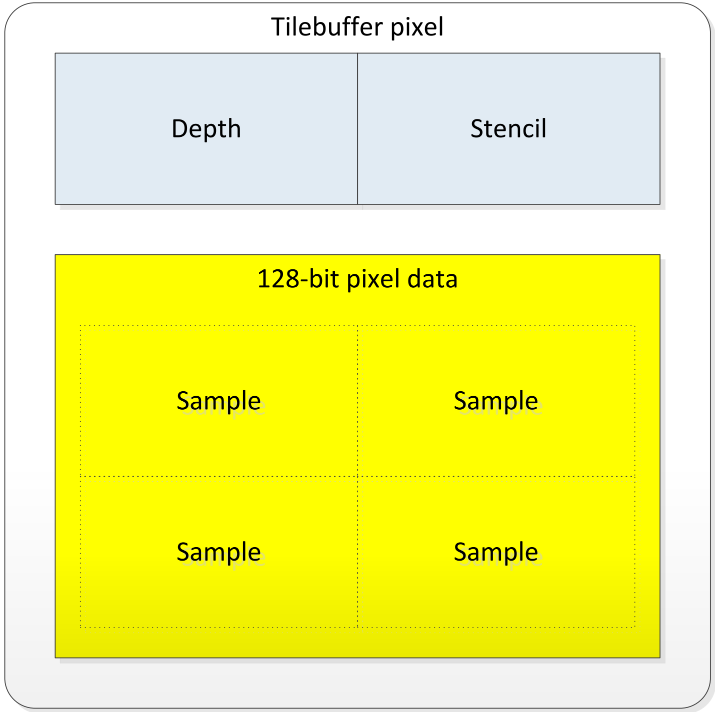
Bandwidth-efficient graphics

High Performance Graphics 2014

Marius Bjørge

Background

- ARM® Mali™-T600 GPU
  - Tile-based rendering
- 16x16 tile size
  - Fast on-chip memory
  - 128-bits per-pixel color data
  - Raw bit access
- Exposing the tile



Color Buffer Access

- Read existing color from thread's pixel location
- Multi-sampling support
  - Approximate but fast reading of MSAA framebuffer
  - Adds support for explicit per-sample shading
- Useful for things like programmable blending
- Exposed as ARM_shader_framebuffer_fetch

ARM

Depth / Stencil Buffer Access

- Read existing depth and stencil from thread's pixel location
- Useful for
  - Soft particles
  - Modulated shadows
  - Position reconstruction
  - Programmable depth/stencil testing
  - Re-interpret depth to color: variance shadowmap moments on-chip
- Exposed as ARM_shader_framebuffer_fetch_depth_stencil

Pixel Local Storage (PLS)

- Exposed as EXT_shader_pixel_local_storage
- Per-pixel scratch memory available to fragment shaders
  - Automatically discarded once a tile is fully processed
  - No impact on external memory bandwidth
- Shader declares a view of PLS memory
  - Re-interpret PLS between different passes
  - Can have separate input and output views
  - Independent of framebuffer format

Pixel Local Storage (PLS)

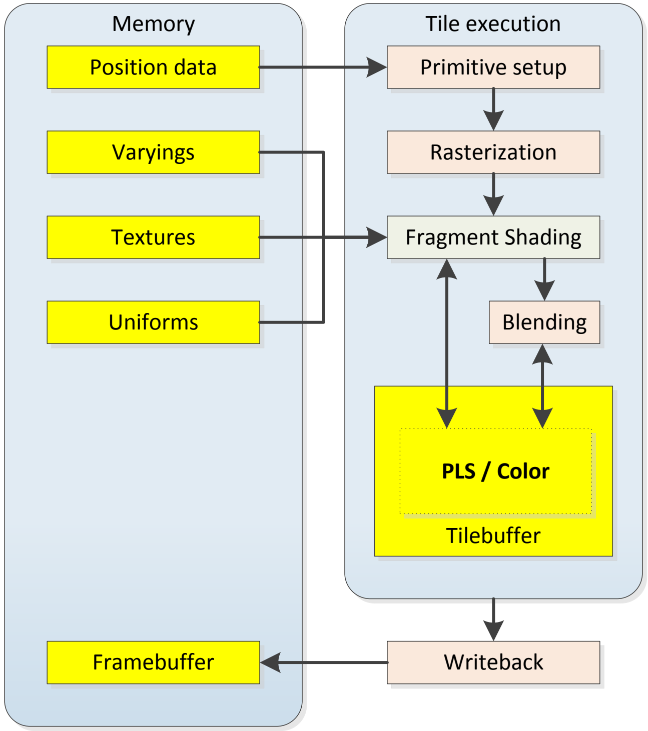
- An example:

```
__pixel_localEXT FragDataLocal
{
  layout(r32f) highp float_value;
  layout(r11f_g11f_b10f) mediump vec3 normal;
  layout(rgb10_a2) highp vec4 color;
  layout(rgba8ui) mediump uvec4 flags;
} pls;
```

