Robust Real-Time Multiprocessor Interrupt Handling Motivated by GPUs

Glenn Elliott
gelliott@cs.unc.edu

Jim Anderson
anderson@cs.unc.edu

The University of North Carolina at Chapel Hill

Work supported by NSF grants CNS 1016954 and CNS 1115284; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.
Real-Time GPUs
Real-Time GPUs

- Graphics processing units (GPUs) can now be programmed with high-level languages to solve general purpose problems
- Practice called “GPGPU”
Real-Time GPUs

• Graphics processing units (GPUs) can now be programmed with high-level languages to solve general purpose problems

• Practice called “GPGPU”

• Why use GPUs in real-time systems?
Primary Motivation: Performance

![Graph showing GFLOPS (single precision) from 1/1/03 to 1/1/12 for GPU (Nvidia) and CPU (Intel).]
Domains for GPUs

- GPUs excel at data parallel problems
  - Digital signal processing
  - Matrix-like computations
  - Sorting and searching
Future Automotive Applications

• Vehicle and pedestrian detection
• Object tracking
• Fusion of video, laser, and radar sensor data
• Clear real-time implications!
Target Platform
Target Platform

• We want to develop a system using components available today
Target Platform

• We want to develop a system using components available today

• Current state of technology motivates the following platform:
Target Platform

• We want to develop a system using components available today

• Current state of technology motivates the following platform:

  • Multicore system with one or more GPUs
Target Platform

• We want to develop a system using components available today

• Current state of technology motivates the following platform:
  • Multicore system with one or more GPUs
  • Soft real-time
Target Platform

• We want to develop a system using components available today

• Current state of technology motivates the following platform:
  • Multicore system with one or more GPUs
  • Soft real-time
  • Linux-based operating system
Challenges: I/O Device
Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use
Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use

2. Not directly schedulable like a CPU
   - Allocation/arbitration issues
Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use

2. Not directly schedulable like a CPU
   - Allocation/arbitration issues

3. Interrupt-driven communication
Synchronous GPU Usage Pattern
Synchronous GPU Usage Pattern

CPU

GPU

Monday, July 16, 12
J1 sends work to the GPU and blocks waiting for results.
J1 sends work to the GPU and blocks waiting for results.
Synchronous GPU Usage Pattern

JI sends work to the GPU and blocks waiting for results.

An interrupt from the GPU signals that work has completed. (Handler not depicted.)
Synchronous GPU Usage Pattern

J1 sends work to the GPU and blocks waiting for results.

An interrupt from the GPU signals that work has completed.

(Handler not depicted.)
Asynchronous GPU Usage Pattern
Asynchronous GPU Usage Pattern

J1 may continue executing before blocking...
Asynchronous GPU Usage Pattern
Asynchronous GPU Usage Pattern

...or may never block if GPU finishes before J1 needs results.
Asynchronous GPU Usage Pattern

...or may never block if GPU finishes before J1 needs results.

This case has interesting implications on interrupt handling in global schedulers.
Interrupt Handling
Interrupt Handling

- May arrive at **unpredictable** moments
Interrupt Handling

- May arrive at unpredictable moments
- Interrupt preempt {currently scheduled task and} prevents this task from resuming until interrupt is handled
Interrupt Handling

• May arrive at unpredictable moments

• Interrupt preempts currently scheduled task and prevents this task from resuming until interrupt is handled

• A CPU must acknowledge interrupt and may often perform additional computations
Interrupt Handling

- A CPU must **acknowledge** interrupt and may often **perform additional computations**
Interrupt Handling

• A CPU must acknowledge interrupt and may often perform additional computations

• Handling often split:
Interrupt Handling

- A CPU must **acknowledge** interrupt and may often **perform additional computations**
- Handling often **split**:
  - **Top Half**: performs **acknowledgement**
Interrupt Handling

- A CPU must **acknowledge** interrupt and may often **perform** additional computations

