INTERFERENCE FROM GPU SYSTEM SERVICE REQUESTS

ARKAPRAVA BASU, JOSEPH L. GREATHOUSE, GURU VENKATARAMANI, JÁN VESELÝ
AMD RESEARCH, ADVANCED MICRO DEVICES, INC.
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”

CPUs  Video Decode  Video Encode
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”

Accelerators from Industry and Academia
MODERN SYSTEMS ARE POWERED BY HETEROGENEITY

6th Gen. AMD A-Series Processor “Carrizo”

Accelerators from Industry and Academia

- Machine Learning
- Databases
- Computer Vision
- Regular Expressions
- Physics
- Graph Analytics
- Finite State Machines
- Genome Sequencing
- Reconfigurable (e.g., FPGA)
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

- CPU
- Data Transfer
- Accelerator
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

CPU

Data Transfer

Accelerator
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

CPU
Data Transfer
Accelerator
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

CPU
Data Transfer
Accelerator
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

CPU
Data Transfer
Accelerator
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

- Work launched by the CPU to accelerators
• Work launched by the CPU to accelerators

• Data marshalled by the CPU
TRADITIONAL HETEROGENEOUS COMPUTING DRIVEN BY CPUS

- Work launched by the CPU to accelerators
- Data marshalled by the CPU
- Communication and computation are coarse-grained
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU

Data Transfer

Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS
MODERN ACCELERATORS ARE EQUAL PARTNERS
MODERN ACCELERATORS ARE EQUAL PARTNERS

CPU
Data Transfer
Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS
MODERN ACCELERATORS ARE EQUAL PARTNERS
MODERN ACCELERATORS ARE EQUAL PARTNERS

- CPU
- Data Transfer
- Accelerator
MODERN ACCELERATORS ARE EQUAL PARTNERS

- Accelerators and CPUs can launch work to one another (and themselves)
MODERN ACCELERATORS ARE EQUAL PARTNERS

- Accelerators and CPUs can launch work to one another (and themselves)
- Data transfer implicit based on usage
• Accelerators and CPUs can launch work to one another (and themselves)

• Data transfer implicit based on usage

• Communication and computation are fine-grained
ACCELERATORS CAN NOW REQUEST OS SERVICES
ACCELERATORS CAN NOW REQUEST OS SERVICES

• GPU-to-CPU Callbacks (Owens et al., UCHPC 2010)
ACCELERATORS CAN NOW REQUEST OS SERVICES

- GPU-to-CPU Callbacks (Owens et al., UCHPC 2010)
- Page Faults / Demand Paging (Veselý et al., ISPASS 2016)
ACCELERATORS CAN NOW REQUEST OS SERVICES

• GPU-to-CPU Callbacks (Owens et al., UCHPC 2010)

• Page Faults / Demand Paging (Veselý et al., ISPASS 2016)

• GPU-Initiated Network Requests (GPUnet, USENIX 2014)
ACCELERATORS CAN NOW REQUEST OS SERVICES

- GPU-to-CPU Callbacks (Owens et al., UCHPC 2010)
- Page Faults / Demand Paging (Veselý et al., ISPASS 2016)
- GPU-Initiated Network Requests (GPUnet, USENIX 2014)
- GPU-Initiated File System Requests (GPUfs, ASPLOS 2013)
ACCELERATORS CAN NOW REQUEST OS SERVICES

• GPU-to-CPU Callbacks (Owens et al., UCHPC 2010)

• Page Faults / Demand Paging (Veselý et al., ISPASS 2016)

• GPU-Initiated Network Requests (GPUnet, USENIX 2014)

• GPU-Initiated File System Requests (GPUfs, ASPLOS 2013)

• “Generic” System Calls (Genesys, ISCA 2018)
  • ioctl() for other devices
  • Memory management (e.g., sbrk(), mmap())
  • Signals
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

CPU 2  CPU 1  CPU 0  GPU

Memory
Interference from GPU System Service Requests

October 5, 2018

How a GPU System Service Request Is Handled

Step 1: Set up request arguments in memory

CPU 2  CPU 1  CPU 0

GPU

Memory
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
Step 3: (a) Schedule bottom-half handler
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
Step 3: (a) Schedule bottom-half handler
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
Step 3: (a) Schedule bottom-half handler
(b) ACK request to GPU
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 1: Set up request arguments in memory
Step 2: Send request interrupt to a CPU
Step 3: (a) Schedule bottom-half handler
       (b) ACK request to GPU
Step 4: (a) Set up kernel work queue arguments
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 4: (a) Set up kernel work queue arguments

