The Bifrost Shader Core

Version 1.0

Non-Confidential

Issue 02
102546_0100_02_en

The Bifrost Shader Core

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	19 August 2020	Non-Confidential	First release

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly

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 describes the top-level layout and the benefits of and shader core functionality of a typical Mali Bifrost GPU programmable core, the third generation of Mali GPUs. The Bifrost family includes the Mali-G30, Mali-G50, and Mali-G70 series of products.

When optimizing 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 particularly 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 understand how the Mali Bifrost series of GPUs perform shader core operations using shared access to the L2 cache. You will also learn the benefits of the Warp Manager and the Execution Engines (EE) within the Execution Core, and the benefits of the Index-Driven Vertex Shading (IDVS) geometry pipeline.

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. Third Gen Mali GPU Architecture

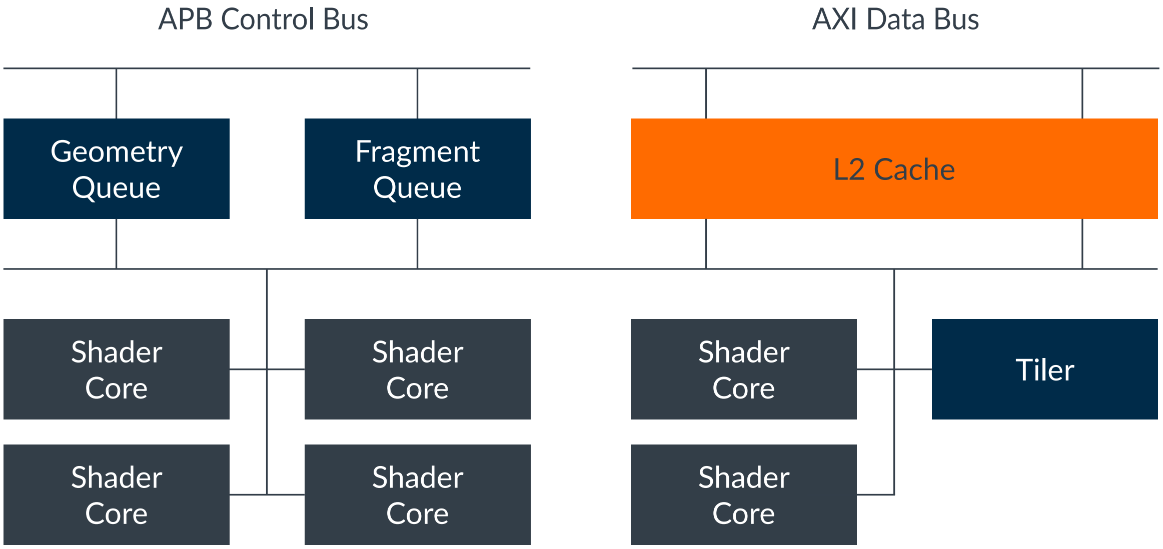
The Bifrost family of Mali GPUs use the same top-level architecture as the earlier Midgard GPUs. They use a unified shader core architecture. This means that the design only includes a single type of hardware shader processor. This single shader processor can execute all types of shader code, such as vertex shaders, fragment shaders, and compute kernels.

The exact number of shader cores present 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-G72 GPU can scale from a single core, for low-end devices, up to 32 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 Bifrost GPU:

Figure 2-1: Bifrost top-level overview



To improve performance, and to reduce memory bandwidth wastage caused by 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 64-128KB per shader core in the GPU. However, the size of the L2 cache depends on how much silicon area is available.

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

The Bifrost 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.

Work issue

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

One independent pass deals with all geometry and compute related workloads. The other independent pass is for the fragment-related workload. As 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.

