Accelerated graphics with real-time. How do they play together? – Part 2

How accelerated graphics processors may interact with general-purpose operations

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

Topics

In theory, there are at least two functional areas where accelerated graphics processors may interact with general purpose operations:

1. Memory bus and cache bandwidth
2. Locking during execution of atomic code sections

Topics

In theory, there are at least two functional areas where accelerated graphics processors may interact with general purpose operations:

1. Memory bus and cache bandwidth
2. Locking during execution of atomic code sections

Let's have a look!

OSADL QA Farm system #1 to be examined

- *Name:* rack0slot2.osadl.org (https://www.osadl.org/?id=1532)
- *Distribution:* Fedora Linux 35 (Workstation Edition)
- *Linux kernel:* 5.15.2-rt19, OSADL add-on patches applied
- *CPU:* x86 Intel Core i7-3770K @3500 MHz (4 cores with HT)
- *Formerly named:* Sandy Bridge
- *Video resolution:* 1920 x 1080 pixels
- *Video adapter interface:* PCI bus
- *DRM subsystem:* Open source mainline Linux driver

OSADL QA Farm system #2 to be examined

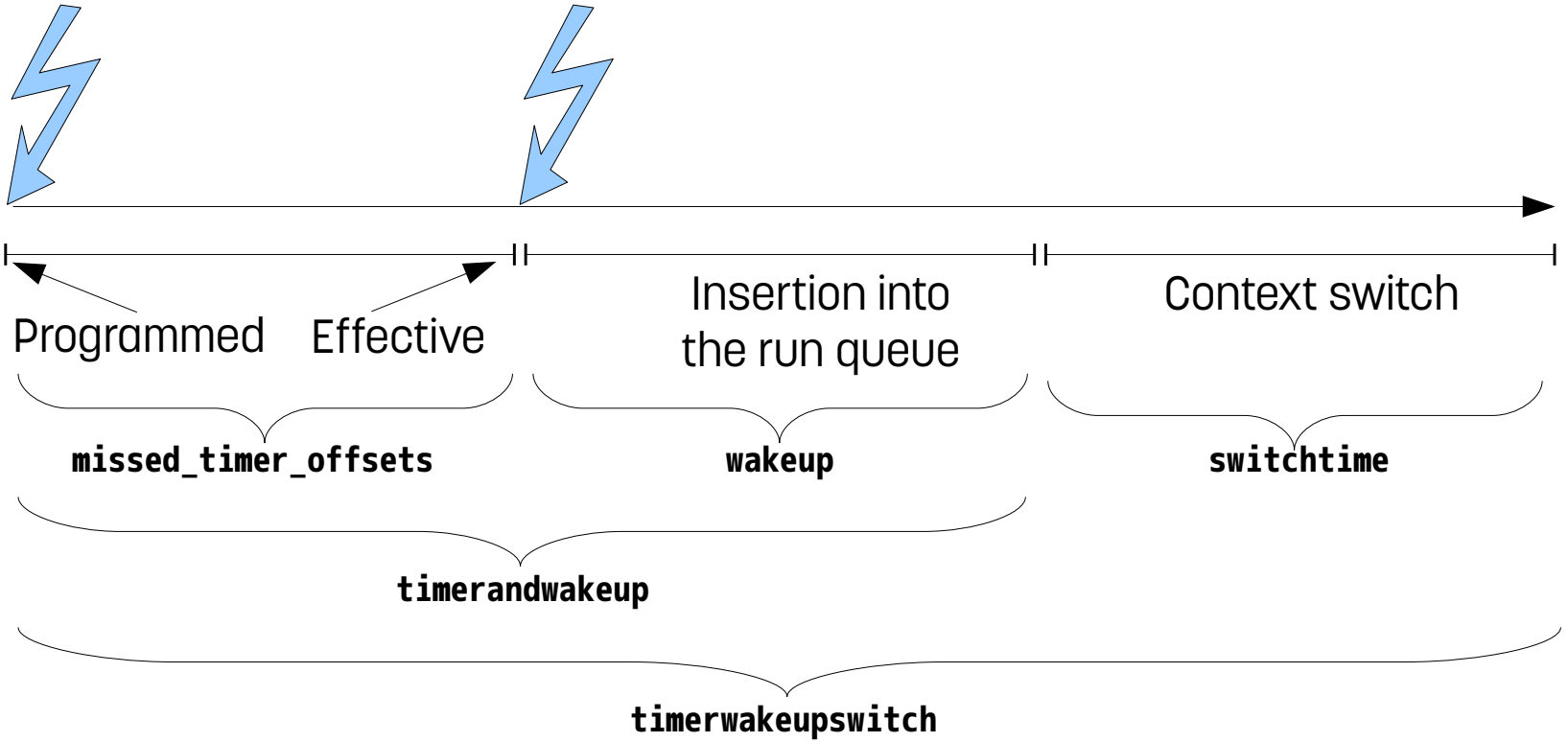
- *Name:* rack0slot3.osadl.org (https://www.osadl.org/?id=1546)
- *Distribution:* Ubuntu 20.04.3 LTS
- *Linux kernel:* 5.15.2-rt19, OSADL add-on patches applied
- *CPU:* x86 Intel Core i9-9900K CPU 3600 MHz (8 cores with HT)
- *Formerly named:* Coffee Lake
- *Video resolution:* 1920 x 1080 pixels
- *Video adapter interface:* On-chip
- *DRM subsystem:* Open source mainline Linux driver

