Getting started on mobile and
Best practices for Arm GPUs

Pete Harris
Distinguished Engineer, Arm
Agenda

- Mobile essentials
- Arm GPU essentials
- Arm GPU Vulkan best practice
- What’s new?
About Me

- Arm since 2004

- Mali GPUs since 2009
  - Driver performance
  - Hardware performance
  - Developer outreach

- Product owner for Arm Mobile Studio
  - Android profiling tools for Arm CPUs and GPUs

- Maintainer for Arm ASTC Encoder
Mobile essentials
Mobile systems

- System on a chip
  - Single logic die
  - Stacked DRAM
  - Stacked flash

- Passive cooling
  - 3-6 Watts stable power

- System thinking needed
  - Thermally constrained
  - Cannot run everything at max
"big" CPUs
Best performance
…but less efficient

"LITTLE" CPUs
Most efficient
…but slower

DRAM accesses are expensive
~80mW/GB/s

DPU may not handle everything
⚠️ Can fall back to GPU
Frequency matters ...

```
Frequency

| Power consumption |

- Underclock
- Overclock
```

© 2023 Arm
Bandwidth matters ...

Energy cost of an operation vs Time between 2013 and Now.

- DRAM line shows a decrease over time.
- Logic line shows a steeper decrease, indicating a more significant improvement in energy cost compared to DRAM.
Arm GPU essentials
Immediate mode rendering

Coherent

Geometry Processing

FIFO

On-chip

Fragment Processing

Incoherent

DRAM
Tile-based rendering
Tile-based rendering

- Coherent Geometry Processing
- Off-chip
- Coherent Fragment Processing
- Tile RAM
Tile-based pros

**TBR Pros**

- Zero bandwidth ZS testing
- Zero bandwidth blending
- Zero bandwidth transient attachments
- Zero bandwidth MSAA resolve
- Spatial locality enables HSR
- Skip writes for unmodified tiles
- Zero bandwidth in-tile shading
- Zero storage transient attachments
Tile-based cons

- Stricter API rules for parallelism
- Tessellation shading is bad
- Geometry shading is bad
- Post-transform memory bandwidth
- Post-transform memory storage
Vulkan for mobile

- No driver magic
- Sensible threading model
- Sensible memory model
- (Mostly) sensible state model
- Lower driver CPU cost
- Driver is doing less
- Driver version fragmentation
- API version fragmentation
Efficient Tile memory

Vulkan API Best Practice
Basic tile memory usage

- Vulkan render passes define tile memory cycles

  - loadOp = tile memory initialize
    - Use LOAD_OP_DONT_CARE
    - Use LOAD_OP_CLEAR

  - storeOp = tile memory finalize
    - Use STORE_OP_DONT_CARE if transient (also transient allocation)
    - Use STORE_OP_NONE if read-only (or read-only attachment)

- Resolve attachments = multi-sample resolve on store
  - NEVER STORE_OP_STORE multi-sample attachments back to memory
Basic tile memory usage

- Avoid creating unnecessary render passes

- Append to existing pass whenever possible
AFBC Framebuffer compression
Arm FrameBuffer Compression

- **Lossless compression**
  - Variable bitrate
  - ~2:1 compression ratio

- **Not supported everywhere**
  - Read on `texture()` path
  - Write on framebuffer write path

- **For Vulkan:**
  - Must use `VK_IMAGE_TILING_OPTIMAL`
  - Must NOT use `VK_IMAGE_USAGE_STORAGE_BIT`
  - Must NOT use `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`
  - Must NOT use `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT`
  - Must NOT use `VK_IMAGE_CREATE_ALIAS_BIT`
  - Must NOT use `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` (Midgard, Bifrost only)
AFRC Framebuffer compression
Arm Fixed-Rate Compression

+ **New** “visually lossless” compression
  - Lossy, fixed-bitrate
  - Support 8-bit UNORM/sRGB images
  - Same API constraints as AFBC

+ **Opt-in via** VK_EXT_image_compression_control
  - Application chooses compression ratio
  - Choices from 3:2 down to 4:1

+ **Benefits over AFBC**
  - Potential for higher compression ratio
  - Smaller memory footprint
  - Better memory locality
## AFRC Framebuffer Compression

### Bitrate options

<table>
<thead>
<tr>
<th>Format</th>
<th>Bitrates</th>
<th>Vulkan setting</th>
<th>Comp. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGBA8</td>
<td>16bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT</td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>12bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_3BPC_BIT_EXT</td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>8bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT</td>
<td>4:1</td>
</tr>
<tr>
<td>RGB8</td>
<td>16bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_5BPC_BIT_EXT</td>
<td>3:2</td>
</tr>
<tr>
<td></td>
<td>12bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT</td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>8bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT</td>
<td>3:1</td>
</tr>
<tr>
<td>RG8</td>
<td>8bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT</td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>6bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_3BPC_BIT_EXT</td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>4bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT</td>
<td>4:1</td>
</tr>
<tr>
<td>R8</td>
<td>4bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_4BPC_BIT_EXT</td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>3bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_3BPC_BIT_EXT</td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>2bpp</td>
<td>VK_IMAGE_COMPRESSION_FIXED_RATE_2BPC_BIT_EXT</td>
<td>4:1</td>
</tr>
</tbody>
</table>
AFRC Framebuffer Compression

Bitrate options

Original 32-bpp RGBA
Albedo (RGB) + Roughness (Alpha)
Arm Fixed-Rate Compression 8-bpp RGBA
Albedo (RGB) + Roughness (Alpha)
Efficient Swapchains

Vulkan API Best Practice
Play nicely with the DPU

- Display processor is outside of the GPU
  - May handle rotation, scaling, color-conversion, layer composition
  - ... or it may not ...