- See the extension spec for more information!
  - http://www.khronos.org/registry/gles/

Pixel Local Storage (PLS)

- Rendering pipeline changes slightly when PLS is enabled
  - Writing to PLS bypasses blending
- Note
  - Fragment order
  - PLS and color share the same memory location



Pixel Local Storage (PLS)

- Limitations of the current implementation
  - No MRT or MSAA support
  - Size limitation
- Design trade-offs
  - Access synchronization
  - Format conversion

Pixel Local Storage (PLS)
Use-cases

- Deferred shading
- OIT
- Volume rendering
  - Multiple entry and exit points
- Debugging purposes
  - Per-pixel RDTSC deltas

Why Pixel Local Storage?

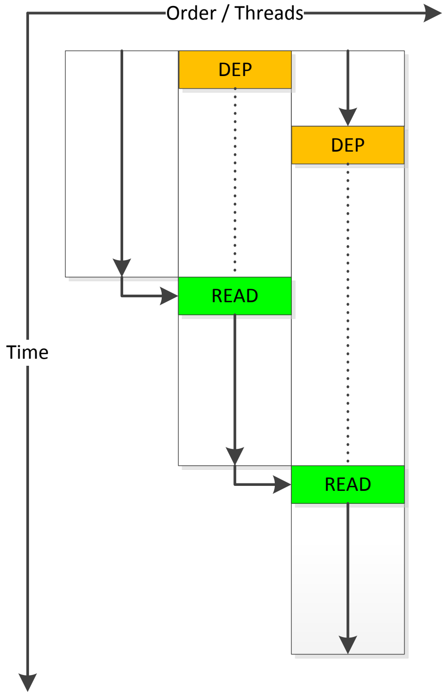
- An alternative approach is to use MRT with framebuffer fetch
  - …if the driver can prove that render targets are not used later, it can avoid the write-back
- PLS is more explicit than MRT
  - Harder for the application to get it wrong
  - Driver doesn't have to make guesses
- PLS is more flexible
  - Re-interpret PLS data between fragment shader invocations
  - Better format support (than core GLES3.x)

Instruction scheduling

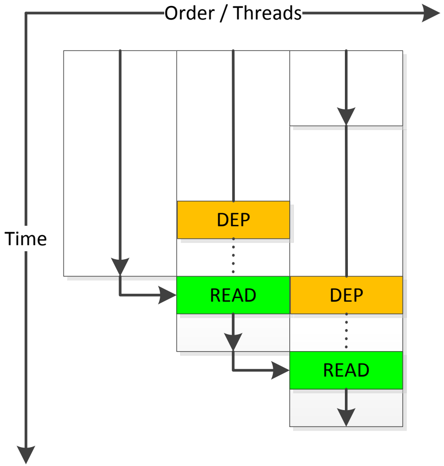
- Instruction scheduling can have huge impact on performance
- Per-pixel dependency checks
  - Think "per pixel write barrier"
  - Similar to Intel's PixelSync
- Best practice is to schedule the read as late as possible during shader execution

Instruction scheduling

Bad



Good



Future Work

- Applications and generalizations
  - "Efficient Rendering With Tile Local Storage" at SIGGRAPH 2014
- Implementation techniques
  - Add support for MSAA and MRT when used together with PLS
  - More flexible size of PLS
  - ARM® Mali™-T760 GPU

Thank you!

Marius.Bjorge@arm.com