Particular OSADL add-on patch in use

- The patch is called *latency-histograms.patch.*
- It is publicly available at https://www.osadl.org/?id=2945.
- It was originally part of the *PREEMPT_RT* patches, but was removed to not jeopardize the mainlining process.
- All RT Linux kernel versions are supported since kernel patch level 4.16.
- Configuration: `Kernel hacking ---> [*] Tracers --->`

```
[*] Missed Timer Offsets Histogram
[*] Scheduling Latency Tracer
[*]   Scheduling Latency Histogram
[*]     Context Switch Time Histogram
```

Available histograms



Enabling the histograms

- By default, histograms are disabled, thus imposing very little extra load.
- Histograms are enabled by writing non-zero to the related virtual file:

```
enabledir=/sys/kernel/debug/latency_hist/enable
for i in wakeup missed_timer_offsets timerandwakeup switchtime \
  timerwakeupswitch
do
  echo 1 >$enabledir/$i
done
```

Results are available per core and histogram

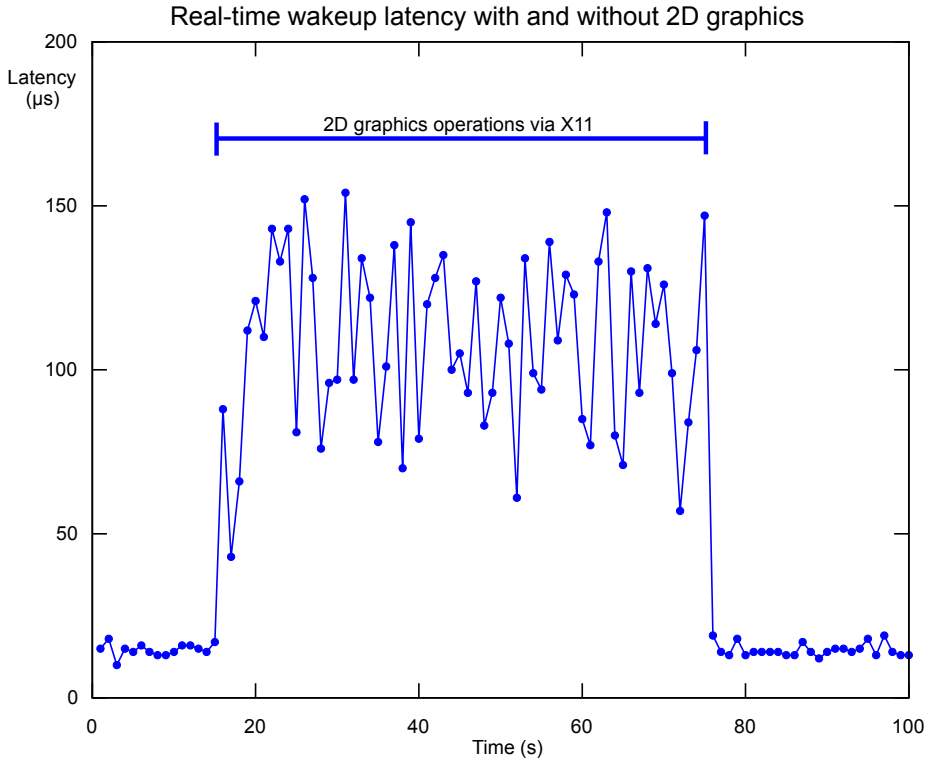
- All output of the latency histograms is available in the directory `/sys/kernel/debug/latency_hist`.
- The entire preemption latency histogram is available in the virtual file `timerwakeupswitch/CPU$core`.
- The maximum latency since most recent reset along with switch data is available in the virtual file `timerwakeupswitch/max_latency-CPU$core`.
- Format of the maximum latency file is:

```
1234 99 37 (0,13) cyclictest <- 0 -21 swapper/0 1234.123456 sleep
 PID  Prio  Latency    Command     PID Prio Command  Timestamp  Syscall
```

Graphics & RT: Reproduce the interference

- Set *cpufreq* governor to *performance*.
- Run *cyclictest* with usual OSADL parameters, but only on core `$core`:
  `cyclictest -m -n -t1 -a$core -p99 -i200`
- Determine maximum preemption latency every minute from file:
  `/sys/kernel/debug/latency_hist/timerwakeupswitch/max_latency-CPU$core`
- Apply graphics stress with parallel execution of 2D graphics:
  `taskset -c $core x11perf -sync -rect500`
- Set affinity of *Xorg* and graphics interrupt to same core.
- Plot latency over time with and without graphics stress.

