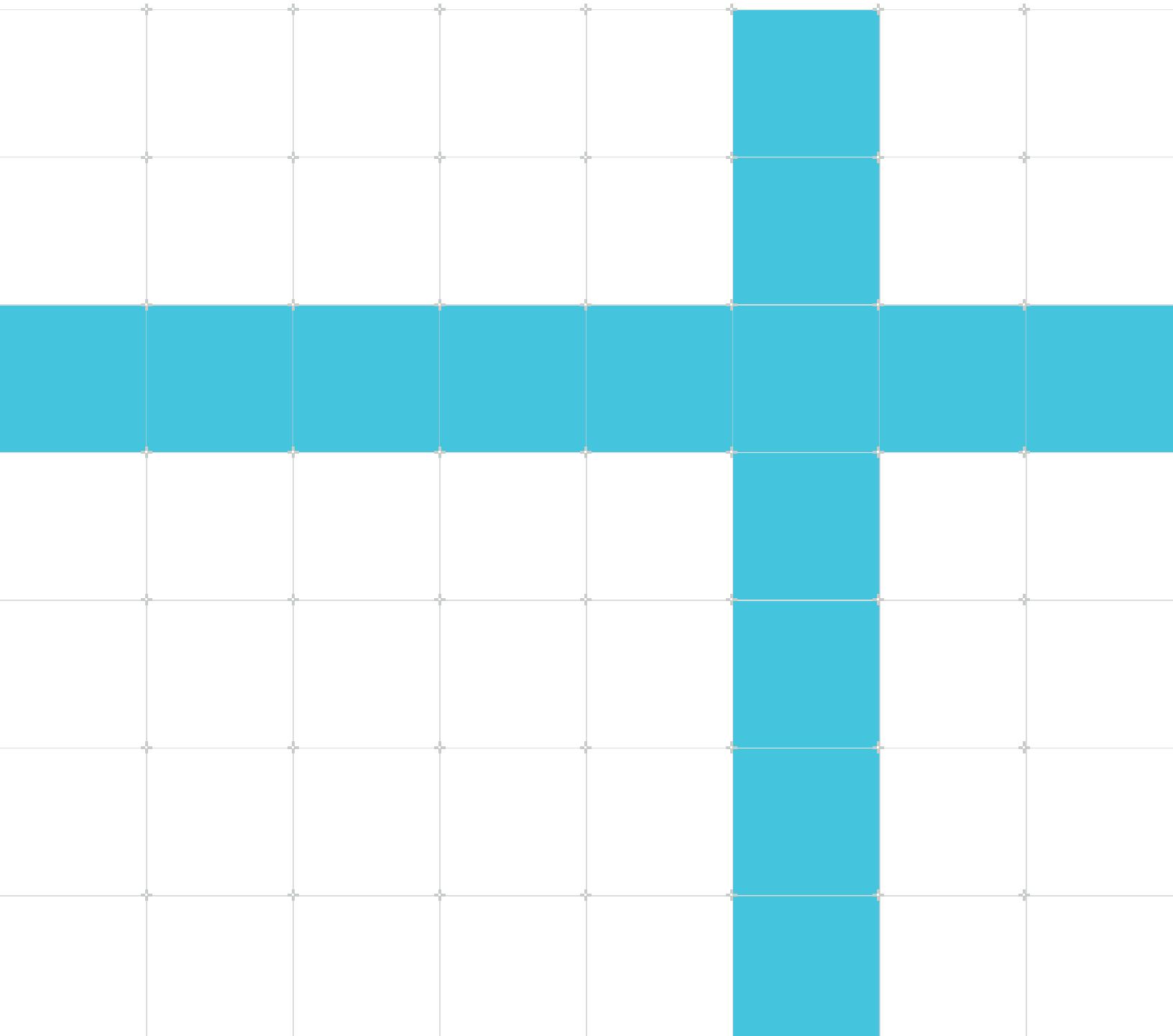
Learn the basics - The Midgard Shader Core

Version 1.0

Issue 02
102560_0100_02_en



Learn the basics - The Midgard Shader Core

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	19 August 2019	Non-Confidential	First release

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

This guide discusses a typical Mali Midgard GPU programmable core. Midgard is the second-generation Mali GPU architecture, and the first to support OpenGL ES 3.0 and OpenCL. The Midgard family includes the Mali-T600, Mali-T700, and Mali-T800 series products.

To optimize the 2D and 3D performance of your applications, you need a high-level understanding of how the hardware works. For example, understanding the Mali GPU block architecture is important when optimizing using the performance counters of the GPU. This is because this counter data is tied directly to GPU blocks.