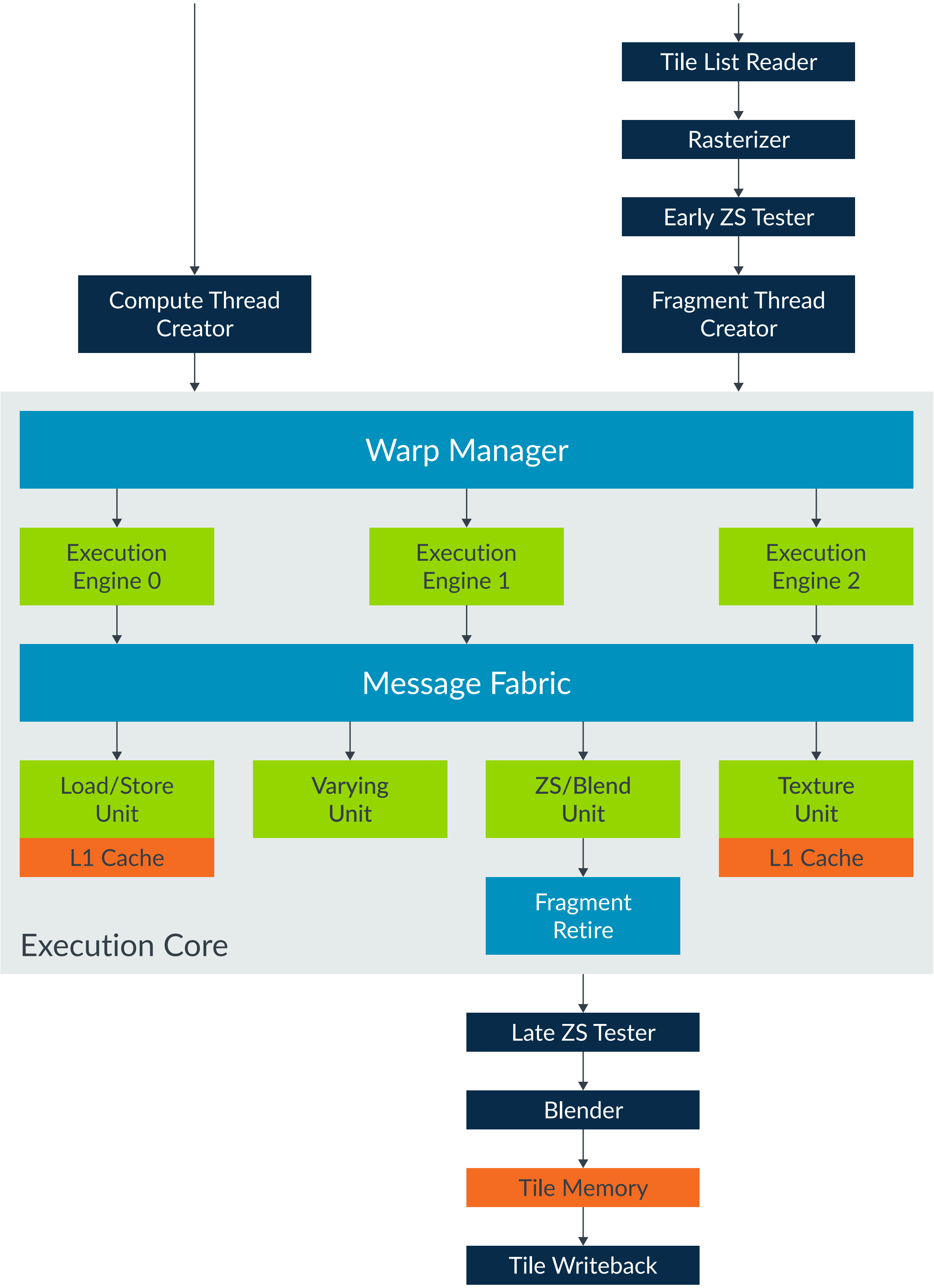
Bifrost GPUs can support two parallel issue queues, that the driver can use, one for each workload type. Geometry and fragment workloads from both queues can be processed in parallel by the GPU at the same time. This arrangement allows the workload to be distributed across all available shader cores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

3. The Bifrost Shader Core

All Mali shader cores are structured with several fixed-function hardware blocks wrapped around a programmable core. The programmable core is the most significant change between the Bifrost GPU family and the earlier Midgard GPU designs.