Graphics & RT: Result of the interference



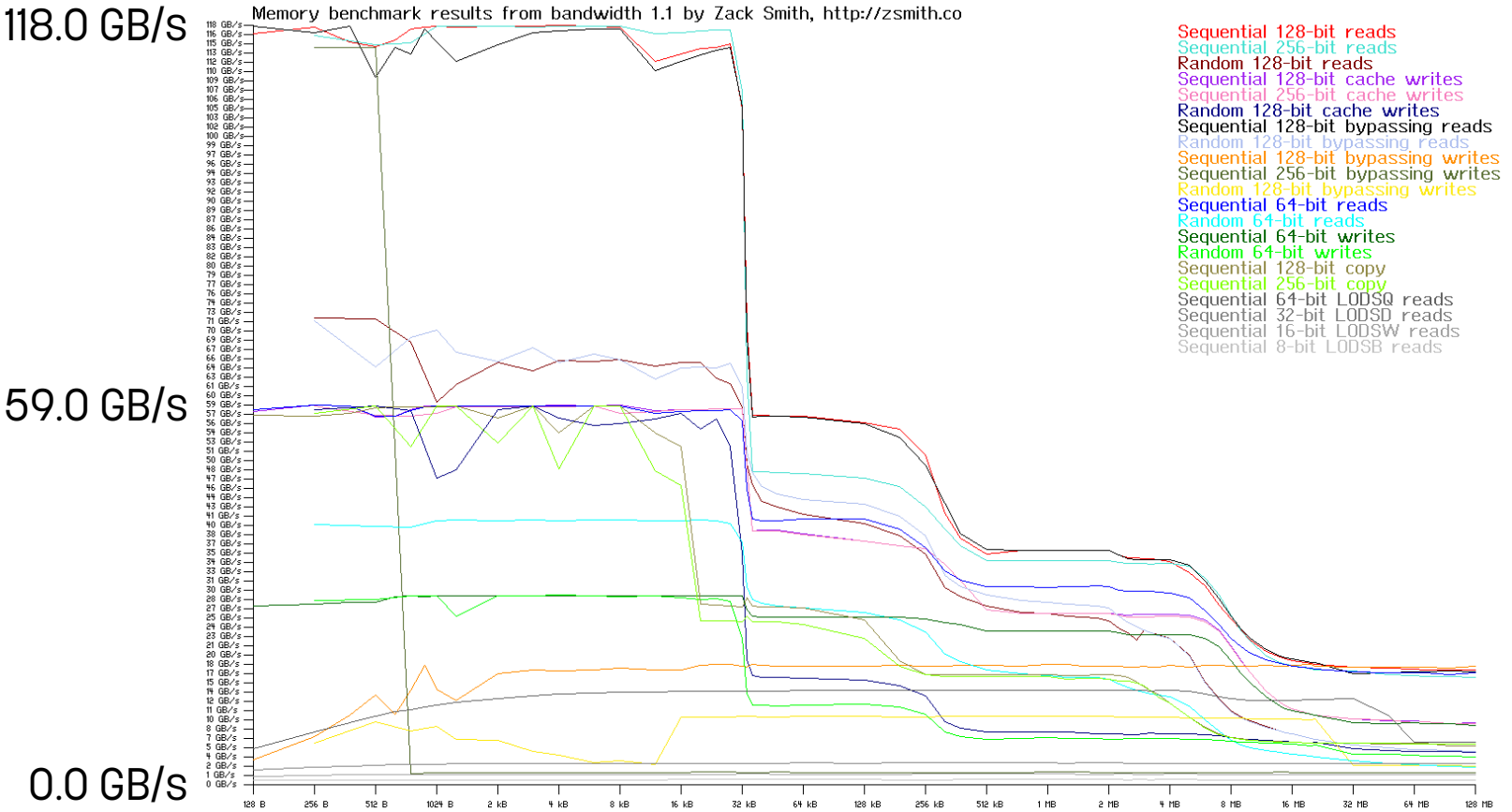
System #1

Graphics & RT: Reproduce the interference

By the way:
We call this "level #1 latency fighting"

