GPUpIO: The Case for I/O-Driven Preemption on GPUs

Lior Zeno, Avi Mendelson, Mark Silberstein

Technion – Israel Institute of Technology
What
I/O-driven preemption support for programs running on GPU

Why
Blocking I/O operations result in underutilized hardware

Results
A complete prototype that implements a preemption mechanism

1.8x throughput
Motivation: Direct GPU I/O

- **GPUfs** and **GPUnet** provide high level I/O abstractions for GPU programs.
- Easier to program, data-driven data access.
- Native GPU applications.
Inside a native GPU server

But look, there is a problem here:

```
recv()
GPU_compute()
send()
```
Inside a native GPU server

Suggestion – how about swapping SM and Scheduler queue
Typically normal people thinking left to right
Inside a native GPU server

Scheduler work queue

I/O threadblock

SM

TB

TB
Inside a native GPU server

Scheduler work queue

Compute threadblock

SM

TB

TB
Inside a native GPU server

Scheduler work queue

TB

Lior Zeno - EE@Technion
Inside a native GPU server

TB is scheduled

Scheduler work queue
Inside a native GPU server

Scheduler work queue

recv()
Inside a native GPU server

recv()  GPU_compute()  time

Scheduler work queue

SM  TB

TB
Inside a native GPU server

recv()  GPU_compute()  send()  

time
Inside a native GPU server

TB ran to completion

Scheduler work queue

recv()  GPU_compute()  send()  

Time
Inside a native GPU server

Scheduler work queue

recv()  GPU_compute()  send()
Inside a native GPU server

Scheduler work queue

recv() GPU_compute() send() compute()
The Problem

I/O threadblocks occupy the SM while waiting for I/O operations to complete

Add a call out – need preemption
Vision: Preemption mechanism
With Preemption

recv() is called
With Preemption

Yield its execution

Scheduler work queue

TB

time
With Preemption

The block is enqueued to the wait queue
With Preemption

The TB is scheduled

Scheduler work queue

Scheduler wait queue
With Preemption

Scheduler work queue

Scheduler wait queue

TB

SM

TB

time
With Preemption

Scheduler work queue

Scheduler wait queue

The I/O operation completes

TB

time
With Preemption

The TB is enqueued to the work queue

Scheduler wait queue
With Preemption

Scheduler work queue

Scheduler wait queue

compute()
With Preemption

The TB completes its execution
With Preemption

The TB is scheduled

Scheduler work queue

Scheduler wait queue

compute()
With Preemption

Scheduler work queue

Scheduler wait queue

compute()  GPU_compute()
With Preemption

Scheduler work queue

Scheduler wait queue

send() is called

compute()  GPU_compute()
With Preemption

The block is enqueued to the wait queue

Scheduler work queue

Scheduler wait queue

compute() GPU_compute()
With Preemption

The I/O operation completes
With Preemption

The TB is enqueued to the work queue

Scheduler wait queue

compute() GPU_compute()
With Preemption

The TB is scheduled and runs to completion.
Preemption in traditional OSes

• Preemptive multitasking
  – Scheduling policy
  – I/O-driven

• Advantages:
  – Throughput
  – Resource sharing (fairness)
  – Interactivity
CPU Scheduling FSM

- new
  - admitted
  - interrupt
  - I/O or event completion
  - Scheduler dispatch
  - waiting
  - running
    - exit
    - terminated

Lior Zeno - EE@Technion
CPU ≠ GPU

• Scheduling is implemented in hardware
• Threadblocks cannot yield
  - GPU complex memory hierarchy

• Hardware context switch support is not needed
  - Most modern OSes do context switches in software, although x86 provides full hardware support.

Suggestion – I would use the diagram in the previous slide to mark which parts of the traditional system DO NOT work...
GPUpIO: I/O-Driven Preemption Support for GPUs

- Design
- Implementation
- Evaluation
- Limitations
Design

• Checkpoint-restart
  – Checkpoint the state and seamlessly restart execution

• Yield-restore
  – Yield execution and restart at a later time using the checkpointed state
Design Considerations

- Preemption Granularity
  - Kernel-level preemption (coarse-grained)
    - Low performance
  - Warp-level preemption (fine-grained)
    - Evicting a few warps of a TB is useless
    - I/O libraries provide a TB-level API

What is the bottom line?? TB-level preemption, right?
System Overhead Characterization

- Consider a GPU with 1 SM.
- Parameters: $C$ (compute time), $I$ (I/O time), $\alpha$ (preemption overhead).
- Define compute intensity: $\frac{C}{I}$
- Execution time without preemption: $I + C$
- Execution time with preemption: $\max(I, C) + \alpha$

It's not clear what question you are asking. Like: does it make sense to implement preemption without hardware support???