- Handling often **split**:
  - **Top Half**: performs **acknowledgement**
  - **Bottom Half**: performs **computations**
Interrupt Handling

CPU0

CPU1
Interrupt Handling

CPU0

CPU1

J1
Interrupt Handling

CPU0

CPU1

Interrupt
Interrupt Handling

Interrupt

CPU0  I  Non-Split Handler

CPU1
Interrupt Handling

Interrupt

CPU0

J1
Non-Split Handler

CPU1

J2

Monday, July 16, 12
Interrupt Handling

CPU0

- J1
- Non-Split Handler
- J1

CPU1

- J2

Interrupt

Monday, July 16, 12
Interrupt Handling

Blocked J1 may experience a priority inversion.
Interrupt Handling

Interrupt

CPU0:
- J1
- Non-Split Handler
- J1

CPU1:
- J2
Interrupt Handling

CPU0

J1  Top  Bottom  J1

Interrupt

CPU1

Split interrupt handler.
(Linux still often executes these together by default.)

Monday, July 16, 12
Interrupt Handling

Interrupt

CPU0

J1  Top  Bottom  J1

CPU1

J2

Monday, July 16, 12
Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?

Monday, July 16, 12
Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?

Need to know:
1) Priority of interrupt
2) “Owner” of interrupt
Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?

Need to know:
1) Priority of interrupt
2) “Owner” of interrupt

Why?
Interrupt Ownership

CPU0

CPU1

GPU
Interrupt Ownership

CPU0

CPU1

GPU

J1

J2
Interrupt Ownership

Synchronous I/O

CPU0

J1

CPU1

J2

GPU
Interrupt Ownership

Synchronous I/O

CPU0

CPU1

GPU

J1

J2

J1
Interrupt Ownership

Synchronous I/O

CPU0

CPU1

GPU

Monday, July 16, 12
Interrupt Ownership

Synchronous I/O

CPU0

CPU1

GPU

Interrupt

J1

J2

Top

Monday, July 16, 12
Interrupt Ownership

Synchronous I/O

Schedule bottom half. Thread inherits priority of J1.

CPU0

J1

J1: Bottom

CPU1

J2 Top J2

GPU

J1

Interrupt

Monday, July 16, 12
Interrupt Ownership

Schedule bottom half. Thread inherits priority of J1.

CPU0

J1: Bottom

J1

J1

CPU1

J2

Top

J2

GPU

J1

Interrupt
Interrupt Ownership

CPU0

CPU1

GPU
Interrupt Ownership

CPU0

CPU1

GPU

J1

J2
Interrupt Ownership

Asynchronous I/O

CPU0

CPU1

GPU

J1

J2

J1
Interrupt Ownership

CPU0

CPU1

GPU

Interrupt
Interrupt Ownership

CPU0

CPU1

GPU

Interrupt
Interrupt Ownership

Schedule bottom half.
Thread inherits priority of J1.
Interrupt Ownership

CPU0

CPU1

GPU

J1

J2

Top

J1: Bottom

Interrupt
Interrupt Ownership

CPU0

J1

J2

CPU1

J2

Top

J1: Bottom

J1

GPU

Interrupt
Interrupt Ownership

CPU0

CPU1

GPU

Interrupt
Two threads with same identity!
Breaks single threaded sporadic task model!
Interrupt Ownership

This can only occur under global scheduler with asynchronous I/O.

---

- CPU0
  - J1
  - J2

- CPU1
  - J2
  - Top
  - J1: Bottom
  - J1

- GPU
  - J1

---

Interrupt

Monday, July 16, 12
Interrupt Ownership

CPU0
J1
J2

CPU1
J2
Top
J1: Bottom
J1

GPU
J1

Saturday, July 16, 12
Interrupt Ownership

Defer bottom half until J1 suspends, preventing co-scheduling.
Defer bottom half until J1 suspends, preventing co-scheduling.