CPU 2  →  CPU 1  →  CPU 0  →  GPU

1  →  Memory

3a  →  3b  →  2

4a
Step 4: (a) Set up kernel work queue arguments (b) Queue worker thread
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 4: (a) Set up kernel work queue arguments
(b) Queue worker thread

CPU 2 ——— 4b ——— CPU 1 ——— 3a ——— CPU 0 ——— 3b ——— GPU

Memory

1

2

4a

3b

3a

4b
INTERFERENCE FROM GPU SYSTEM SERVICE REQUESTS

HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

CPU 2 → 4b → CPU 1 ← 3a ← CPU 0 → 3b → GPU

Step 5: Perform SSR

Memory

1 2

4a 4b
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 5: Perform SSR
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 5: Perform SSR
Step 6: Send response to GPU
HOW A GPU SYSTEM SERVICE REQUEST IS HANDLED

Step 5: Perform SSR
Step 6: Send response to GPU
SYSTEM SERVICE REQUESTS CAN LEAD TO DIFFICULTIES

• GPUs and accelerators can request OS (system) services
SYSTEM SERVICE REQUESTS CAN LEAD TO DIFFICULTIES

- GPUs and accelerators can request OS (system) services
- These SSRs can interfere with unrelated CPU-based work
SYSTEM SERVICE REQUESTS CAN LEAD TO DIFFICULTIES

• GPUs and accelerators can request OS (system) services

• These SSRs can interfere with unrelated CPU-based work

• Unrelated CPU-Based work can slow down GPU SSR handling
SYSTEM SERVICE REQUESTS CAN LEAD TO DIFFICULTIES

- GPUs and accelerators can request OS (system) services
- These SSRs can interfere with unrelated CPU-based work
- Unrelated CPU-Based work can slow down GPU SSR handling
- CPUs lose opportunity to sleep because of GPU SSRs
MODES OF INTERFERENCE FROM GPU SSRS

- GPU
- CPU0
- CPU1
- CPU2

- 4b
- 3a
- 3b
- 6
- 4a
- 2
- 5
- 1

- CPU User Work
- GPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

- GPU
- CPU0
- CPU1
- CPU2

CPU User Work

GPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

- GPU
- CPU0
- CPU1
- CPU2

- CPU User Work
- GPU User Work

Numbers indicate the sequence of interactions:
1. 4a
2. 3b
3. 3a
4. 4b
5. 6
6. 5
MODES OF INTERFERENCE FROM GPU SSRS

GPU

CPU0

CPU1

CPU2

CPU User Work

GPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

- GPU
- CPU0
- CPU1
- CPU2

1. Mode/Task Switch
2. CPU User Work
3. GPU User Work

- 4a
- 4b
- 3a
- 3b

- 5
- 6

- 1
- 2
MODES OF INTERFERENCE FROM GPU SSRS

CPU0
CPU1
CPU2

GPU

1. Mode/Task Switch
2. CPU User Work
3. GPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

- CPU0
- CPU1
- CPU2

1. Mode/Task Switch
2. Kernel Task Operation
3. CPU User Work
4. GPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

**CPU0**
- Mode/Task Switch
- CPU User Work
- GPU User Work

**CPU1**
- Mode/Task Switch
- CPU User Work

**CPU2**
- Mode/Task Switch
- GPU User Work

**GPU**
- Mode/Task Switch
- CPU User Work
- GPU User Work

1. **Mode/Task Switch**
2. **CPU User Work**
3. **GPU User Work**
4a. **Mode/Task Switch**
4b. **CPU User Work**
5. **Mode/Task Switch**
6. **GPU User Work**
MODES OF INTERFERENCE FROM GPU SSRS

- **GPU**
- **CPU0**
- **CPU1**
- **CPU2**

- **Mode/Task Switch**
- **CPU User Work**
- **GPU User Work**
- **Kernel Task Operation**