By the end of this guide, you will have a better understanding of how the Mali Midgard series of GPUs perform shader core operations through shared access to the L2 cache, how the three different pipelines of the Tripipe execution core work, and the benefits of early ZS testing.

Before you begin This guide assumes that you understand the tile-based rendering approach that the Mali GPUs adopt. For more information, read our guide on Tile-Based Rendering.

2. Midgard GPU Architecture

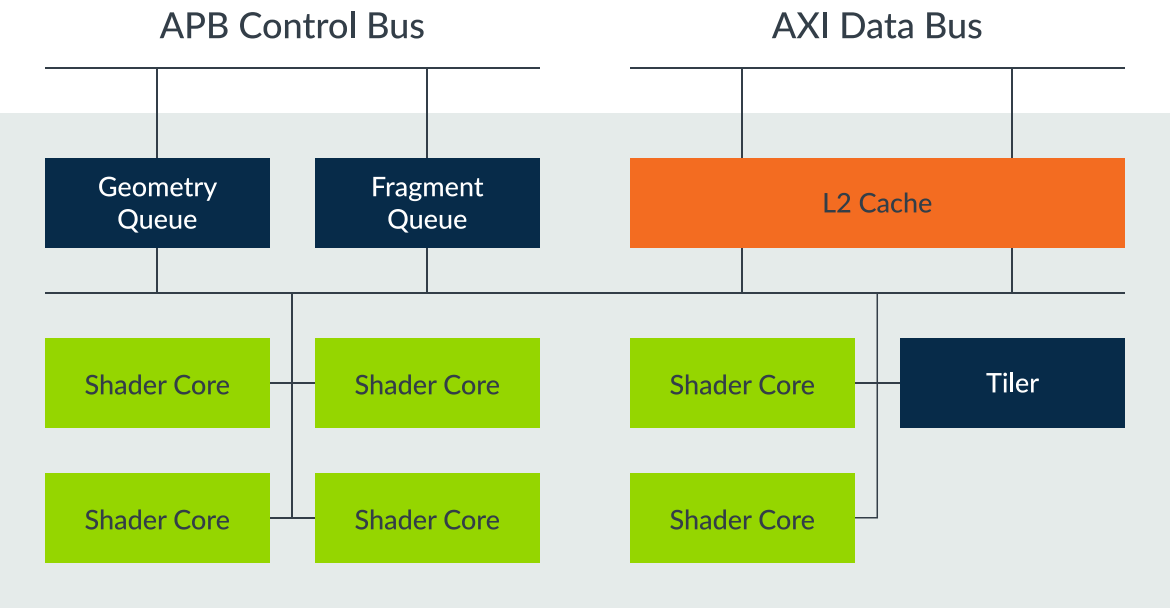
The Midgard family of Mali GPUs use a unified shader core architecture. This means that only a single type of hardware shader processor exists in the design. This single shader processor can execute all types of shader code, such as vertex shaders, fragment shaders and compute shaders.

The exact number of shader cores in a silicon chip can vary. We license configurable designs to our silicon partners, who can then choose how to configure the GPU in their specific chipset, based on their performance needs and silicon area constraints.

For example, the Mali-T880 GPU can scale from a single core, for low-end devices, up to 16 cores for the highest performance designs.

The following diagram provides a top-level overview of the Control Bus and Data Bus of a typical Mali Midgard GPU:

Figure 2-1: Midgard top level



To improve performance, and to reduce memory bandwidth wasted on repeated data fetches, the shader cores in the system all share access to a level 2 cache. The size of the L2 cache, while configurable by our silicon partners, is typically in the range of 32-64KB per shader core in the GPU.

Also, our silicon partners can configure the number, and bus width, of the memory ports that the L2 cache has to external memory.

The Midgard architecture aims to write one 32-bit pixel, per core, per clock. Therefore, it is reasonable to expect an eight-core design to have a total of 256-bits of memory bandwidth, for both read and write, per clock cycle. This can vary between chipset implementations.

2.1 Work issue

Once the application has completed defining the render pass, the Mali driver submits a pair of independent workloads for each render pass.

The first pass handles all geometry and compute related workloads. The second pass is for the fragment-related workload. Because Mali GPUs are tile-based renderers, all geometry processing for a render pass must be complete before the fragment shading can begin.

A finalized tile rendering list is required to provide the fragment that is processing the per-tile primitive coverage the information that it needs.

Midgard GPUs can support two parallel issue queues that the driver can use, one for each workload type. Geometry and fragment workloads from both queues can be processed in parallel by the GPU. This arrangement allows the workload to be distributed across all available shader cores in the GPU.