Account for bottom half execution time as being of J1.
Globally Scheduled GPU Interrupt Handlers
Globally Scheduled GPU Interrupt Handlers

• Real-Time GPU Interrupts:
Globally Scheduled GPU Interrupt Handlers

• Real-Time GPU Interrupts:
  1. Thread bottom halves of GPU interrupts, inheriting priority of owners
Globally Scheduled GPU Interrupt Handlers

• Real-Time GPU Interrupts:

  1. **Thread bottom halves** of GPU interrupts, inheriting priority of owners

  2. **Prevent co-scheduling** of bottom halves and owners
Globally Scheduled GPU Interrupt Handlers

• Real-Time GPU Interrupts:
  1. **Thread bottom halves** of GPU interrupts, inheriting priority of owners
  2. **Prevent co-scheduling** of bottom halves and owners

• **PROBLEM:** GPU driver is **closed source.**

  • **Which** GPU raised the interrupt?
  • **What is the priority** of the bottom half?
GPU Interrupt Handling

Callback Function Pointer → GPU Driver

Bottom Half Data

Callback Arguments
(binary blob)

GPU Registry

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$T_1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k$</td>
<td></td>
</tr>
</tbody>
</table>

$k$-exclusion lock ($k$-FMLP)

LITMUS$^R_T$ Scheduler

GPU Index $k$

threaded bottom half

Callback Pointers

GPU Driver

Bottom Half Data

Callback Arguments
(binary blob)

GPU Index $k$

GPU Registry

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$T_1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k$</td>
<td></td>
</tr>
</tbody>
</table>

LITMUS$^R_T$ Scheduler

GPU Index $k$

threaded bottom half

Callbacks

Function Pointer

Callback Arguments
(binary blob)
GPU Allocation

![Diagram showing GPU allocation process]

- **Callback Function Pointer**
  - Connected to **GPU Driver**
  - Input: Bottom Half Data, Callback Arguments
    - Bottom Half Data: (binary blob)
    - Callback Arguments
- **GPU Registry**
  - Input: GPU Index
  - Output: Task
  - Data Structure:
    | GPU Index | Task |
    |-----------|------|
    | 0         | T1   |
    | ...       | ...  |
    | k         |      |

- **LITMUS RT Scheduler**
  - Input: threaded bottom half

**Monday, July 16, 12**
GPU Allocation

$k$-exclusion lock
($k$-FMLP)

GPU Registry

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$T_1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$n$

$k$

$k$

21
GPU Allocation

Table records GPU allocation assignments.

\[ k \text{-exclusion lock} \quad (k\text{-FMLP}) \]

**GPU Registry**

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T_1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Monday, July 16, 12
GPU Allocation

Callback Function Pointer → GPU Driver

Bottom Half Data

Callback Arguments
(binary blob)

GPU Registry

k-exclusion lock (k-FMLP)

GPU Index

Task

0

T_1

...

...


GPU Index

klmirqd Thread

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

LITMUS_RT Scheduler

threaded bottom half

Monday, July 16, 12
**klmirqd GPU Threads**

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

Monday, July 16, 12
### klmirqd GPU Threads

One klmirqd thread per GPU.

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td></td>
</tr>
</tbody>
</table>
klmirqd GPU Threads

Callback Function Pointer

GPU Driver

Callback Arguments
(binary blob)

GPU Registry

k-exclusion lock (k-FMLP)

GPU Index

Task

0

T_1

...

...

LITMUS\textsuperscript{RT}

Scheduler

threaded bottom half

GPU Index klmirqd Thread

0

...

k
Interrupt Interception

Callback Function Pointer → GPU Driver

Bottom Half Data

Callback Arguments
(binary blob)

GPU Registry

k-exclusion lock (k-FMLP)

LITMUS\textsuperscript{RT} Scheduler

threaded bottom half
Spawned by top half.

Interrupt Interception

Callback Function Pointer

GPU Driver

Bottom Half Data

Callback Arguments

(binary blob)

... GPU Index ...

