ARM Mali GPU
Midgard Architecture

Mathias Palmqvist
Senior Software Engineer/ARM MPG

Lund University, Faculty of Engineering LTH
December 6th 2016

©ARM 2016
Agenda

- Midgard Architecture Overview
- Device Driver Work building
- GPU Software Interface
- GPU Hardware Jobs
- Job Manager
- Unified Shader Core
- Shader Core: Vertex Work
- Tiling Unit
- Shader Core: Fragment Work
- Shader Core: Tripipe

- Power Consumption & Bandwidth Reduction
  - Power Consumption
  - Forward Pixel Kill
  - Transaction Elimination
  - ARM Framebuffer Compression
  - Pixel local storage

- ARM Lund
  - ARM Lund GPU teams
  - GPU HW/SW development
  - Party!
  - Student Opportunities
Mathias Palmqvist

Work

- Senior Software Engineer, ARM ~5 years
  GPU driver, EGL on Android/X11, Multimedia.

- Previously at Sony Ericsson 7 years
  Android Graphics Integration.

Education

- LTH, M.S.EE (e98). Fourth year at U.C. San Diego.
  Hardware and graphics main focus.

Interests

- Game Development(often non-graphics related)
- Real-time techniques with mobile focus

Personal

- Married to California girl from UCSD since 13 years
- Two children, 5 and 2. Aiden and Avery.
Midgard GPU Architecture
Acronyms

AMBA  Advanced Microcontroller Bus Architecture
AXI   AMBA Advanced eXtensible Interface
APB   AMBA Advanced Peripheral Bus
ACE   AMBA AXI Coherency Extensions
GPU   Graphics Processing Unit
VPU   Video Processing Unit
DPU   Display Processing Unit
ISA   Instruction Set Architecture
SIMD  Single Instruction Multiple Data
Midgard Architecture Overview

- Tile-based, Unified Shading Architecture
- Latest Midgard GPU is T880
- Maximum of 16 shader cores
- Tile size 16x16 (4x4-32x32 internally)
- GLES 3.2, Vulkan 1.0, CL 1.2, DX FL11_2
- MSAA 4x, 8x, 16x
- T880 in Samsung GS7 and Huawei P9

- First product T604 in 2011q4
  - SoC: Samsung Exynos 5250
  - Products: Nexus10, Samsung Chromebook
GPU top level
Simplified Application-Driver Interaction

**App**
- Frame 0
- Frame 1
- Frame 2
- Frame 3

**Driver**
- F0
- F1
- F2
- F3

**GPU**
- F0
- F1
- F2

**DPU**
- F0
- F1
## Driver building GPU resource structures

<table>
<thead>
<tr>
<th>App</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>glTexImageX()</td>
<td>Build Texture structure</td>
</tr>
<tr>
<td>glShaderX()</td>
<td>Build Shader structure</td>
</tr>
<tr>
<td>glBufferX()</td>
<td>Build Vertex/Index buffer structures</td>
</tr>
<tr>
<td>glDraw() glShaderX() glDraw() ...</td>
<td>Build Vertex Job Structure Link with vertex/index/shader structures</td>
</tr>
<tr>
<td>eglSwapBuffers()</td>
<td>Build Fragment Job Structure Link with FB and last Vertex Job</td>
</tr>
</tbody>
</table>

### Tex Structure
- Format, Dimensions
- Data Pointer

### Buffer Structure
- Format, Dimensions
- Data Pointer

### Shader1 Structure
- Program Pointer

### Shader2 Structure
- Program Pointer

### VertexJob1 Structure
- Buffer Pointer
- Index Pointer
- Shader Pointer

### VertexJob2 Structure
- Buffer Pointer
- Index Pointer
- Shader Pointer

### FragmentJob Structure
- FrameBuffer Pointer
- Last Job Pointer
GPU hardware job types

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Job(V)</td>
<td>Vertex shader running on a set of vertices</td>
</tr>
<tr>
<td>Tiler Job(T)</td>
<td>Tiling Unit (Fixed Function) work to split transformed primitives into affected tiles</td>
</tr>
<tr>
<td>Fragment Job(F)</td>
<td>Single render target job running over all tiles</td>
</tr>
</tbody>
</table>

Job Chain

A chain of jobs

[Diagram of Job Chains 1 and 2]
GPU job progression

Tile-based Renderer Data Flow

GPU: Vertex Shader → Tiler → Fragment Shader → Local Tile Memory

DDR: Attributes → Geometry Working Set → Textures → Compressed Framebuffer
GPU Software Interface

- **CPU**
  - **User**
  - **Submit chains**
  - **Kernel**
    - **MMU**
    - **Power**
    - **Interruptions**

