Why are GPUs (not) fast

Lucas Stach – l.stach@pengutronix.de

graphics stack developer @ Pengutronix
Magic?

Graphics Processing Units are magical go fast devices, right?

Well, no...
Deep down

DRAM Capacity, Bandwidth & Latency

Example:

- DDR4 at 4000 MT/s
- 32 Bit Bus
- 20 ns access time
- 16 GB/s

bandwidth*delay = 320 Bytes

Deep down (the memory lane)

How to avoid the memory bus looking like this?

Caches
Filling a 2D grid of pixels

- Inherently parallel problem
- Latency matters only at the grid level

Spend HW resources on more, but less sophisticated execution engines
Throughput over latency

SIMT – single instruction multiple threads
- Multiple threads share one execution engine
  - Shared register file

If threads use more registers, lower number of threads can be in flight
- Less opportunities to hide memory latency
GPU hardware

- Optimized for (ridiculously) parallel workloads
- Memory latency hiding
- Breaks down if problem isn’t parallelizable or individual strands are too complex
GPU drivers

Split between kernel and user mode

-----------------------------  OpenGL, Vulkan, OpenCL, etc  -----------------------------

User mode driver
MESA library

-----------------------------  UAPI  -----------------------------

kernel mode driver
GPU drivers

- (Relatively) expensive submissions to hardware
- User mode driver amortizes cost via batching
- Reducing execution latency by forcing job submission (glFlush, vkQueueSubmit) increases cost (driver overhead)
GPU drivers optimize for throughput by allowing the CPU to get ahead of the GPU (pipelining)
GPU drivers

Synchronous waits for results (job finish, pixel data readback, etc) will create a pipeline bubble

Whenever possible extend pipelining into application by using asynchronous interfaces
Updates of shared data can introduce pipeline bubble

Example: change texture data used by consecutive GPU jobs
Display composition

How to get pictures on the screen

Compositor

Application

EGL, Vulkan WSI

OpenGL, Vulkan

KMS

User mode driver

UAPI

Display KMD

GPU KMD
Display pipelining

CPU

frame n+2  frame n+3

frame n+1  frame n+2

GPU

frame n+1  n+1  frame n+2  n+2

Display

frame n  frame n+1
Pipelining keeps hardware busy

Sacrifices latency (a lot) to gain throughput
Display latency reduction

CPU

Frame n+1

Frame n+2

GPU

Frame n+1

Frame n+2

Display

Frame n

Frame n+1
Display latency reduction (failed)

**CPU**
- Frame $n+1$
- Frame $n+2$

**GPU**
- Frame $n+1$
- Frame $n+2$

**Display**
- Frame $n$
- Frame $n$
- Frame $n+1$
GPU driver

- Tuned to optimize throughput

- Latency reduction is possible to some degree

- Low latency at good hardware utilization rates is (really) hard
Bonus: fences

- Fences keep track of committed work
- Eventual completion guarantees
Bonus: fences

CPU

GPU

job 1

job 2

Job 3 wait result job 3

busy job 1 job 2 job 3