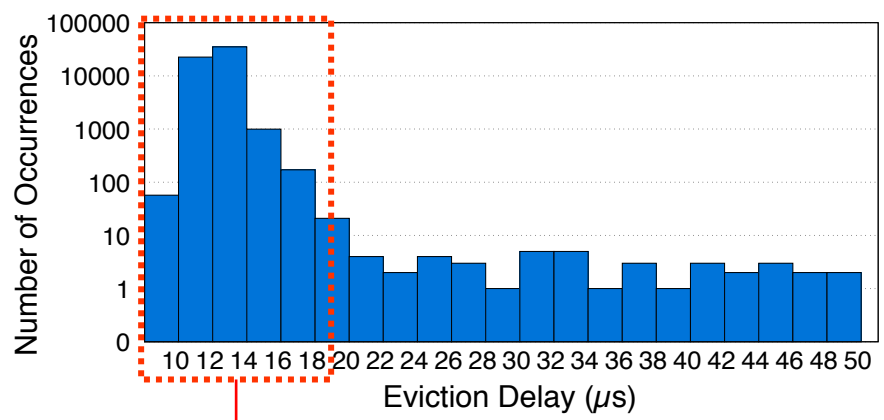
- Histogram of scheduling delay of high priority kernels
  - Scheduling delay depending on the running kernel length in normal environment
- Our approach suppressed the launch delay of high-priority kernels within 18μs