- Android Vulkan API capability reporting is not useful ...
  - SurfaceFlinger's GPU fallback means everything is possible
  - ... but not everything is fast!

- Advice: Always ensure swapchain is in a DPU-friendly orientation
  - Apply screen pre-rotate as a geometry transform in the application
  - Set swapchain preTransform to match the surface currentTransform
  - Recreate swapchain when the surface currentTransform changes
arm

Efficient Scheduling

Vulkan API Best Practice
Tile-based pipelining

Application
Software

Render Pass 1
Render Pass 2
Render Pass 3

Geometry
Processing

Render Pass 1
Render Pass 2
Render Pass 3

Fragment
Processing

Render Pass 1
Render Pass 2
Render Pass 3
Overlap your render passes

**Good:** Pipeline barriers that allow overlap
- e.g. `srcStage = ALL_GRAPHICS`, `dstStage = FRAGMENT`

**Bad:** Pipeline barriers that serialize the world
- e.g. `srcStage = BOTTOM_OF_PIPE`, `dstStage = TOP_OF_PIPE`
The cake timer query is a lie

render passes are disjoint

Draw scheduling is ... worse ...

Elaphsed time
Efficient Geometry

Vulkan API Best Practice
Geometry bandwidth use

**Texel**
4 bits

**Vertex**
32 bytes
Geometry bandwidth use

**Texel**
4 bits

**Effective vertex**
64 bytes
Geometry bandwidth use

**Texel**
4 bits

**Effective vertex**
≈110 bytes
Geometry layout for binning

**Goal:** Only load position data for position shading

**Bad:** Fully interleaved

**Good:** Split streams

Image: Crash on the Run, King
Your GPU ran into a problem and needs to restart ...

If you call a support person, give them this info:
Stop code: VK_ERROR_DEVICE_LOST
Beware of unexpected shading

Arm GPUs shade vertices in groups of 4 contiguous indices

We can shade indices your index buffer skipped
  • E.g. Index buffer [0, 7, 15] will shade [0, 1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15]
  • Advice: Tightly pack index ranges to minimize over-shading

We can shade indices your index buffer didn’t even contain
  • E.g. Index buffer [0, 1, 2, 3, 4, 5, 6, 7, 8] will shade [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
  • Hardware ensures that direct attribute access cannot exceed buffer bounds
    • ... but shaders with indirect loads may read OOB and fault
  • Advice: Pad attribute data to a multiple of 4 verts if using indirect loads
Beware of intermediate storage

Intermediate data geometry needs memory storage
• Mali typically supports up to 180MB per render pass
• DEVICE_LOST if intermediate allocation fails
• Advice: Minimize geometry complexity

Midgard and Bifrost GPUs:
• Allocate space for min_index to max_index
• Advice: Tightly pack index ranges to reduce footprint
• Advice: Don’t pack metadata into high index bits

Valhall GPUs:
• Allocate space only for visible vertices
• Valhall CSF GPUs use a flexible allocator that can exceed 180MB
• Advice: Tightly pack index ranges to minimize overshading
Arm GPUs
What's new?

The highlights ...
Hardware support
Mali-G710 (Valhall + CSF)

- New 4-pixel shader core architecture

- Native command streams
  - Significantly lower driver CPU overhead
  - Native secondary command buffers
  - Native indirect draws, including indirect multi-draw count
  - Native queue synchronization

- AFBC for FP16 image formats
  - Lower bandwidth HDR rendering

- Improved shader tile memory access
  - Faster merged subpass rendering
Hardware support

Immortalis-G715 (Valhall + CSF)

- Ray tracing
  - Ray query
  - Ray tracing pipeline
  - See talk from Jose-Emilio

- Variable rate shading
  - See talk from Albin

- Conservative rasterization

- Arm Fixed-Rate Compression

- Faster shader core
  - 2x FMAs
  - 4x subgroup ops
  - 2x varying interpolations

- Faster texturing
  - 2x textureCube()
  - 2x textureLod() with uniform/const LOD

- Faster memory access
  - 4x uniform load access
  - 4x stack load/store access

- Faster FP16 blending & resolve
Build the future on Arm

- Join our developer community
- Connect with like-minded developers
- Get fresh insights from Arm experts
- **Sign up:** arm.com/developerprogram

Find out more

- **Forums and blogs:** community.arm.com
- **Documentation:** developer.arm.com/graphics

© 2023 Arm
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
ありがとう
Asante
Merci
감사합니다
धन्यवाद
धन्यवाद
شكرًا
toda
tolu