1. **Mode/Task Switch**
2. **CPU User Work**
3a. **GPU User Work**
3b. **CPU User Work**
4a. **Mode/Task Switch**
4b. **CPU User Work**
5. **Mode/Task Switch**
6. **CPU User Work
MODES OF INTERFERENCE FROM GPU SSRS

1. CPU0

   - Mode/Task Switch

2. CPU1

   - CPU User Work

3. CPU2

   - GPU User Work

   - Mode/Task Switch

   - Kernel Task Operation
MODES OF INTERFERENCE FROM GPU SSRS

- **CPU0**
  - Mode/Task Switch
  - CPU User Work
  - Mode/Task Switch
  - GPU User Work

- **CPU1**
  - CPU User Work
  - Mode/Task Switch
  - GPU User Work
  - Kernel Task Operation

- **CPU2**
  - CPU User Work
  - Mode/Task Switch
  - GPU User Work
  - Kernel Task Operation
  - Low-Performance User-Mode Operation
MODES OF INTERFERENCE FROM GPU SSRS

CPU0

CPU1

CPU2

GPU

Mode/Task Switch

Low-Performance User-Mode Operation

CPU User Work

GPU User Work

Kernel Task Operation

1. Mode/Task Switch
2. GPU User Work
3. Low-Performance User-Mode Operation
4. CPU User Work

October 5, 2018
MODES OF INTERFERENCE FROM GPU SSRS

- **CPU0**
  - 2
  - 3
  - 3a
  - 4

- **CPU1**
  - 3a
  - 4
  - 4b

- **CPU2**
  - 4b

- **GPU**
  - 2
  - 3

Legend:
- **CPU User Work**
- **GPU User Work**
- **Mode/Task Switch**
- **Kernel Task Operation**
- **Low-Performance User-Mode Operation**
MODES OF INTERFERENCE FROM GPU SSRS

- **GPU**
- **CPU0**
  - Mode/Task Switch
  - CPU User Work
  - GPU User Work
- **CPU1**
  - Mode/Task Switch
  - CPU User Work
  - GPU User Work
- **CPU2**
  - Mode/Task Switch
  - CPU User Work
  - GPU User Work

1. **Mode/Task Switch**
2. **CPU User Work**
3. **GPU User Work**
4. **Low-Performance User-Mode Operation**
MODES OF INTERFERENCE FROM GPU SSRS

- **CPU User Work**
- **GPU User Work**
- **Mode/Task Switch**
- **Kernel Task Operation**
- **Low-Performance User-Mode Operation**

1. **CPU0**
   - 3a (Mode/Task Switch)
   - 4 (Kernel Task Operation)

2. **CPU1**
   - 2
   - 3
   - 3a
   - 4
   - 4b

3. **CPU2**
   - 1
   - 2
   - 3b
   - 4a
   - 6

4. **GPU**
   - GPU Stalls....

- **Mode/Task Switch**
  - Represents transitions between user and kernel modes
- **Kernel Task Operation**
  - Represents operations like context switching or task management
- **Low-Performance User-Mode Operation**
  - Represents operations that are less performance-critical
MODES OF INTERFERENCE FROM GPU SSRS

CPU0

CPU1

CPU2

GPU

Mode/Task Switch

Low-Performance User-Mode Operation

CPU User Work

GPU User Work

1 2 3 4 5

3a 4a 3b 4b 6 2 1 5

Mode Task Switch

Kernel Task Operation

Interference from GPU system service requests is illustrated through various modes and tasks within different CPU's. The diagram shows the impact of GPU stalls on the execution of CPU tasks, highlighting the modes of interference such as CPU user work, GPU user work, mode/task switch, and low-performance user-mode operation. The timeline and task allocation provide insights into how these interactions affect system performance.
MODES OF INTERFERENCE FROM GPU SSRS

- **CPU0**
  - Mode/Task Switch
  - Kernel Task Operation
  - Low-Performance User-Mode Operation

- **CPU1**
  - Mode/Task Switch
  - Kernel Task Operation
  - Low-Performance User-Mode Operation

- **CPU2**
  - Mode/Task Switch
  - Kernel Task Operation
  - Low-Performance User-Mode Operation

- **GPU**
  - GPU Stalls....
MODES OF INTERFERENCE FROM GPU SSRS

- CPU0
- CPU1
- CPU2

GPU Stalls....