- **GPU**
  - **Job Manager**
  - ** AXI**

- **External Shared Memory**
  - **VertextJob1 Structure**
  - **VertextJob2 Structure**
  - **TilingJob1 Structure**
  - **TilingJob2 Structure**
  - **FragmentJob Structure**
Job Manager

VertexJob (96 vertices)

- VTask1 (32)
- VTask2 (32)
- VTaskN (32)

TilingJob

- TTask1

FragmentJob (Full Frame)

- FTask1 (Tile1)
- FTask2 (Tile2)

Job Manager

VertexJob

- VTaskN
- VTask2
- VTask1

TilingJob

- TTaskN
- TTask2
- TTask1

FragmentJob

- FTask3
- FTask2
- FTask1

Shader Core

Tiling Unit

External Shared Memory

External Shared Memory

Shade Core
Unified Shader Core

- **Vertex Frontend**
- **Fragment Frontend**
- **Fragment Backend**
- **External Shared Memory**
  - Tiler Data
  - Attributes/Uniforms in Varyings out
  - Textures
  - Framebuffer

**Tripipe**

- Arithmetic
- Load / Store / Varying

**Thread Issue**

**Thread Completion**

**Tile Buffer**
Shader Core: Vertex Work

- Vertex Threads do not write to tile buffers but directly to main memory
- Vertex Tasks contain multiple of 4 vertices
Tiling Unit

- Hierarchy Level 0 - 16x16 tile bins
- Hierarchy Level 1 - 32x32 tile bins
- Hierarchy Level 2 - 64x64 tile bins

Tiler’s goal

Find out what tiles are covered by a primitive. Update tile structure with that info.

Assumptions

Small primitive -> Affect few tiles -> Use low hierarchy level -> Save read bandwidth

Large primitive -> Affect many tiles -> Use high hierarchy level -> Save write bandwidth

Heuristic approach

Determine what is best given distribution
Shader Core: Fragment Work

Fragment Front-End

- Polygon List Reader
- Depth-bounds & Hierarchical ZS Test
- Rasterizer
- Early ZS Testing
- FPK Queue (128 Quads)
- Fragment Thread Creator

Fragment Back-End

- Tripipe
- Tile Writeback
- Tile Memory
- Blending
- Late ZS Testing
Shader Core: Tripipe details

- SIMD processing core that executes instructions from the Midgard ISA

- 3 types of pipes, but 5 total
  - 3 Arithmetic
  - 1 Load/Store
  - 1 Texture

- 128-bit operands, float or integer
  - 2xFP64, 4xFP32, 8xFP16

- Up to 256 active threads at the same time
Power Consumption and Bandwidth Reduction
Power Consumption

- **Power comparison**
  - Phone SoC 1-3W
  - Tablet ~10W
  - Desktop Gfx card ~100-200W

- **Constraints**
  - Battery life time & heat dissipation

- **Static and Dynamic power consumption**
  - Static power. Related to technology and area. Improved through power management (gating)

- **Performance Density**
  
  FPS/mm^2. Smaller Area and less dynamic power -> possibly more cores
Reduce Overdraw: Forward Pixel Kill

Insert small FIFO for 2x2 quads after EarlyZ but before entering tripipe.

Reject quads "in front" before they reach expensive tripipe execution.
Transaction Elimination

- Signature/Hash calculated of color tilebuffer before write-out and saved.
- If it matches previous signature, write-out is skipped saving bandwidth.

Typical GPU has to write out all tiles

Green tile-overlay shows tiles are not changing
ARM Framebuffer Compression/AFBC

- Real-time, lossless, small-area framebuffer compression technology
- Used on both external (window) and internal (FBO) buffers for GPU
- Formats: RGB/YUV (including 10-bit), depth/stencil

Diagram:

INPUT 4K VIDEO STREAM → MALI VPU → AFBC ENCODE/DECODE → MALI GPU → AFBC ENCODE/DECODE → MALI DPU → AFBC DECODE → OUTPUT TO 4K-DISPLAY

- 1: AFBC COMPRESSED REFERENCE FRAMES
- 2: AFBC COMPRESSED VIDEO
- 3: INTERMEDIATE RENDER TARGETS
- 4: FRAME BUFFER EXTERNAL MEMORY
EXT_shader_pixel_local_storage

- Bandwidth Efficient Deferred Shading
- Define G-Buffer in Pixel Local Storage space (in tile memory)

G-Buffer initialization

```c
__pixel_local_outEXT FragData
{
    layout(rgba8) highp vec4 Color;
    layout(rg16f) highp vec2 NormalXY;
    layout(rg16f) highp vec2 NormalZ_LightingB;
    layout(rg16f) highp vec2 LightingR8;
} gbuf;

void main()
{
    gbuf.color = calcDiffuseColor();
    vec3 normal = calcNormal();
    gbuf.NormalXY = normal.xy;
    gbuf.NormalZ_LightingB.x = normal.z;
}
```