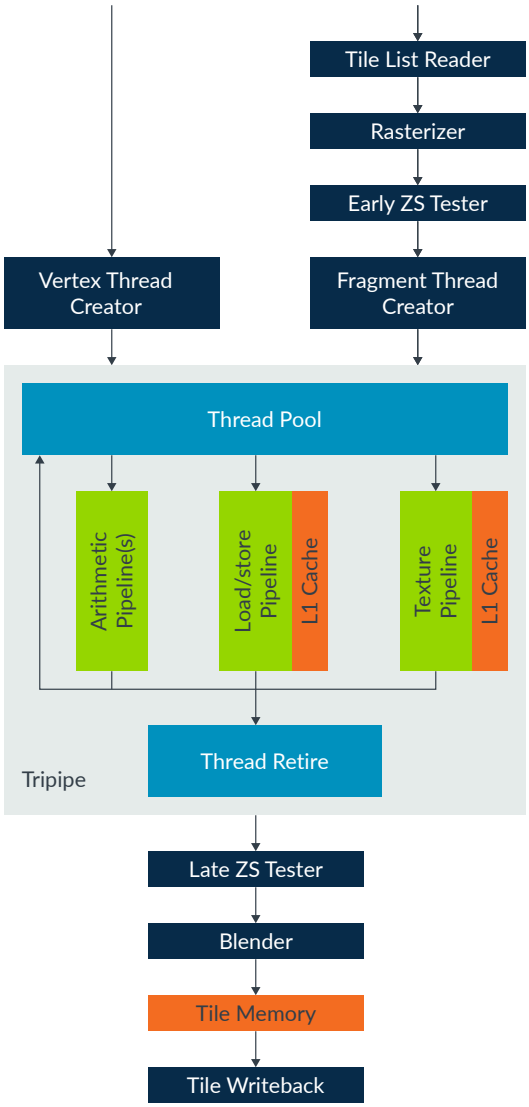
3. Midgard Tripipe Execution Core

A programmable Tripipe execution core, with several fixed-function units wrapped around it, make up the structure of the GPU's shader core.

These fixed function units perform the setup for a shader operation or handle the post-shader activities.

The image below shows the Tripipe execution core and the fixed-function units that surround it:

Figure 3-1: Midgard shader core



The programmable shader core is a multi-threaded processing engine that can run hundreds of threads simultaneously, where each running thread equals a single shader program instance.

This large number of threads exists to hide the costs of cache misses and memory fetch latency. Performance drops due to cache misses can be avoided if some of the threads are ready to run at each cycle. This is the case even if many of the other threads are stalled and blocked fetching data.

3.1 The Tripipe

The Tripipe is the programmable part of the core responsible for the execution of shader programs and it contains the following classes of parallel execution pipeline:

- The arithmetic pipeline, or A-pipe, handles all arithmetic processing.
- The load/store pipeline, or LS-pipe, handles all general purpose memory access, varying interpolation, and read/write image access.
- The texture pipe, or T-pipe, handles all read-only texture accesses and texture filtering.

While all Midgard shader cores have one load/store pipe and one texture pipe, the number of arithmetic pipelines can vary depending on which GPU you are using:

- The Mali-T720 and T820 GPUs have one each.
- The Mali-T880 has three.
- All remaining Midgard GPUs have two.

3.2 The Arithmetic pipeline

The Arithmetic pipeline (A-pipe), is a Single Instruction Multiple Data (SIMD) vector processing engine, with arithmetic units that operate on 128-bit quad-word registers. The registers can be accessed as several different data types, for example, as 4xFP32/I32, 8xFP16/I16, or 16xI8. Therefore, a single arithmetic vector operation can process eight `mediump` values in a single cycle.

Although we cannot disclose the internal architecture of the A-pipe, the publicly available performance data for each Midgard GPU gives some idea of the number of mathematical units available. For example, the Mali-T760, using 16 cores, is rated at 326 FP32 GFLOPS at 600MHz.

Using a T760, using 16 cores, equals a total of 34 FP32 FLOPS per clock cycle, per shader core. Because the Mali-T760 has two pipelines, the final mathematical units available is 17 FP32 FLOPS per pipeline. The operational performance increases for smaller data types and decreases for larger ones.

3.3 Texture pipeline

The Texture pipeline (T-pipe) is responsible for all texture-related memory accesses. It can return one bilinear filtered texel per clock for most texture formats. However, performance can vary for some texture formats and filtering modes.

The following table shows that trilinear filtering of a 3D texture would require four cycles, where two cycles are used for trilinear filtering, and a further two cycles for the 3D texture:

Operation	Performance Scaling
Trilinear filter	x2
3D format	x2
Depth format	x2
32-bit channel format	x channel count
YUV format	x planes

At a cost of one cycle per plane, importing YUV surfaces from camera and video sources can be read directly without needing prior conversion to RGB. However, importing semi-planar Y + UV sources are preferred to fully planar Y + U + V sources.