Spawned by top half.
Interrupt Interception

Callback Function Pointer → GPU Driver

Bottom Half Data

Callback Arguments

(binary blob)

... GPU Index ...

Pointer address identifies bottom half as from GPU driver.
Interrupt Interception

- Callback Function Pointer
- Bottom Half Data
- Callback Arguments
  (binary blob)
  - GPU Index

GPU index in blob. Location reversed engineered.
Interrupt Interception

Bottom Half Data

Callback Function Pointer

Callback Arguments
(binary blob)

GPU Driver

k-exclusion lock (k-FMLP)

GPU Registry

GPU Index
0
...
k

LITMUSRT Scheduler

threaded bottom half

GPU Index | klmqrd Thread
---|---
0 | ... |
... | ... |
k | ... |
GPU Interrupt Handling

Bottom Half Data

Callback Function Pointer

GPU Driver

Callback Arguments
(binary blob)

k-exclusion lock (k-FMLP)

GPU Registry

GPU Index

Task

0

T1

...

...

...

...

LITMUSRT Scheduler

threaded bottom half

GPU Index klmirqd Thread

0

...

k

Monday, July 16, 12
GPU bottom half arrives.

Callback Function Pointer

Bottom Half Data

Callback Arguments
(binary blob)

GPU Index

GPU Driver

k-exclusion lock (k-FMLP)

GPU Registry

GPU Index

Task

0

T_{1}

...

...

...

...

k

LITMUS^{RT}
Scheduler

threaded bottom half

Monday, July 16, 12
GPU Interrupt Handling

GPU index extracted.

Bottom Half Data

Callback Function Pointer

GPU Driver

k-exclusion lock (k-FMLP)

GPU Registry

GPU Index

Task

0

T_1

...

...

k

GPU Index

Glimq Thread

0

...

k

LITMUS RT
Scheduler

thraedd bottom half

Monday, July 16, 12
GPU Interrupt Handling

Callback Function Pointer

Bottom Half Data

Callback Arguments
(binary blob)

GPU Registry

GPU owner identified.

LITMUS RT Scheduler

threaded bottom half

GPU Index

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

GPU Index

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

GPU owner identified.

Monday, July 16, 12
GPU Interrupt Handling

Bottom Half Data

Callback Function Pointer

GPU Driver

Callback Arguments

(binary blob)

GPU Index

k-exclusion lock (k-FMLP)

GPU Registry

LITMUS RT Scheduler

GPU Index klmirqd Thread

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

k-lmirqd thread identified.

threaded bottom half

n

Monday, July 16, 12
GPU Interrupt Handling

klmirqd thread inherits priority of GPU owner and executes bottom half callback.

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

GPU Registry

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

LITMUS<sup>RT</sup> Scheduler

threaded bottom half

Callback Function Pointer → GPU Driver

n → ... → k

k-exclusion lock (k-FMLP)
GPU Interrupt Handling

Scheduler prevents co-scheduling of owner and klmirqd thread.

LITMUS$^\text{RT}$ Scheduler

GPU Indexklmirqd Thread

<table>
<thead>
<tr>
<th>GPU Index</th>
<th>klmirqd Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

• Test platform:
Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
  - Eight NVIDIA GTX-470 GPUs
Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
  - Eight NVIDIA GTX-470 GPUs
  - Scheduled in clusters along NUMA boundaries
Evaluation

- **Test platform:**
  - **Two six-core** Xeon X5060 processors at 2.67GHz
  - **Eight** NVIDIA GTX-470 GPUs
  - Scheduled in **clusters** along NUMA boundaries
  - **One** X5060 and **four** GPUs per cluster
Effect on Priority Inversions
Effect on Priority Inversions

- **Inversions** measured in LITMUS\textsuperscript{RT} for:
Effect on Priority Inversions

• Inversions measured in LITMUS$^\text{RT}$ for:

• klmirqd
Effect on Priority Inversions

- **Inversions** measured in LITMUS\textsuperscript{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
Effect on Priority Inversions