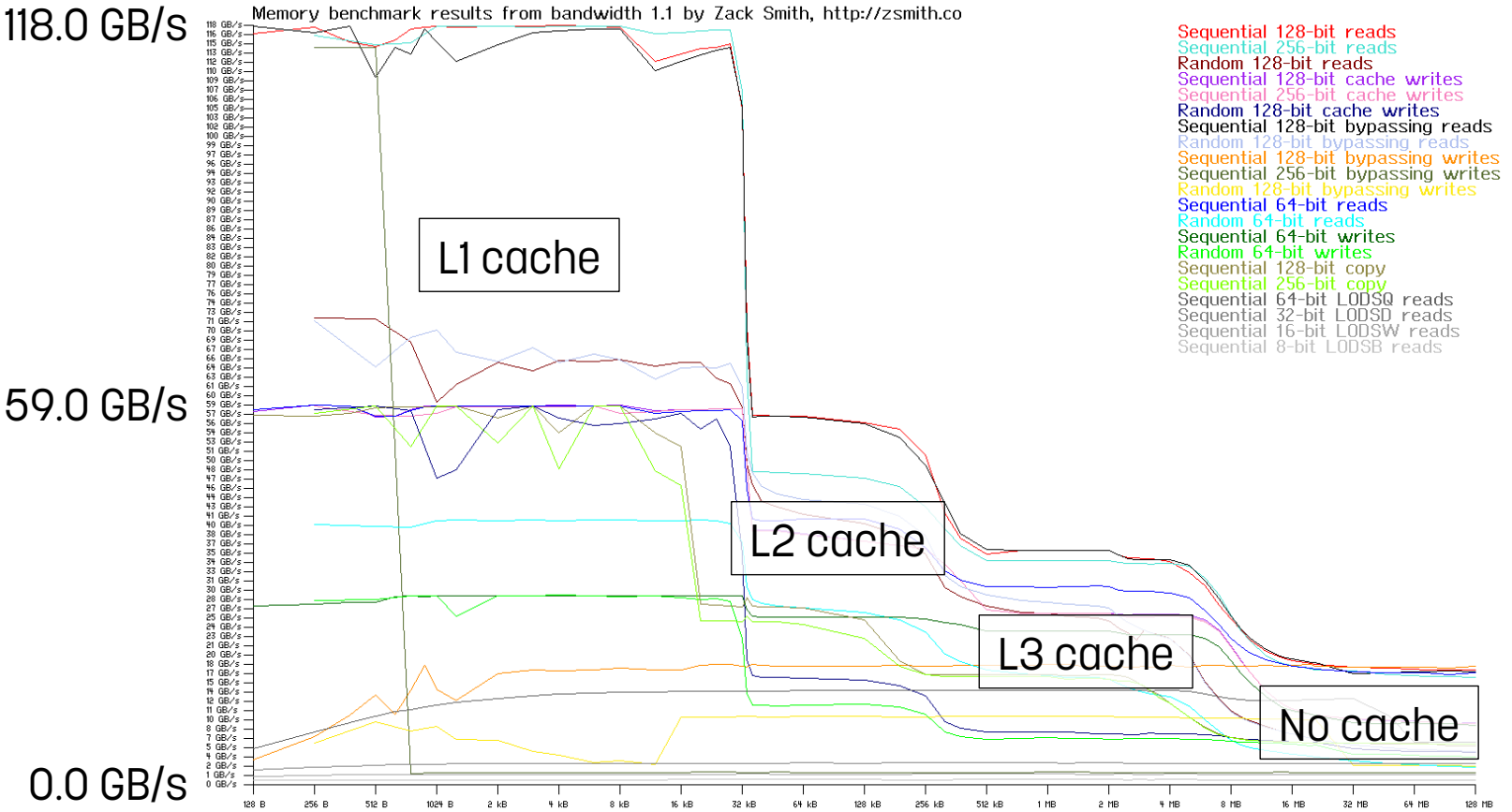
- Set *cpufreq* governor to *performance*.
- Run cyclictest with OSADL parameters, but only on core `$core`:
  `cyclictest -m -n -t1 -a$core -p99 -i200`
- Determine maximum preemption latency every minute from file:
  `/sys/kernel/debug/latency_hist/timerwakeupswitch/max_latency-CPU$core`
- Apply graphics stress with parallel execution of 2D graphics:
  `taskset -c $core x11perf -sync -rect500`
- Set affinity of *Xorg* and graphics interrupt to same core.
- Plot latency over time with and without graphics stress.