Due to overlap

Font of the equations are of different sizes
Expected performance gains of preemption

Make the graphs take most of the space on the slide

Note the change – update all the slides
Performance comparison

Add annotation to explain what beta means.
Performance comparison

50% speedup, even for high overhead
Performance comparison

![Graph showing performance comparison]
Performance comparison

Maximal speedup is achieved with balanced pipeline

What is “pipeline” With I/O=compute
Performance comparison

Less important for compute-intensive workloads

You need a bottom line slide saying - here are our conclusions from the model. Basically repeating the graphs

Maybe replace this graph with the one which shows much higher values of compute intensity. It’s still 50% here
Checkpoint-restart for a single TB

• Large context
  – Compiler assisted checkpoint (live variable analysis)

Need a transition slide saying – now we are talking about Checkpoint- Restore implementation
The Cost of checkpoint-restore per TB
(256 threads per TB)
Yield-Restart

- CPU cannot interrupt a running GPU kernel
  - Cooperative preemption
- Lack of control on hardware work queues
  - Scheduling TB-at-a-time with the appropriate context
Nested Parallelism

Not sure what this slide is doing here
Yield-Restart Overhead

![Graph showing Yield-Restart Overhead with I/O Kernels on the x-axis and Re-Invocation Delay in microseconds on the y-axis. The graph indicates that as the number of I/O Kernels increases, the re-invocation delay also increases significantly.]
Yield-Restore Overhead

Low restore overhead

It should all be RESTART, not restore.
The Full Picture

Very good idea, but I'd show that the TB is enqueued in CPU on the way back. This will be consistent with the picture where you want the GPU to reenqueue the block.
The Full Picture

The TB reads from disk

- GPU
- TB
- CPU
- PCIe bus
- Disc
The Full Picture

The data is not on GPU. The TB yields (checkpoint).
The Full Picture

The I/O operation completes.

 GPUs
 PCIe bus
 CPU
 Interrupt
 Disc
The Full Picture

The TB is enqueued in a new stream.
Implementation Bits

- Compilation process
  - Live variable analysis
  - Kernel modifications
  - Final code generation

- Run-time support
  - Memory management
  - Scheduling
Evaluation

• Setup
  – Intel Xeon E5-2620 CPU
  – NVIDIA Tesla K40m
  – HDD Seagate Barracuda ST1000DM003 7200 RPM

• End-to-end benchmark
  – How many Tbs, how long are each kernel, how much I/O, how much compute?
  – I/O kernels: Read two matrices A and B, from disk and compute the product $AxB^T$
  – Computer kernels: 3 samples from the Rodinia Suite
Results

![Graph showing speedup with varying I/O Kernels]

The graph above illustrates the speedup achieved with increasing I/O Kernels. As the number of I/O Kernels increases from 1 to 10, the speedup also increases, reaching a significant boost at 10 I/O Kernels.
Results

![Graph showing speedup with varying I/O kernels]

- Full Overlap (Disk)

- Speedup

- I/O Kernels
Results

Higher speedup than expected
More than full-overlap

• Without preemption:

• With Preemption:
Limitation of software approach

I/O-intensive workload

Mixed workload
Limitation of software approach

good bottomline, but must annotate the graphs to explain exactly what you are looking at.

I/O kernels are starved by compute kernels

I/O-intensive workload

Mixed workload
We need hardware support

- High overhead for CR and PR.
- I/O kernels are starved due to the limited interface to the internal HW queues in the GPU.

- HW support is crucial for performance and scheduling fairness.
- See the paper for Yield – an HW extension.
Related work

• Preemption on GPUs
  – Chimera (Static preemption, no CR)

• Checkpoint restore
  – NVCR/CheCUDA (kernel boundary)

• I/O support
  – GPUfs, GPUnet

• Compiler support for GPUs
  – Dandelion

• Scheduling
  – TimeGraph, PTask
GPUpIO is a library providing a preemption mechanism for GPUs

https://github.com/liorze/gpupio

liorz@tx.technion.ac.il
Backup

• Implementation
  – Full layout of a transformed kernel
  – Talk about implementation bits, compiler internals

• Yield

• More graphs