The T-pipe has a 16KB L1 data cache.

3.4 Load/Store pipeline

The Load/Store pipeline (LS-pipe) is responsible for all shader memory accesses that are not related to texture samplers. This responsibility includes generic pointer-based memory access, buffer access, varying interpolation, atomics, and `imageLoad()` and `imageStore()` accesses for mutable image data.

Although every instruction is a single cycle memory access operation, the hardware also supports wide vector operations and can load an entire `highp` vec4 varying in a single cycle.

The LS-pipe has a 16KB L1 data cache.

3.5 Early and late ZS testing

In the OpenGL ES specification, 'fragment operations', which includes depth (Z) and stencil (S) testing, occurs at the end of the pipeline after fragment shading has completed.

Although the specification is simple to understand, it also implies that fragment shading must occur, even if it is then thrown away afterwards by ZS testing.

Coloring fragments and then discarding them costs a significant amount of performance and wasted energy. So, where possible, early ZS testing is performed before fragment shading occurs. Late ZS testing occurs after fragment shading only when unavoidable. For example, a dependency on a shader fragment that calls discard creates an indeterminate depth state until it exits the Tripipe.

Midgard GPUs also add a hidden surface removal capability, called Forward Pixel Kill (FPK). FPK can stop rasterized fragments, that have already passed ZS testing, from turning into real rendering work. However, the Mali stack must first determine that the fragments are occluded and do not contribute to the output scene in a useful way.

4. Performance Expectations

Here are some of the fundamental performance properties that you can expect from a Mali Midgard GPU. The GPU can do the following tasks simultaneously:

- Issue one new thread per shader core per clock.
- Retire one fragment per shader core per clock.
- Write one pixel per shader core per clock.
- Issue one instruction per pipe per clock.

From the point of view of shader core performance:

- Each A-pipe can process:
  - 17 FP32 operations per clock.
- The LS-pipe can process:
  - 128-bits of vector load/store per clock, or
  - 128-bits of varying interpolation per clock, or
  - one `imageLoad()` or `imageStore()` per clock, or
  - one atomic access per clock.
- The T-pipe can process:
  - One bilinear filtered sample per clock, or
  - One bilinear filtered depth sample every two clocks, or
  - One trilinear filtered sample every two clocks, or
  - One plane of YUV sampled data every clock.

4.1 Theoretical peak performance example

If we scale the shader core performance to match an example reference chipset that contains a Mali-T760 MP8 running at 600MHz, we can calculate the theoretical peak performance as:

Fill rate:

- 8 pixels per clock = 4.8 GPix/s.
- That is 2314 complete 1080p layers per second.

Texture rate:

- 8 bilinear texels per clock = 4.8 GTex/s.
- That is 38 bilinear filtered texture samples per pixel for 1080p @ 60 FPS.

Arithmetic rate:

- 17 FP32 FLOPS per pipe per core = 163 FP32 GFLOPS.

- That is 1311 FLOPS per pixel for 1080p @ 60 FPS.

Bandwidth:

- 256-bits of memory access per clock = 19.2GB/s read and write bandwidth.
- That is 154 bytes per pixel for 1080p @ 60 FPS.

4.2 Platform dependencies

The performance of a Mali GPU in any specific chipset is very dependent on both the configuration choices made by the silicon implementation, and the final device form factor the chipset is used in.

Some characteristics, including the number of shader cores and the size of the GPU L2 cache, are visible in terms of the GPU logical configuration that the silicon partner has built.

Other characteristics depend on the memory system's logical configuration, like the memory latency, bandwidth, DDR memory type, and how memory is shared between multiple users.

Some characteristics depend on analogue silicon implementation choices, like which silicon process was used, the target top frequency, the DVFS voltage, and frequency choices available at run time.

Finally, some characteristics depend on the physical form factor of the device, because this determines the available power budget. Therefore, an identical chipset can have very different peak performance results in different form factor devices.

For example:

- A small smartphone has a sustainable GPU power budget between 1-2 Watts.
- A large smartphone has a sustainable GPU power budget between 2-3 Watts.
- A large tablet has sustainable GPU power budget between 4-5 Watts.
- An embedded device with a heat sink may have a GPU power budget up to 10 Watts.

When combined, it can be hard to predict the performance of any GPU implementation based solely on the GPU product name, core count, and top frequency. If in doubt, write some test scenarios that match your own use cases, and then run them to see how well they work on your target devices.

5. Next steps

Understanding how the Tripipe execution core of a Midgard GPU functions, how the shader cores share access to the L2 cache that sits on the Data Bus, and the benefits of performing early ZS tests presents you with an opportunity to better optimize for this GPU architecture.