1. Memory bus and cache bandwidth



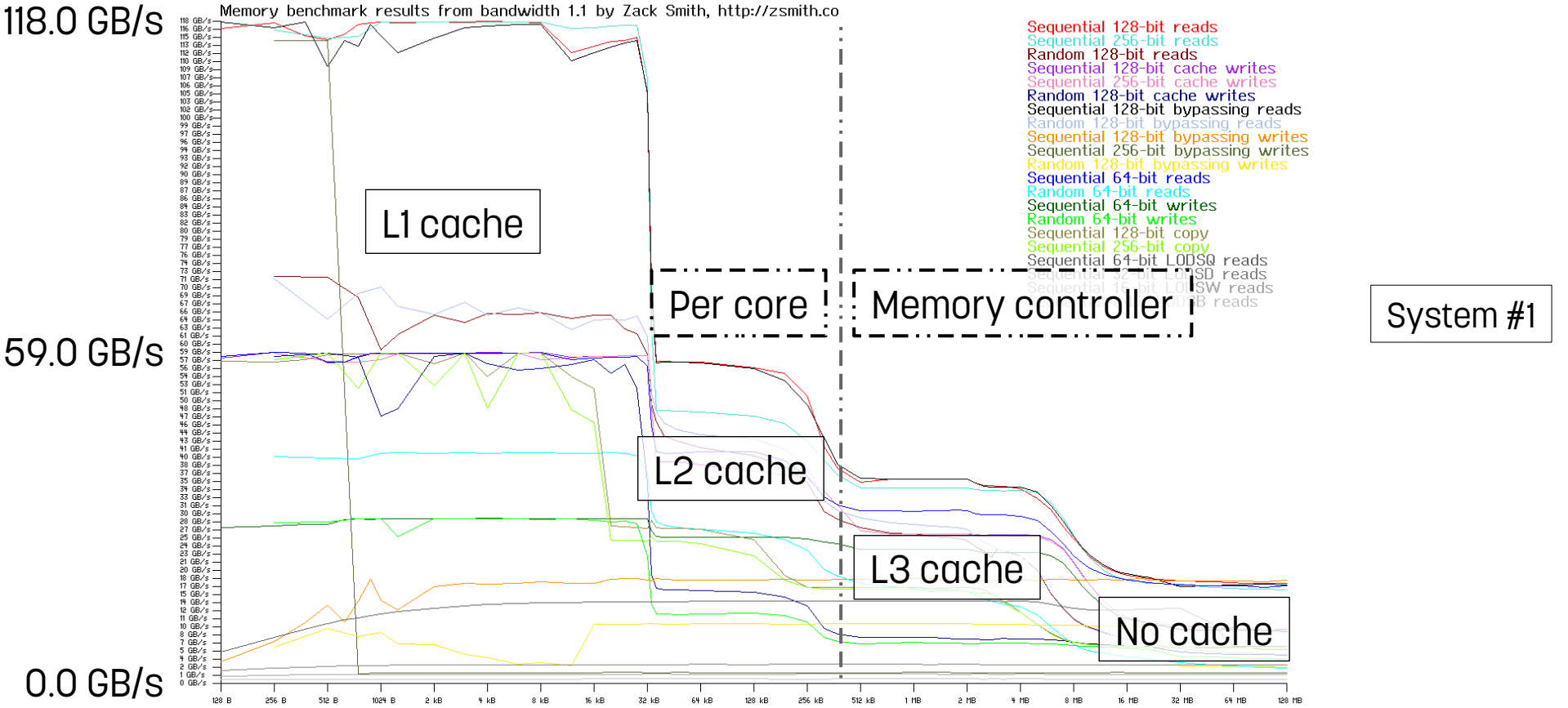
System #1

1. Memory bus and cache bandwidth



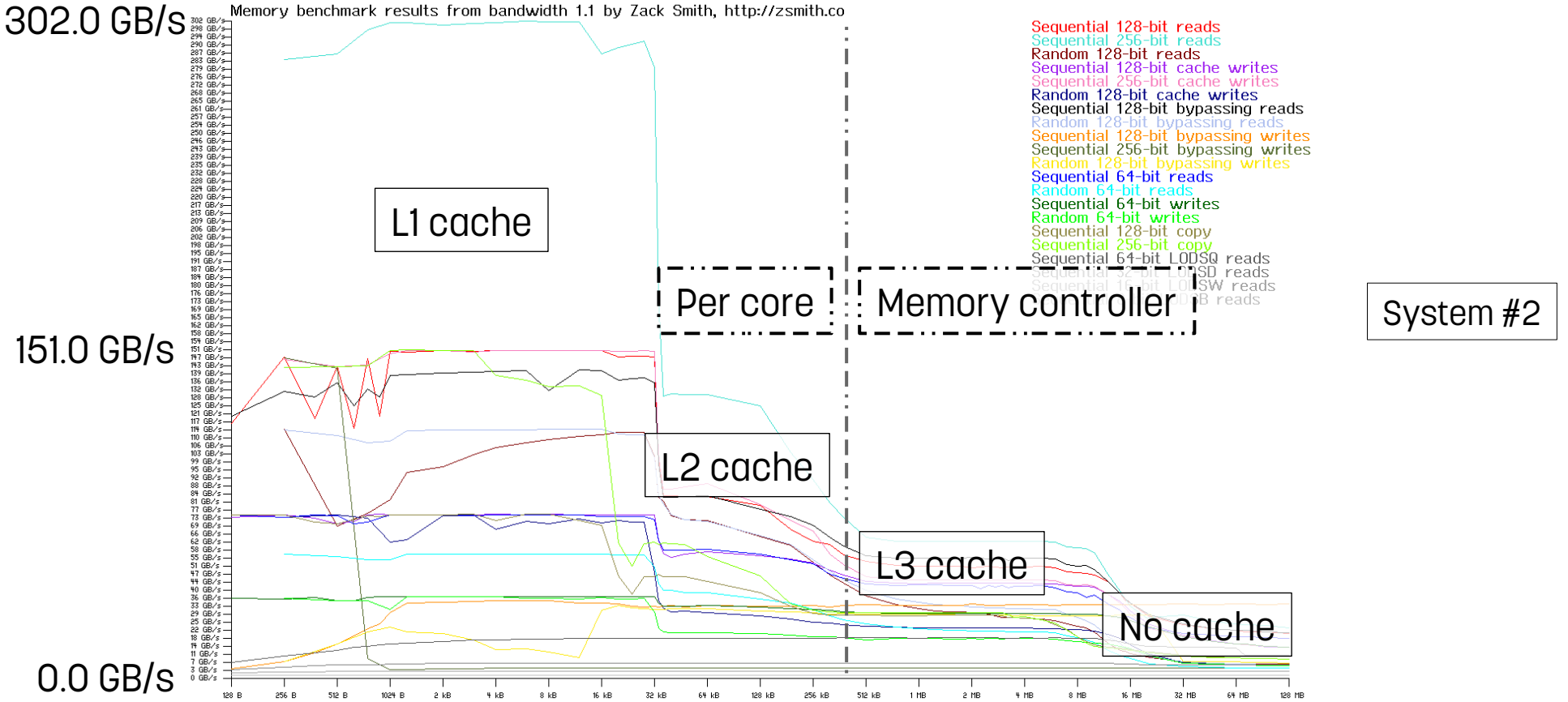
System #1

1. Memory bus and cache bandwidth



System #1

1. Memory bus and cache bandwidth



Test the effect of memory load

Run *cyclictest* with OSADL standard parameters, but only on one core:

```
cyclictest -m -n -t1 -a$core -p99 -i200
```

