Midgard GPU Architecture

October 2014
The Midgard Architecture

HARDWARE EVOLUTION
Mali GPU Roadmap

Energy Efficient GPU Roadmap

- **Mali™-T604**
  - First OpenGL® ES 3.1 and OpenCL™ Full Profile mobile GPU

- **Mali-T624**
  - 50% performance uplift
  - Full ASTC support

- **Mali-T628**
  - Extending the scalability to 8 cores

- **Mali-T760**
  - Increased SoC energy efficiency
  - Scalability to 16 cores

Cost Efficient GPU Roadmap

- **Mali-400 MP**
  - First OpenGL ES 2.0 multi-core GPU with leading area efficiency

- **Mali-450 MP**
  - Double the performance of Mali-400 MP

- **Mali-T622**
  - Enabling Full Profile Compute and OpenGL ES 3.1 in mid-range

- **Mali-T720**
  - Optimized area efficiency and decreased cost & time to market
Mali-T760 High-Level Architecture

- **Distributes tasks to shader cores**
- **Up to sixteen shader cores**
- **Configurable cache shared among all shader cores**

Mali T760

**Inter-Core Task Management**

- Shader Core

**Advanced Tiling Unit**

**Memory Management Unit**

- L2 Cache: AMBA®4 ACE-Lite™
- L2 Cache: AMBA®4 ACE-Lite™

**Efficient mapping of geometry to tiles**

**Address Translation and Protection**

**Maintains cache coherency between different processors**
The Midgard Architecture

THE GRAPHICS PIPELINE
Immediate Mode Rendering

Geometry (Triangles) → Vertex Shader → Rasterizer → Fragment Shader → Blender

Triangle data drawn immediately

High memory bandwidth for blending

Typical desktop GPU
Tiled Mode Rendering

Geometry (Triangles) → Vertex Shader → Rasterizer → Fragment Shader → Blender → Tile Buffer

- Triangle lists, one per tile
- Triangle data stored temporarily
- Blender bandwidth stays on-chip

Typical mobile GPU
Basic Dataflow

- Application
- Driver
- Job Descriptors
- Job Manager
- Input Data
- Intermediate Data
- Output Data
- Display System
- CPU
- Shared Memory
- Midgard Hardware
The Midgard Architecture

THE TRI-PIPE ARCHITECTURE
Shader Core Architecture

- Compute Thread Creator
- Rasterizer
- Triangle Setup Unit
- Tiler Data Structures
- Texturing
- Compute Data and Results
- Thread Execution – “Tri Pipe”
- Early Z
- Late Z
- Blender
- Tile Buffers
- Frame Buffer
- Z/Stencil Buffer
- Textures
Tri-pipe Architecture

- Unified shader architecture
  - Fragment and vertex shaders
  - Geometry and compute shaders
  - Very high throughput graphics

- Multiple parallel pipelines
  - Two low-latency arithmetic pipes
  - 256 simultaneous threads
  - Low-latency for computation

Mali-T624

- Multiple parallel pipelines
- Two low-latency arithmetic pipes
- 256 simultaneous threads
- Low-latency for computation
The Midgard Architecture

SCALABILITY
T720 scalability: 1-8 shader cores
T760 scalability: 1-16 shader cores
The Midgard Architecture

64-BIT SUPPORT
Midgard architecture: 64-bit embedded

- Midgard is a fully-featured 64-bit GPU architecture
  - Native 64-bit address space, 64-bit arithmetic (integer and IEEE FP)
  - Very fast, optimized 16/32-bit performance – 'right-sized computing'

- Wake-up call: 32-bit addresses are not enough
  - 32-bit address space is nearly full on current high-end ARM platforms
  - Mali-T624 uses same page tables as ARM CPU for >4GByte memory
  - Be ready, do not get left behind with a 32-bit only GPU architecture