The following image shows a single execution core implementation and the fixed-function units that surround it:

Figure 3-1: Bifrost shader core



The programmable Execution Core (EC) on a Bifrost GPU consists of one or more Execution Engines (EEs). The Mali-G71 has three EEs, and several shared data processing units, all of which are linked by a messaging fabric.

Depending on which product configuration you choose, Bifrost shader cores come in two sizes:

1. Small shader cores can read one texture sample, blend one fragment, and write one pixel per clock.
2. Large shader cores can read two texture samples, blend two fragments, and write two pixels per clock.

The execution engines

Each EE executes the programmable shader instructions. Each EE includes a single composite arithmetic processing pipeline, and the required thread state for the threads that the EE is processing.

Thread state

Bifrost GPUs implement a general-purpose register file for the shader programs that is substantially larger than the register file used on Mali Midgard GPUs. This larger general-purpose register file improves performance, and performance scalability, for complex programs. Programs maintaining full thread occupancy can use up to 32x32-bit registers, and up to 64x64-bit registers can be used by more complex programs, at the expense of thread availability.

Arithmetic processing

To improve processing speed, multiple threads are grouped into bundles, called a warp, before being executed in parallel by the GPU. Functional units are also improved by the arithmetic units in the EE due to the warp-based vectorization scheme.

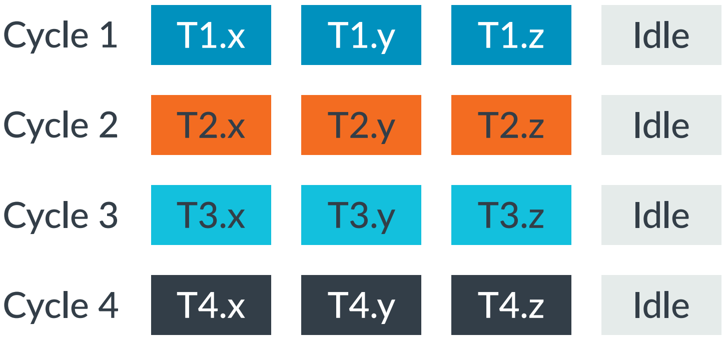
For a single thread, processing looks like a stream of scalar 32-bit operations, therefore ensuring that the shader compiler utilizes the hardware available.

However, the underlying hardware keeps the efficiency benefit of being a vector unit with a single set of control logic that can be amortized over every thread in the warp.

The two diagrams below illustrate the advantages of keeping the hardware units busy, regardless of the vector length in the program.

The diagram below shows how a vec3 arithmetic operation is mapped onto a pure SIMD unit, where one idle cycle per operation is visible:

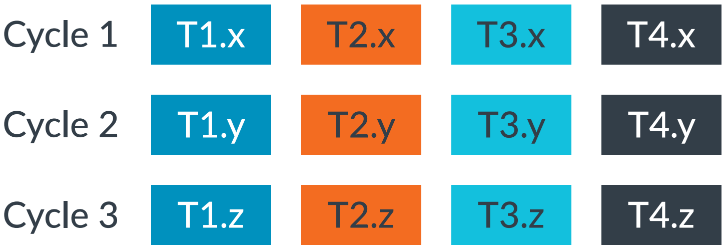
Figure 3-2: vec3 operation on pure SIMD unit



Cycle 1	T1.x	T1.y	T1.z	Idle
Cycle 2	T2.x	T2.y	T2.z	Idle
Cycle 3	T3.x	T3.y	T3.z	Idle
Cycle 4	T4.x	T4.y	T4.z	Idle

Compare this to a 4-wide warp unit, as shown in the following image, where the pipeline executes one lane per thread for four threads per clock:

Figure 3-3: vec3 operation on 4-wide warp unit



Cycle 1	T1.x	T2.x	T3.x	T4.x
Cycle 2	T1.y	T2.y	T3.y	T4.y
Cycle 3	T1.z	T2.z	T3.z	T4.z

Note

The warp width of the Bifrost shader cores can vary. Single-pixel per-core per-cycle products have a 4-wide warp, while the two-pixel per-core per-cycle use an 8-wide warp.