While *cyclictest* is running:

1. Check latency
   Apply continuous memory load to the selected core `$core` with random data that fit into L3 cache and above L3.
2. Check graphics performance
   Using the same test as before, but additionally record performance:
   ```
   taskset -c $core x11perf -sync -rect500
   ```
3. Do both

Results of testing the effect of memory load

System #1

	Memory bandwidth*	Preemption latency**	Graphics performance*
No memory load	-	12 µs	-
No memory load	-	127 µs	7.530 rects/s
L3 memory load	11.622 MB/s	15 µs	-
L3 memory load	11.389 MB/s	131 µs	7.520 rects/s
Above L3 memory load	8.902 MB/s	19 µs	-
Above L3 memory load	8.896 MB/s	133 µs	7.520 rects/s

*Average during 5-min measurement interval **Maximum during 5-min measurement interval

Results of testing the effect of memory load

System #2

	Memory bandwidth*	Preemption latency**	Graphics performance*
No memory load	-	24 µs	-
No memory load	-	58 µs	21.100 rects/s
L3 memory load	23.778 MB/s	23 µs	-
L3 memory load	20.033 MB/s	61 µs	21.200 rects/s
Above L3 memory load	12.684 MB/s	25 µs	-
Above L3 memory load	11.442 MB/s	57 µs	21.000 rects/s

*Average during 5-min measurement interval **Maximum during 5-min measurement interval

Topics

In theory, there are at least two functional areas where accelerated graphics processors may interact with general purpose operations:

1. Memory bus and cache bandwidth

2. Locking during execution of atomic code sections

Tracing the interference of graphics with RT

- Run *cyclictest* on a selected core `$core` with the break trace function enabled to stop tracing when the latency exceeds 100 µs and enable function tracing (`-fb100`):
  ```
  cyclictest -m -n -t1 -a$core -p99 -i200 -fb100
  ```
- Apply graphics stress with parallel execution of 2D graphics:
  ```
  taskset -c $core x11perf -sync -rect500
  ```
- When *cyclictest* stopped search the trace file of core `$core` for the most recent but one execution of *cyclictest.*
- Add 200 µs to the time stamp at the end of *cyclictest's* execution.
- Find out why the system failed to switch to *cyclictest* at this time.

Tracing the interference of graphics with RT

- Run *cyclictest* on a selected core `$core` with the break trace function enabled to stop tracing when the latency exceeds 100 µs and enable function tracing (`-fb100`):

  ```
  cyclictest -m -n -t1 -a$core -p99 -i200 -fb100
  ```

- Apply graphics stress with parallel execution of 2D graphics:

  ```
  taskset -c $core x11perf -sync -rect500
  ```

- When *cyclictest* stopped search the trace file of core `$core` for the most recent but one execution of *cyclictest.*
- Add 200 µs to the time stamp at the end of *cyclictest's* execution.
- Find out why the system failed to switch to *cyclictest* at this time.

Tracing the interference of graphics with RT

By the way:
We call this "level #2 latency fighting"

- Run *cyclictest* on the selected core `$core` with the break trace function enabled to stop tracing when the latency exceeds 100 µs and enable function tracing (-f 100):
  `cyclictest -m -n -t1 -a$core -p99 -i200 -fb100`
- Apply graphics stress with parallel execution of 2D graphics:
  `taskset -c $core x11perf -sync -rect500`
- When *cyclictest* stopped search the trace file of core `$core` for the most recent but one execution of *cyclictest.*
- Add 200 µs to the time stamp of the end of *cyclictest's* execution.
- Find out why the system failed to switch to *cyclictest* at this time.

Examination of the trace file

- Search for the most recent task switch when *cyclictest* gives up its time slice

```
cyclicte-390094    5d...2.. 337980218us : sched_switch: prev_comm=cyclictest prev_pid=390094 ...
cyclicte-390094    5d...2.. 337980218us : _raw_spin_lock_irqsave <-__schedule
cyclicte-390094    5d...3.. 337980218us : do_raw_spin_lock <-_raw_spin_lock_irqsave
cyclicte-390094    5d...3.. 337980219us : _raw_spin_unlock_irqrestore <-__schedule
cyclicte-390094    5d...3.. 337980219us : do_raw_spin_unlock <-_raw_spin_unlock_irqrestore
cyclicte-390094    5d...2.. 337980219us : enter_lazy_tlb <-__schedule
cyclicte-390094    5d...2.. 337980219us : __switch_to <-__schedule
cyclicte-390094    5d...2.. 337980219us : save_fpregs_to_fpstate <-__switch_to
  <idle>-0         5d...2.. 337980219us : finish_task_switch.isra.0 <-__schedule
```