- IEEE double precision floating-point, Full Profile OpenCL
  - Numerical stability issues break many compute/graphics algorithms
  - FP64 is the default standard for scientific algorithm development
  - FP64 is exposed in key APIs:
    - OpenGL 4.x, DirectX 11, CUDA and OpenCL 1.x
64-bit use-cases

- Pointer arithmetic for addresses larger than 32 bits => 64-bit integers
- 64-bit integers and 64-bit floats needed
- High-dynamic range image processing such as summed area tables, allowing:
  - Bokeh effect
  - Summed Area Variance Shadow maps
- Double-precision GPU computing algorithms coming from the desktop and scientific computing worlds
  - Approximate may be okay for images to be viewed by human eye
  - Accuracy is needed for other (non-image) computing
The Midgard Architecture

MEMORY BANDWIDTH
T6xx bandwidth and power save features

- Improved Texture Compression
- Transaction Elimination
- Hierarchical Tiling

Percentage of Memory Bandwidth (Relative to Mali-400MP)

Textures  Frame Buffer  Other

Mali-400MP  Mali-T624
ASTC – Texture Compression

- Texture compression format developed by ARM and adopted by Khronos
- Support for LDR + HDR + 3D in ARM Mali GPUs:
  - Mali-T62x, Mali-T760, Mali-T720
- Support for vast color formats and bits/pixel
Transaction Elimination

- Lower memory bandwidth
  - Eliminates redundant writes to the output buffer

List of CRCs calculated per tile for frame N

<table>
<thead>
<tr>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
</tr>
<tr>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduction</th>
<th>Reduction 32bpp HD@60fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active game</td>
<td>&gt;20% 95MB/s</td>
</tr>
<tr>
<td>Static frame</td>
<td>100% 475MB/s</td>
</tr>
</tbody>
</table>

Compared with CRCs calculated for frame N+1

<table>
<thead>
<tr>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
</tr>
<tr>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
<td>CRC</td>
</tr>
</tbody>
</table>

Eliminated tiles matching previous frame are highlighted by
1311 – Transaction elimination signature matches: 1142 (28% elimination)
1312 – Transaction elimination signature matches: 1148 (28% elimination)
Mali-T760 bandwidth saving summary

- Reusing highly efficient Mali-T620 bandwidth saving features
  - Advanced Scalable Texture Compression (ASTC)
  - Transaction Elimination
- …and extending them even further
  - ARM Frame Buffer Compression (AFBC)
  - Smart Composition
AFBC: Bandwidth reduction in Multimedia System

- **Increase Energy Efficiency across the SoC**
  - Designed for integration with all major multimedia IP blocks

- **Maintain Image Quality**
  - Lossless Frame buffer Compression
  - Flexible formats extensible to higher pixel-depth formats

- **Efficient Implementations**
  - Easy parallelization of encode and decode

- **Supported in current and all future generations of media processing IP from ARM**
  - Mali-T760, Mali-V500, Mali-DP-500
  - licensable for integration with 3rd Party media process IP

![Graph showing System Bandwidth Reduction for various benchmarks:](image-url)
Extending Transaction Elimination
- Significantly lower composition bandwidth
- Reduces unnecessary shader workload
- Ignoring repetitive tile data by tracking Change Maps

Integrates with the window manager
- Notifies Mali GPU which regions need updating

For partially-static applications further savings
- Applications notify which screen regions are updated
- Bandwidth and workload reduction
- Enables complete end-to-end Smart Composition

Bandwidth reduction with Smart Composition

- Chrome Browser: 90%
- Facebook application: 80%
- Twitter application: 70%

Source: ARM
The Midgard Architecture

SUMMARY
Evolution of Midgard architecture

- High Scalability
  - Multiple configurations of shader cores
  - Different product lines extracted by the same architecture

- Bandwidth reduction and high power efficiency
  - ASTC Texture Compression
  - Transaction Elimination/ Smart Composition
  - ARM Frame Buffer Compression
  - Tiler & Interconnect optimisations