- **Inversions** measured in LITMUS\(^{RT}\) for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)
Effect on Priority Inversions

- **Inversions** measured in LITMUS\textsuperscript{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)

- **Executed** 41 task sets with utilizations [7.5, 11.5] each for 2 minutes
Effect on Priority Inversions

- **Inversions** measured in LITMUS\textsuperscript{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)

- **Executed** 41 task sets with utilizations [7.5, 11.5] each for 2 minutes

- Every task used GPUs *asynchronously*
Effect on Priority Inversions

- **Inversions** measured in LITMUS\textsuperscript{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)

- **Executed** 41 task sets with utilizations $[7.5, 11.5]$ each for 2 minutes

- Every task used GPUs *asynchronously*

- Scheduled under **Clustered EDF**
Effect on Priority Inversions

Distribution of Priority Inversions: Task Set Utilization of 11.2

Priority Inversion Duration (microseconds)

P(Inversion Duration) <= X

Monday, July 16, 12
Priority inversion durations decreased.

Distribution of Priority Inversions: Task Set Utilization of 11.2

P(Inversion Duration) <= X

Priority Inversion Duration (microseconds)
Priority inversion durations decreased.

Distribution of Priority Inversions: Task Set Utilization of 11.2

- klmirqd: 90% inversions <5µs

Priority inversion durations decreased.
Effect on Priority Inversions

Priority inversion durations decreased.

Distribution of Priority Inversions: Task Set Utilization of 11.2

PAI: 90% inversions <35µs

[1] klmirqd
[2] PAI
[3] SLIH

Monday, July 16, 12
Priority inversion durations decreased.

SLIH: 90% inversions <40µs (with long tail)
Overhead-Aware Schedulability Experiments
Overhead-Aware Schedulability Experiments

- Gathered overhead measurements for many system tasks (such as scheduling)
Overhead-Aware Schedulability Experiments

- **Gathered** overhead measurements for many system tasks (such as scheduling)
- **Incorporated** overheads into *soft* real-time schedulability experiments
- Task sets a *mix* of GPU-using and CPU-only
Overhead-Aware Schedulability Experiments

- **Gathered** overhead measurements for many system tasks (such as scheduling)

- **Incorporated** overheads into *soft* real-time schedulability experiments

- Task sets a **mix** of GPU-using and CPU-only

- **Different accounting techniques are required** for each interrupt handling method
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

- [1] klmirqd, $i=1$
- [2] PAI, $i=1$
- [3] SLIH, $i=1$
- [4] klmirqd, $i=3$
- [5] PAI, $i=3$
- [6] SLIH, $i=3$
- [7] klmirqd, $i=6$
- [8] PAI, $i=6$
- [9] SLIH, $i=6$

Ratio of Schedulable Task Sets (soft)

CPU Utilization (prior inflations)
Overhead-Aware Schedulability Experiments

Ratio of Schedulable Task Sets (soft)

CPU Utilization (prior inflations)

$i$ is number of interrupts generated per job

- [1] klmirqd, $i=1$
- [2] PAI, $i=1$
- [3] SLIH, $i=1$
- [4] klmirqd, $i=3$
- [5] PAI, $i=3$
- [6] SLIH, $i=3$
- [7] klmirqd, $i=6$
- [8] PAI, $i=6$
- [9] SLIH, $i=6$

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

[1] klmirqd, i=1
[2] PAI, i=1
[3] SLIH, i=1

[4] klmirqd, i=3
[5] PAI, i=3
[6] SLIH, i=3

[7] klmirqd, i=6
[8] PAI, i=6
[9] SLIH, i=6

klmirqd, w/ i=6, 50% of task sets with utilization of \(~10.5\) are schedulable.
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

Ratio of Schedulable Task Sets (soft)

[1] klmirqd, i=1
[2] PAI, i=1
[3] SLIH, i=1
[4] klmirqd, i=3
[5] PAI, i=3
[6] SLIH, i=3
[7] klmirqd, i=6
[8] PAI, i=6