2
3
3a
4
4b
5
MODES OF INTERFERENCE FROM GPU SSRS

- GPU Stalls...
- Direct CPU Overheads

CPU0

CPU1

CPU2

- 2
- 3
- 3a
- 4
- 4b
- 5
MODES OF INTERFERENCE FROM GPU SSRS

- Direct CPU Overheads
- Indirect CPU Overheads

CPU0

CPU1

CPU2

GPU

GPU Stalls....

2 3 3a 4 4b 5
MODES OF INTERFERENCE FROM GPU SSRS

- Direct CPU Overheads
- Indirect CPU Overheads
- GPU Overheads
TEST PLATFORM USED TO DEMONSTRATE THIS
TEST PLATFORM USED TO DEMONSTRATE THIS

• AMD A10-7850K APU
  • 4x 3.7 GHz CPU cores. AMD Family 15h Model 30h
  • 720 MHz AMD GCN 1.1 (“Sea Islands”, gfx700) GPU, 8 CUs
  • 32 GB Dual-Channel DDR3-1866
TEST PLATFORM USED TO DEMONSTRATE THIS

• AMD A10-7850K APU
  • 4x 3.7 GHz CPU cores. AMD Family 15h Model 30h
  • 720 MHz AMD GCN 1.1 (“Sea Islands”, gfx700) GPU, 8 CUs
  • 32 GB Dual-Channel DDR3-1866
• Ubuntu 14.04.3 LTS (AMD64)
  • Linux® kernel 4.0.0 with HSA Drivers (amdkfd 1.6.1)
TEST PLATFORM USED TO DEMONSTRATE THIS

- **AMD A10-7850K APU**
  - 4x 3.7 GHz CPU cores. AMD Family 15h Model 30h
  - 720 MHz AMD GCN 1.1 (“Sea Islands”, gfx700) GPU, 8 CUs
  - 32 GB Dual-Channel DDR3-1866
- **Ubuntu 14.04.3 LTS (AMD64)**
  - Linux® kernel 4.0.0 with HSA Drivers (amdkfd 1.6.1)
- **PARSEC benchmarks for “CPU work”**
TEST PLATFORM USED TO DEMONSTRATE THIS

- **AMD A10-7850K APU**
  - 4x 3.7 GHz CPU cores. AMD Family 15h Model 30h
  - 720 MHz AMD GCN 1.1 (“Sea Islands”,gfx700) GPU, 8 CUs
  - 32 GB Dual-Channel DDR3-1866
- **Ubuntu 14.04.3 LTS (AMD64)**
  - Linux® kernel 4.0.0 with HSA Drivers (amdkfd 1.6.1)
- **PARSEC benchmarks for “CPU work”**
- **OpenCL™ benchmark applications modified to create SSRs (page faults)**
  - BPT [Veselý et al., ISPASS 2016]
  - XSBench [Veselý et al., ISPASS 2016]
  - SHOC BFS
  - SHOC SpMV
  - Pannotia SpMV
  - µBenchmark
INDIRECT CPU OVERHEADS FROM GPU SSRS

**Increase of User-Level Branch Mispredict Rate**

**Increase of User-Level L1 Cache Miss Rate**
GPU PERFORMANCE WITH CONCURRENT CPU APPS

Normalized GPU application perf.
CPU cores avoid going to sleep, which can increase responsiveness to GPU requests.
Mitigation Strategies

TAKE INSPIRATION FROM OTHER DOMAINS (E.G. HIGH PERFORMANCE NETWORKING)

- Interrupt Coalescing
- Interrupt Steering
- Merged SSR Handlers
- Driver for Enforcing CPU QoS
INTERFERENCE FROM GPU SYSTEM SERVICE REQUESTS

MITIGATION: INTERRUPT COALESCING

CPU0

CPU1

CPU2

GPU

GPU Stalls....

1. Interrupt coalescing
2. CPU0
3. CPU1
4. CPU2
5. GPU

Diagram shows the process flow and components involved in interrupt coalescing to mitigate GPU stalls.
MITIGATION: INTERRUPT COALESCING

- Wait until multiple SSR requests arrive before interrupting CPU core.
**MITIGATION: INTERRUPT COALESCING**

- **GPU**
  - GPU Stalls....