Lighting accumulation

```c
__pixelLocalEXT FragData
{
    layout(rgba8) highp vec4 Color;
    layout(rg16f) highp vec2 NormalXY;
    layout(rg16f) highp vec2 NormalZ_LightingB;
    layout(rg16f) highp vec2 LightingR8;
} gbuf;

void main()
{
    vec3 lighting = calcLighting(gbuf.NormalXY.x,
                                gbuf.NormalXY.y,
                                gbuf.NormalZ_LightingB.x);
    gbuf.LightingRG = lighting.x;
    gbuf.NormalZ_LightingB.y += lighting.z;
}
```

Final shading

```c
__pixel_local_inEXT FragData
{
    layout(rgba8) highp vec4 Color;
    layout(rg16f) highp vec2 NormalXY;
    layout(rg16f) highp vec2 NormalZ_LightingB;
    layout(rg16f) highp vec2 LightingR8;
} gbuf;

out highp vec4 fragColor;

void main()
{
    fragColor = resolve(gbuf.Color,
                        gbuf.LightingRG,
                        gbuf.LightingR6.x,
                        gbuf.NormalZ_LightingB.y);
}
```
ARM Lund
ARM Lund GPU teams

- Hardware design/verification 15 (Fragment front/back-end)
- GPU Modelling 17
- Graphics/Backend Compiler team 16
- HW/SW regression testing 10
- Multimedia(GPU+VPU+DPU) development+regression 4

ARM Lund total ~100. Remaining are Video IP development.

Cambridge/Trondheim major GPU development sites
GPU HW Development

- HDL: Verilog + System Verilog
- git/gerrit
- Cluster based RTL simulation framework
- HW Engineers usually with low-spec desktop/laptop.
  - Everything runs on massive CPU clusters in Cambridge
GPU SW Development

- **Sw tools:** git/gerrit/svn, scons (DDK configuration), codecollaborator, JIRA
- **GPU DDK release cycle:** scrum-based. Release every iteration.
  - **Iteration:** ~2 months
  - **Sprints:** 2 weeks. 4 sprints/iteration
- **Regression Server Test Framework:** Internal
  - Handles FPGA board flashing, driver building, board allocation and test execution, automatic bug filing with git bisect
- **Hardware platforms**
  - ARM Juno/VersatileExpress + single/multiple FPGA tiles
  - Silicon development platforms with Mali
  - Mali GPU Model (bit accurate, close to cycle accurate)
FPGA Lab
ARM Lund Open-House Event TONIGHT (6/12)

- **Where**
  - Tuesday **December 6:th @17.15** Emdalavägen 6, 4:th floor

- **What**
  - Food/Drinks/Snacks
  - Lund Team ”stations”. Talk to a hw designer or compiler engineer
  - Graphics & Video HW/SW presentations
  - Demos
  - Mingle
ARM Lund Student opportunities

- Graduate job offerings – apply through www.arm.com/careers
- Internship (part/full time) – apply through www.arm.com/careers
- Master thesis work – see adds on hand outs, apply to student-se@arm.com

- If you have any questions or specific proposals on how to engage with ARM, just drop an email to student-se@arm.com
Midgard Shader Core: Fragment Frontend

- **Polygon List Reader / Triangle Setup**
  - Loads primitives from the tiler polygon list; ~13 cycles a triangle

- **Depth-bounds and Hierarchical-Z S Testing**
  - Conservative fast cull of primitives based on line equations

- **Rasterizer**
  - Generate 2x2 fragment quads with per per-fragment coverage mask

- **Early ZS Testing**
  - Cull quads based on ZS values, if possible
  - Those which pass ZS may cull old quads in FPK Queue

- **FPK Queue**
  - Buffer with 128 quads; Freya supports priority quad issue into FTC

- **Fragment Thread Creator**
  - Spawn a quad as four fragment threads over four cycles
Midgard Shader Core: Fragment Backend

- **Late ZS Testing**
  - Resolve any outstanding ZS tests we couldn’t do early

- **Blending**
  - Blend up to 4 samples per clock for subset of blend equations
    - 4x MSAA needs 1 cycle, 8x needs 2 cycles, and 16x needs 4 cycles
    - Complex blend operations implemented using blend shaders

- **Tile Memory**
  - Storage for color and depth+stencil data
  - Mali-T760 onwards provide flexible allocation for MRT and/or MSAA

- **Tile Writeback**
  - Requires one cycle per output pixel
  - Provides the CRC generation for transaction elimination
  - AFBC compression and YUV writeback supported (Mali-T760 onwards)
Smart Composition

- Selectively update parts of a frame

- Khronos EGL extensions
  - KHR_partial_update
  - EXT_buffer_age

- Producer can utilize age of existing backbuffer contents to draw frame with multiple bounding boxes

- Similar to TE, but here we wont even write to the tilebuffer