PAI & SLIH, w/ i=6, 50% of task sets with utilization of ~9.1 are schedulable.
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

Ratio of Schedulable Task Sets (soft)

Effective CPU Utilization (prior inflations)

[1] klmirqd, i=1
[2] PAI, i=1
[3] SLIH, i=1
[4] klmirqd, i=3
[5] PAI, i=3
[6] SLIH, i=3
[7] klmirqd, i=6
[8] PAI, i=6
[9] SLIH, i=6
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

“Effective CPU utilization” presumes that each GPU provides 16x utilization.
Overhead-Aware Schedulability
Experiments

klmirqd, w/ \( i=6 \), 50% of task sets with effective CPU utilization of \(~73.0\) are schedulable.
Overhead-Aware Schedulability Experiments

Crit. Sec. Exe 75%; GPU Task Share [50, 60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]

PAI & SLIH, w/ i=6, 50% of task sets with effective CPU utilization of ~61.0 are schedulable.
Conclusion
Conclusion

• Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind
Conclusion

- Developed method for **threaded interrupt handling under global scheduling with asynchronous I/O in mind**

- Integrated **closed source GPU driver** through interrupt **interception and decoding**
Conclusion

- Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind
- Integrated closed source GPU driver through interrupt interception and decoding
- Evaluations indicate klmirqd significantly reduces priority inversions while avoiding schedulability analysis pitfalls
Conclusion

• Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind

• Integrated closed source GPU driver through interrupt interception and decoding

• Evaluations indicate klmirqd significantly reduces priority inversions while avoiding schedulability analysis pitfalls

• Source available at www.litmus-rt.org
Thank you!

Questions?
Backup Slides
# Measured Overheads

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Average Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>0.63</td>
</tr>
<tr>
<td>Context Switch</td>
<td>0.36</td>
</tr>
<tr>
<td>IPI</td>
<td>0.60</td>
</tr>
<tr>
<td>Job Release</td>
<td>0.67</td>
</tr>
<tr>
<td><strong>Top Half</strong></td>
<td><strong>16.44</strong></td>
</tr>
<tr>
<td><strong>Bottom Half</strong></td>
<td><strong>29.90</strong></td>
</tr>
<tr>
<td>klmirqd Release</td>
<td>1.39</td>
</tr>
<tr>
<td>PAI Release</td>
<td>0.13</td>
</tr>
<tr>
<td>PAI Scheduling</td>
<td>0.56</td>
</tr>
</tbody>
</table>
Effect on Priority Inversions

Cumulative Inversion Length: Task Set Utilization of 11.2

[1] klmirqd
[2] PAI
[3] SLIH
Total inversion time reduced.

Cumulative Inversion Length: Task Set Utilization of 11.2

- [1] klmirqd
- [2] PAI
- [3] SLIH

Monday, July 16, 12
Total inversion time reduced.

Cumulative Inversion Length: Task Set Utilization of 11.2

klmirqd: 1.2ms inversion time in 2 minutes
Total inversion time reduced.

Cumulative Inversion Length: Task Set Utilization of 11.2

PAI: 220ms

[1] klmirqd
[2] PAI
[3] SLIH
Total inversion time reduced.

Cumulative Inversion Length: Task Set Utilization of 11.2

- [1] klmirqd
- [2] PAI
- [3] SLIH

SLIH: 350ms
System-Level Evaluation

- Compared klmirqd against SLIH, PAI, and PREEMPT_RT (real-time Linux patch)
- PREEMPT_RT interrupt handler threads have fixed priority
- Scheduled using Clustered Rate Monotonic
  - Needed to make fair comparisons to PREEMPT_RT
Pathological Task Set