System #1

Examination of the trace file

- Search for the most recent task switch when *cyclictest* gives up its time slice

```
cyclicte-390094    5d...2.. 337980218us : sched_switch: prev_comm=cyclictest prev_pid=390094 ...
cyclicte-390094    5d...2.. 337980218us : _raw_spin_lock_irqsave <-__schedule
cyclicte-390094    5d...3.. 337980218us : do_raw_spin_lock <-_raw_spin_lock_irqsave
cyclicte-390094    5d...3.. 337980219us : _raw_spin_unlock_irqrestore <-__schedule
cyclicte-390094    5d...3.. 337980219us : do_raw_spin_unlock <-_raw_spin_unlock_irqrestore
cyclicte-390094    5d...2.. 337980219us : enter_lazy_tlb <-__schedule
cyclicte-390094    5d...2.. 337980219us : __switch_to <-__schedule
cyclicte-390094    5d...2.. 337980219us : save_fpregs_to_fpstate <-__switch_to
  <idle>-0         5d...2.. 337980219us : finish_task_switch.isra.0 <-__schedule
```

System #1

- Calculate the time stamp of the expected task switch to *cyclictest*

```
  337980219us
+      200us
  337980419us
```

Examination of the trace file

- Check the function trace for the task switch to *cyclictest* at time stamp `337980419us`

```
  <idle>-0       5d...2.. 337980375us : sched_switch: next_comm=irq/45-nvkm
  ...
  <idle>-0       5d...2.. 337980376us : __switch_to <-__schedule
irq/45-n-489     5d...2.. 337980376us : finish_task_switch.isra.0 <-__schedule
  ...
irq/45-n-489     5.....12 337980379us : nvkm_pci_intr <-irq_forced_thread_fn
irq/45-n-489     5.....12 337980379us : nvkm_mc_intr_unarm <-nvkm_pci_intr
irq/45-n-489     5.....12 337980380us : gf100_mc_intr_unarm <-nvkm_pci_intr
irq/45-n-489     5d....12 337980506us : irq_enter_rcu <-sysvec_apic_timer_interrupt
```

System #1

- Calculate the time stamp of the expected Task switch to *cyclictest*

```
  337980219us
+      200us
  337980419us
```

Examination of the trace file

- Lookup the function trace around the time stamp `337980419us`

```
irq/45-n-489       5d...212 337980530us : sched_switch: prev_comm=irq/45-nvkm next_comm=cyclictest
irq/45-n-489       5d...212 337980530us : switch_mm_irqs_off <-__schedule
irq/45-n-489       5d...212 337980531us : __switch_to <-__schedule
cyclicte-390094    5d...2.. 337980531us : finish_task_switch.isra.0 <-__schedule
cyclicte-390094    5d...2.. 337980531us : raw_spin_rq_unlock <-finish_task_switch.isra.0
```

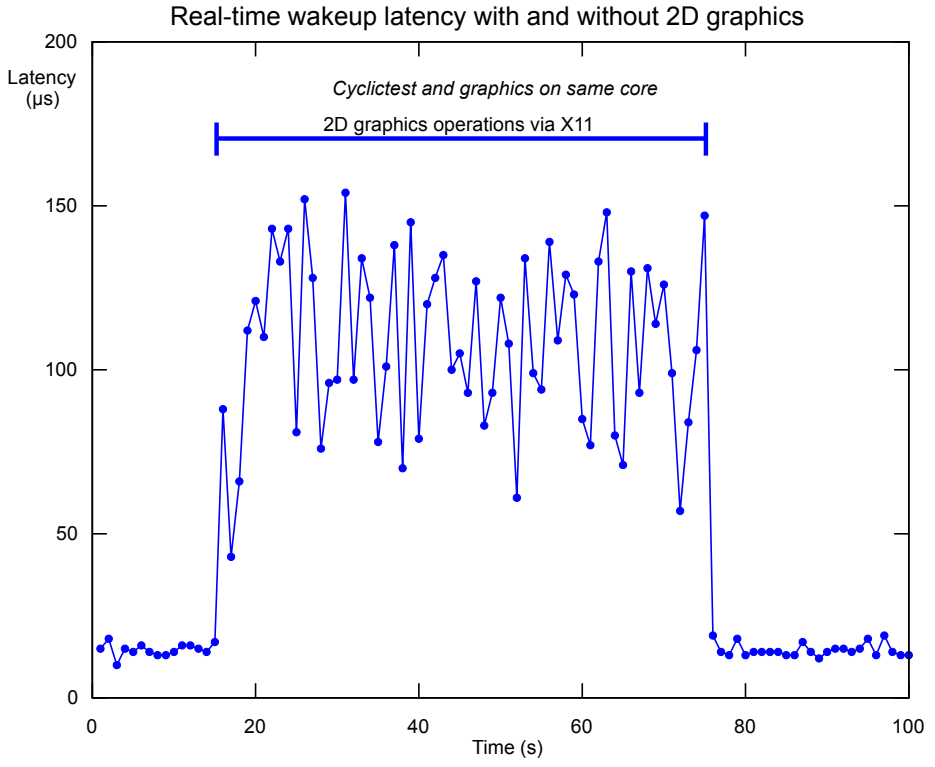
- Verify the delay

System #1

```
 337980531us
-337980219us
       312us
-      200us
       112us
```