- Support for all major graphics and compute APIs
  - Khronos compliant OpenGL® ES 3.1/3.0 /2.0 / 1.1
  - Khronos compliant OpenCL® 1.1/1.0
  - Microsoft Windows compliant Direct3D 11.1
The Midgard Architecture

QUESTIONS AND ANSWERS
Efficient Rendering with Tile Local Storage

Staff Engineer
Marius Bjorge
Agenda

- Motivation
- Introduction to extensions
- Example use-cases
Background

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

- Read existing color from thread's pixel location
  - No need to re-bind texture
- 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
Color Buffer Access - Example

```glsl
#extension GL_ARM_shader_framebuffer_fetch : require
precision mediump float;

uniform vec4 uBlendColor;

void main()
{
    vec4 PrevColor = gl_LastFragColorARM;
    gl_FragColor = mix(PrevColor, uBlendColor, uBlendColor.w);
}
```
Depth / Stencil Buffer Access

- Read existing depth and stencil from thread's pixel location
  - No need to re-bind texture
- Useful for
  - Soft particles
  - Modulated shadows
  - Position reconstruction
  - Programmable depth/stencil testing
  - Re-interpret depth to color: keep variance shadowmap moments on-chip
- Exposed as ARM_shader_framebuffer_fetch_depth_stencil
Depth Buffer Access - Example

```glsl
#pragma extension GL_ARM_shader_framebuffer_fetch_depthStencil : require
precision mediump float;

uniform float uDepthRef;
uniform int uStencilRef;

void main()
{
    if (gl_LastFragStencilARM != uStencilRef) discard;
    if (gl_LastFragDepthARM > uDepthRef) discard;

    gl_FragColor = vec4(1.0, 0.0, 0.0, 0.0);
}
```
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) - Example

```glsl
__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;
```
Pixel Local Storage (PLS)

- Rendering pipeline changes slightly when PLS is enabled
  - Writing to PLS bypasses blending
- Note
  - Fragment order
  - Fragment tests still applies
  - PLS and color share the same memory location
Pixel Local Storage (PLS)

- Limitations of the current implementation
  - No MRT or MSAA support
  - Size limitation
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
  - Not limited to OpenGL® ES 3.x frame buffer formats
Deferred Shading

- Popular technique on PC and console games
  - Very memory bandwidth intensive
  - Traditionally not a good fit for mobile
Order Independent Transparency

- “Unsolved” problem
- For correct result, fragments must be rendered in order, from nearest to farthest or farthest to nearest
Order Independent Transparency

- Input fragments:

  0 1 2 3
Order Independent Transparency

- Input fragments:

- Order: 0, 1, 2, 3
Order Independent Transparency

- Input fragments:
- Order: 0, 1, 2, 3
- Order: 0, 2, 3, 1
Order Independent Transparency

- Input fragments:
  - Order: 0, 1, 2, 3
  - Order: 0, 2, 3, 1
  - Order: 2, 3, 1, 0
Order Independent Transparency

- **Exact OIT**
  - Accurately computes the final color – all fragments must be sorted

- **Approximate OIT**
  - Does not store all fragments
  - Multi-Layer Alpha Blending [Salvi et al, 2014]
  - Adaptive Range

- **Pixel Local Storage can be used for both**
Chaining pipelines
Performance

- Deferred Shading (MRT): 100%
- Deferred Shading (PLS): 122%
- Deferred Shading and Adaptive Range OIT (PLS): 100%
- Deferred Shading and MLAB3 OIT (PLS): 93%
Summary

- **Future work**
  - Add support for MSAA and MRT when used together with PLS
  - More flexible size of PLS
  - ARM® Mali™-T760 GPU

- **Device availability**
  - Samsung Note 4 (Exynos version)
  - Samsung Galaxy Tab S (Exynos versions)
Thank you!

Marius.Bjorge@arm.com