- **CPU0**
  - 2
  - 3
  - Take fewer interrupts and see less indirect user-level overhead
  - But more likely to stall the GPU

- **CPU1**
  - 4a
  - 4b

- **CPU2**
  - 5

Wait until multiple SSR requests arrive before interrupting CPU core.
MITIGATION: INTERRUPT COALESCING

Take fewer interrupts and see less indirect user-level overhead

But more likely to stall the GPU

Wait until multiple SSR requests arrive before interrupting CPU core
MITIGATION: INTERRUPT STEERING

GPU

CPU0

CPU1

CPU2

GPU Stalls....

MITIGATION: INTERRUPT STEERING

GPU

CPU0

CPU1

CPU2

GPU Stalls....
Ensure that all interrupts go to the same core, rather than round-robin to all cores.
MITIGATION: INTERRUPT STEERING

- Ensure that all interrupts go to the same core, rather than round-robin to all cores.
- Limit how many cores are affected.
- Reduce global indirect overheads.

Diagram:

- GPU
- CPU0
- CPU1
- CPU2

Steps:

1. 
2. 
3. 
4a. 
4b. 
5. 
6. 

Notes:

- Ensure all interrupts go to the same core.
MITIGATION: INTERRUPT STEERING

Limit how many cores are affected.
Reduce global indirect overheads.
But harms fairness!

Ensure that all interrupts go to the same core, rather than round-robin to all cores.
MITIGATION: MERGED SSR HANDLER

Merge SSR pre-proceeding (4) with interrupt handler (3)
MITIGATION: MERGED SSR HANDLER

- Fewer interrupts and less indirect overhead.
- Handle GPU requests with much less latency.
- Merge SSR pre-proceeding (4) with interrupt handler (3)
INTERFERENCE FROM GPU SYSTEM SERVICE REQUESTS

MITIGATION: MERGED SSR HANDLER

Fewer interrupts and less indirect overhead.

Handle GPU requests with much less latency.

But runs more CPU code in high-priority context!

Merge SSR pre-proceeding (4) with interrupt handler (3)

CPU0

CPU1

CPU2

GPU Stalls...

Merge SSR pre-proceeding (4) with interrupt handler (3)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of GPU performance running w/ CPU applications

Geomean of CPU workload performance w/ all non-\(\mu\)-bench GPU apps

Higher is better

Right is better

October 5, 2018
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of GPU performance running w/ CPU applications (Higher is better)

Interrupt Steering + Merged Handler

Interrupt Steering + Interrupt Coalescing

Interrupt Steering + Merged Handler

Geomean of CPU workload performance w/ all non-\mu\text{bench} GPU apps (Right is better)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of GPU performance running w/ CPU applications (Higher is better)

Geomean of CPU workload performance w/ all non-μbench GPU apps (Right is better)

- Merged Handler
- Interrupt Steering + Merged Handler
- Interrupt Steering + Interrupt Coalescing
- Default
INTERFERENCE FROM GPU SYSTEM SERVICE REQUESTS

DRIVER MODIFICATIONS TO ENSURE CPU QOS

CPU 2 — 4b — CPU 1 — 3a — CPU 0 — 3b — GPU

Memory

5 — 4a — 1
DRIVER MODIFICATIONS TO ENSURE CPU QOS

CPU 2 → 4b → CPU 1 → 3a → CPU 0 → 3b → GPU

Memory

2 → 1
DRIVER MODIFICATIONS TO ENSURE CPU QOS

CPU 2 -> 4b -> CPU 1 -> 3a -> CPU 0 -> 3b -> GPU

Governor in Driver

Memory

5
DRIVER MODIFICATIONS TO ENSURE CPU QOS

Step 5: Track SSR overheads + delay step 6 to stall the GPU

Governor in Driver
GOVERNOR TO CONTROL CPU QOS
GOVERNOR TO CONTROL CPU QOS

CPU Cycles Handling SSRs > Threshold
GOVERNOR TO CONTROL CPU QOS

CPU Cycles Handling SSRs > Threshold

N → Delay ← 0
GOVERNOR TO CONTROL CPU QOS

CPU Cycles Handling
SSRs > Threshold

Service SSR and Return GPU Result

Delay ← 0
GOVERNOR TO CONTROL CPU QOS

Delay == 0?