One GPU-using, period = 19.9ms

Four GPU-using, period = 20.1ms

20 CPU-only, period = 20ms

One “sandwich” on each cluster.
## System-Level Evaluation

<table>
<thead>
<tr>
<th>Scheduler:</th>
<th>Operating System:</th>
<th>PREEMPT_RT</th>
<th>Unmod. Linux</th>
<th>LITMUSRT</th>
<th>C-EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt Handling Method:</td>
<td>Low Prio. Interrupts (a)</td>
<td>High Prio. Interrupts (b)</td>
<td>SLIH (c)</td>
<td>SLIH (d)</td>
</tr>
<tr>
<td>Avg. % Miss Per Task</td>
<td>CPU-Only Tasks</td>
<td>12.5%</td>
<td>12.5%</td>
<td>1.6%</td>
<td>10.0%</td>
</tr>
<tr>
<td></td>
<td>GPU-Using Tasks</td>
<td>10.1%</td>
<td>8.5%</td>
<td>6.8%</td>
<td>0%</td>
</tr>
<tr>
<td>Avg. Resp. Time as % Period</td>
<td>CPU-Only Tasks</td>
<td>22474.5%</td>
<td>24061.0%</td>
<td>8992.1%</td>
<td>55.8%</td>
</tr>
<tr>
<td></td>
<td>GPU-Using Tasks</td>
<td>23066.1%</td>
<td>34263.5%</td>
<td>61131.7%</td>
<td>46.7%</td>
</tr>
</tbody>
</table>
# System-Level Evaluation

<table>
<thead>
<tr>
<th>Scheduler: Operating System:</th>
<th>PREEMPT_RT</th>
<th>Unmod. Linux</th>
<th>LITMUS_RT</th>
<th>C-EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Handling Method:</td>
<td>Low Prio. Interrupts (a)</td>
<td>High Prio. Interrupts (b)</td>
<td>SLIH (c)</td>
<td>SLIH (d)</td>
</tr>
<tr>
<td><strong>Avg. % Miss Per Task</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-Only Tasks</td>
<td>12.5%</td>
<td>12.5%</td>
<td>1.6%</td>
<td>10.0%</td>
</tr>
<tr>
<td>GPU-Using Tasks</td>
<td>10.1%</td>
<td>8.5%</td>
<td>6.8%</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Avg. Resp. Time as % Period</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-Only Tasks</td>
<td>22474.5%</td>
<td>24061.0%</td>
<td>8992.1%</td>
<td>55.8%</td>
</tr>
<tr>
<td>GPU-Using Tasks</td>
<td>23066.1%</td>
<td>2763.5%</td>
<td>61131.7%</td>
<td>46.7%</td>
</tr>
</tbody>
</table>

**GPU starvation**
# System-Level Evaluation

<table>
<thead>
<tr>
<th>Scheduler: Operating System:</th>
<th>C-RM</th>
<th>Unmod. Linux</th>
<th>LITMUSRT</th>
<th>C-EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt Handling Method:</strong></td>
<td>PREEMPT_RT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low Prio. Interrupts (a)</td>
<td>SLIH (c)</td>
<td>SLIH (d)</td>
<td>klmirqd (e)</td>
<td>PAI (f)</td>
</tr>
<tr>
<td>High Prio. Interrupts (b)</td>
<td></td>
<td></td>
<td></td>
<td>PAI (h)</td>
</tr>
<tr>
<td><strong>Avg. % Miss Per Task</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-Only Tasks</td>
<td>12.5%</td>
<td>12.5%</td>
<td>1.6%</td>
<td>10.0%</td>
</tr>
<tr>
<td>GPU-Using Tasks</td>
<td>10.1%</td>
<td>8.5%</td>
<td>6.8%</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Avg. Resp. Time as % Period</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-Only Tasks</td>
<td>22474.5%</td>
<td>24061.0%</td>
<td>8992.1%</td>
<td>55.8%</td>
</tr>
<tr>
<td>GPU-Using Tasks</td>
<td>23066.1%</td>
<td>34263.5%</td>
<td>61131.7%</td>
<td>46.7%</td>
</tr>
</tbody>
</table>

CPU response time increase