Normal environment



Increased delay due to kernel length

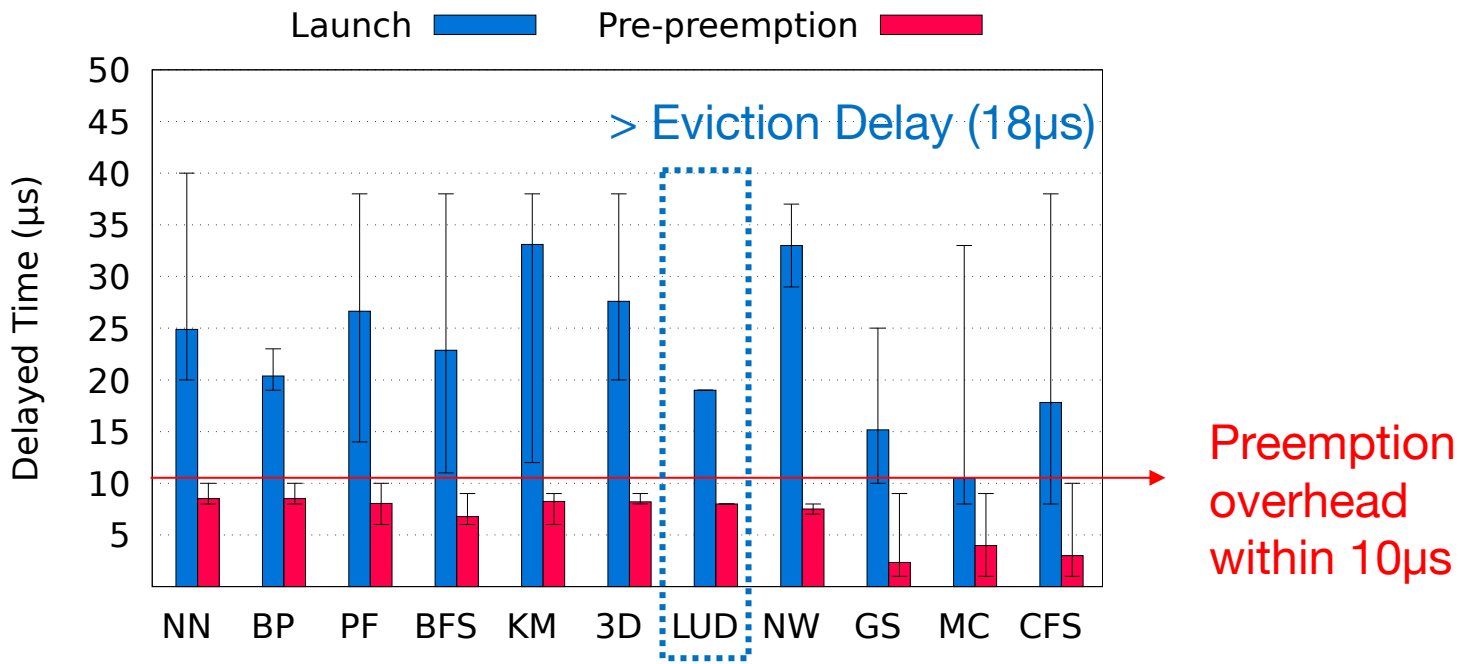
Transactionization scheme



99.9% percentile within 18μs

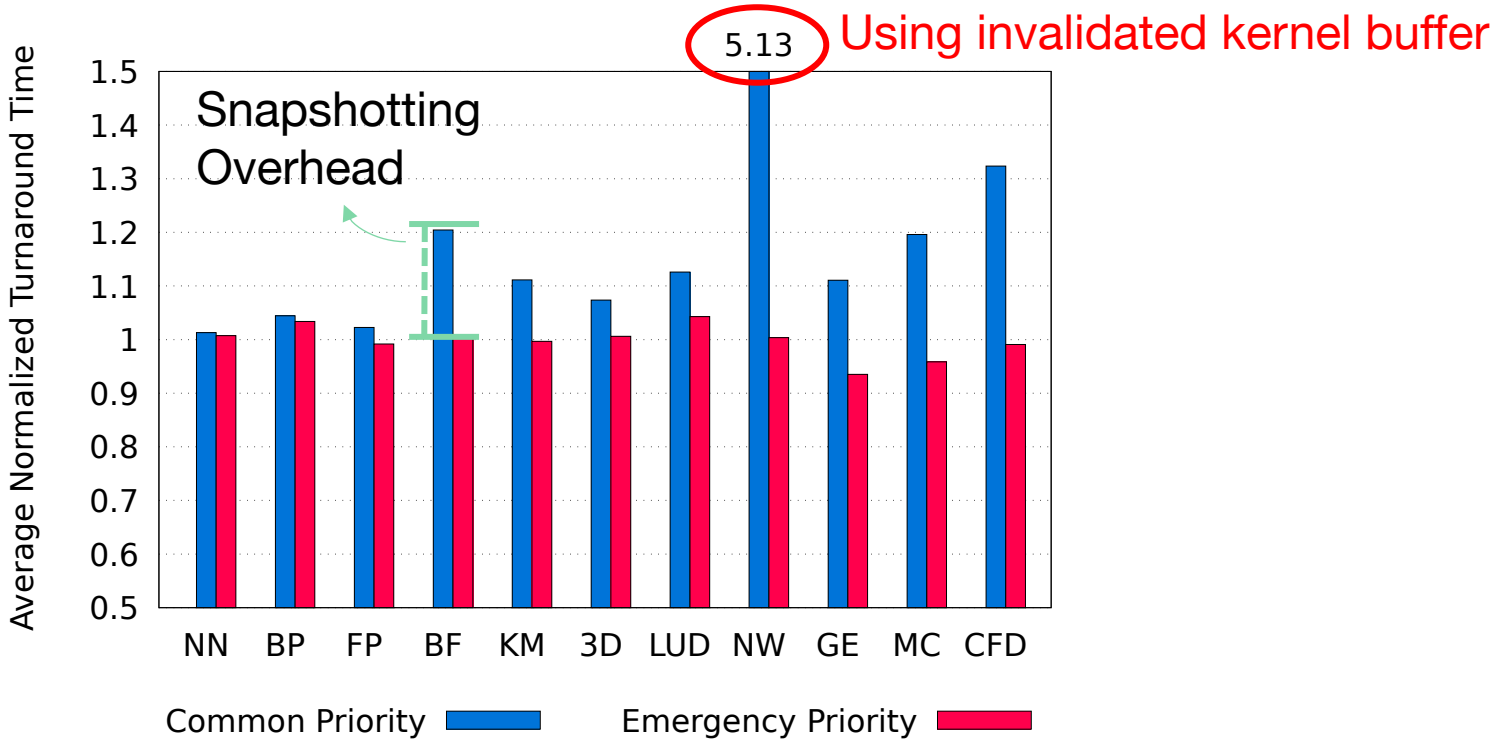
Analysis of Preemption Delay

- There is no delay caused by eviction in the preemption process since it exceeds the majority of 18μs



Snapshotting Overhead

- Our scheme has significant overhead due to snapshotting
  - Advantages of performance improvement due to launch without snapshotting at emergency priority



Discussion

- Reducing snapshot overhead through high bandwidth memory (HBM)
  - Exynos 8890 provides up to 51 GB/s memory bandwidth while the device used in our evaluation presents 28.7 GB/s
  - Snapshot process is more efficient when using HBM because it implements memory copying using ARM's vector instruction set

- Selective snapshot process using compile-time hints
  - GPU kernel is idempotent sometimes
  - If enabling to detect idempotent kernels, then selective snapshot processes can be used

- Introduction
- Motivation
- Related Work
- Background
- Our Approach
  - Design
  - Implementation
- Evaluation
- **Conclusion**

Conclusion

- A GPU kernel transactionization for preemptive scheduling
  - We proposed a GPU kernel transactionization scheme that enables immediate abortion and re-execution of a GPU kernel
  - Based on the transactionization scheme, we developed a pseudo-preemptive GPU kernel scheduler
- Evaluation results showed that the scheduling delay for the urgent task was reduced to approximately 18μs
- Source code is available to public at
  - https://github.com/Hyunsu-Lee/psched_gpu

Thank you

Q&A

Backup Slide (1)

- Multiple priority evaluation