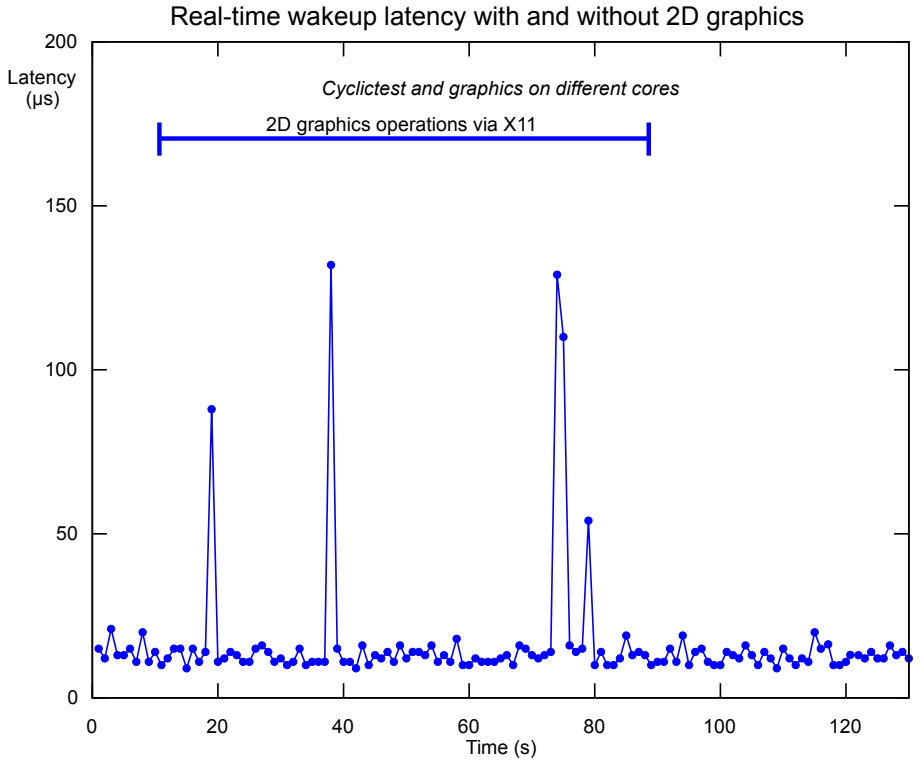
- The final task switch to *cyclictest* occurred 112 µs later than expected thus exceeding the break trace threshold of 100 µs.

RT and graphics running on same core



System #1

RT and graphics running on different cores



System #1

Conclusion (1)

For the time being and at least when running on Intel processors, accelerated graphics interferes with real-time capabilities.

- Is it caused by a limited L3 cache or memory bandwidth?
  - Based on the observations described here, the answer is **No**.

Conclusion (1)

For the time being and at least when running on Intel processors, accelerated graphics interferes with real-time capabilities.

- Is it caused by a limited L3 cache or memory bandwidth?
  - Based on the observations described here, the answer is **No**.
- Is it caused by disabling IRQs or preemption too long in the graphics driver?
  - Based on the observations described here, the answer is **Yes**.

Conclusion (1)

For the time being and at least when running on Intel processors, accelerated graphics interferes with real-time capabilities.

- Is it caused by a limited L3 cache or memory bandwidth?
  - Based on the observations described here, the answer is **No**.
- Is it caused by disabling IRQs or preemption too long in the graphics driver?
  - Based on the observations described here, the answer is **Yes**.

(This is good news, since it means that it can, in principle, be fixed in software.)

Conclusion (2)

Is there a workaround to cope with the interference between accelerated real-time capabilities by setting a defined core affinity to the various tasks such as graphics server, graphics interrupts, graphics clients and real-time control applications?

- Based on the observations described here, the answer is **Maybe**.

Conclusion (3)

Instead of starting to tinker again and inventing some short-term emergency solutions, we should realize that even after the inclusion of the whole PREEMPT_RT patch into the mainline Linux kernel a lot of work is waiting for us.

Conclusion (3)

Instead of starting to tinker again and inventing some short-term emergency solutions, we should realize that even after the inclusion of the whole PREEMPT_RT patch into the mainline Linux kernel a lot of work is waiting for us.

In consequence, we should also be prepared to continue to raise the necessary funding to get this work done.

Conclusion (3)

Instead of starting to tinker again and inventing some short-term emergency solutions, we should realize that even after the inclusion of the whole PREEMPT_RT patch into the mainline Linux kernel a lot of work is waiting for us.

In consequence, we should also be prepared to continue to raise the necessary funding to get this work done.

And, by the way, the interference of accelerated graphics with real-time capabilities is only one of the many topics that merits consideration after the RT full mainline merge will have happened.