Y

CPU Cycles Handling SSRs > Threshold

N

Delay ← 0

Service SSR and Return GPU Result
GOVERNOR TO CONTROL CPU QOS

Delay == 0?

Y

Delay ← 10 µsec

N

CPU Cycles Handling SSRs > Threshold

Delay ← 0

Service SSR and Return GPU Result
GOVERNOR TO CONTROL CPU QOS

CPU Cycles Handling SSRs > Threshold

- Service SSR and Return GPU Result
- Delay ← 0

- Delay ← 10 µsec
- Sleep ‘Delay’ µsec

- Delay == 0?
- Delay ← 10 µsec
- Sleep ‘Delay’ µsec

- Delay ← 0
GOVERNOR TO CONTROL CPU QOS

Delay \leftarrow \text{Delay} \ast 2

Delay \overset{\text{Y}}{=} 0?

Delay \leftarrow 10 \mu\text{sec}

Sleep ‘Delay’ \mu\text{sec}

CPU Cycles Handling SSRs > \text{Threshold}

Service SSR and Return GPU Result

Delay \leftarrow 0
CPU PERFORMANCE AT DIFFERENT QOS LEVELS

- Blackscholes
- Bodytrack
- Canneal
- Dedup
- Facesim
- Ferret
- Fluidanimate
- Freqmine
- Raytrace
- Streamcluster
- Swaptions
- Vips
- X264
- Gmean
SUMMARY

- Heterogeneous systems can include increasingly more accelerators
- GPUs and accelerators now request system services
- These can cause interference between accelerators & unrelated CPU work
- Problem may worsen in the future
- Existing mitigation strategies help, but are not complete solution
QUESTIONS?

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Linux is a registered trademark of Linus Torvalds. OpenCL is a trademark of Apple Inc. used by permission by Khronos. Other names used herein are for identification purposes only and may be trademarks of their respective companies.
BACKUP SLIDES
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of CPU workload performance running with \( \mu \text{bench} \) (Right is better)

Geomean of \( \mu \text{bench} \) performance running w/ CPU applications (Higher is better)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of CPU workload performance running with $\mu$bench
(Right is better)

Geomean of $\mu$bench performance running w/ CPU applications (Higher is better)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

Geomean of CPU workload performance running with $\mu$bench (Right is better)

Geomean of $\mu$bench performance running w/ CPU applications (Higher is better)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

- Merged Handler
- Merged Handler + Interrupt Coalescing
- Merged Handler + Interrupt Steering + Coalescing
- Default

Geomean of CPU workload performance running with μbench (Right is better)

Geomean of user performance running w/ CPU applications (Higher is better)
PARETO CURVE OF MITIGATION STRATEGY TRADEOFFS

- Merged Handler
- Merged Handler + Interrupt Coalescing
- Merged Handler + Interrupt Steering + Coalescing
- Interrupt Steering + Interrupt Coalescing
- Interrupt Coalescing
- Interrupt Steering
- Default

Geomean of CPU workload performance running with $\mu$bench
(Right is better)

Geomean of $\mu$bench performance running w/ CPU applications (Higher is better)
SSRS LIMIT LOW-POWER SLEEP STATES

Execution %age in Sleep State

<table>
<thead>
<tr>
<th></th>
<th>bfs</th>
<th>bpt</th>
<th>spmv</th>
<th>sssp</th>
<th>xsbench</th>
<th>μbench</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_SSR</td>
<td>100</td>
<td>40</td>
<td>70</td>
<td>60</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>gpu_SRR</td>
<td>90</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>no_SSR</td>
<td>80</td>
<td>50</td>
<td>40</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>gpu_SRR</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>no_SSR</td>
<td>60</td>
<td>70</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>gpu_SRR</td>
<td>50</td>
<td>80</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>no_SSR</td>
<td>40</td>
<td>90</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>gpu_SRR</td>
<td>30</td>
<td>80</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>no_SSR</td>
<td>20</td>
<td>70</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>gpu_SRR</td>
<td>10</td>
<td>60</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

October 5, 2018
MITIGATION EFFECT ON SLEEP STATES

- Default
- Steering
- Coalescing
- Merged
- Steering + Coalescing
- Steering + Merged
- Coalescing + Merged
- Steering + Coalescing + Merged

Execution %age in Sleep State

- no SSR
- SSR