Bifrost maintains native support for int8, int16, and fp16 data types and these data types can be packed to fill the 128-bit data width of the data unit. This arrangement maintains the power efficiency and performance provided by the narrower than 32-bit types.

A single 4-wide warp math's unit can perform either 8x fp16/int16, or 16x int, 8 operations per clock cycle.

Load/store unit

The load/store unit is responsible for all shader memory accesses that are not related to texture samplers. This responsibility includes generic pointer-based memory access, buffer access, atomics, and `imageLoad()` and `imageStore()` accesses for mutable image data.

Data stored in a single 64-byte cache line per clock cycle can be accessed. Warp unit accesses are optimized reduce unique cache access requests.

For example, data can be returned in a single cycle if all threads access data inside the same cache line.

We recommend that you use shader algorithms that exploit the wide data access, and cross-thread merging functionalities, of the load/store unit.

For example:

- Use vector loads and stores in each thread.
- Access sequential address ranges across the threads in a warp.

Remember, the load/store unit includes 16KB L1 data cache per core, which is backed up by the shared L2 cache.

Varying unit

The varying unit is a dedicated fixed-function varying interpolator. The varying unit ensures good functional unit utilization by using warp vectorization.

For every clock, the varying unit can interpolate 32-bits for every thread in a warp. For example, interpolating a `mediump` - fp16 - vec4 takes two cycles. Therefore, interpolation of a `mediump` fp16 is both faster, and more energy efficient, than interpolation of a `highp` fp32.

Texture unit

The texture unit implements all texture memory accesses, and offers 16KB of L1 data cache per texel, per clock. The texture unit is backed by the shared L2 cache.

The architectural performance of the texture unit block is variable, with a texels per clock peak performance that matches the pixels per clock of the shader core. And, depending on the product's configuration, may be one, or two, bilinear filtered samples per clock.

Although performance can vary for some texture formats and filtering modes, the baseline performance is one or two bilinear filtered texel, per clock, for most texture formats.

The Mali-G71 and Mali-G72 texture unit

In terms of performance, the texture unit in these two Bifrost GPUs are similar to Midgard GPUs. However, depth sampling is optimized to run at full rate.

The following table shows the operations available, and their performance scaling values:

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

The later Bifrost GPUs

As seen in the following table, the Mali G31, G51, G52, and G76 GPUs use an upgraded texturing unit design, significantly reducing the silicon area required, improving energy efficiency:

Operation	Performance scaling
N x anisotropic filter	Up to x N
Trilinear filter	x2
3D format	x2
32-bit channel format	x channel count

The updated Bifrost GPU design significantly improves the performance of many applications importing camera and video streams by optimizing YUV filtering performance. Regardless of the number of input planes used to store data image in memory, bilinear filtered samples can be processed in a single cycle.

ZS and blend unit

The ZS and color blend units are both responsible for handling all OpenGL ES and programmatic accesses to the tile-buffer for functionality like:

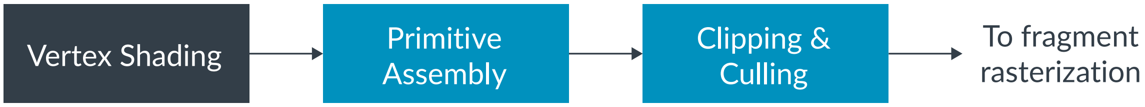
- EXT_shader_pixel_local_storage
- ARM_shader_framebuffer_fetch
- ARM_shader_framebuffer_fetch_depth_stencil

Depending on the shader core size, the blender can write either one or two fragments per clock to the tile memory. All Mali GPUs are designed to support fast Multi-Sample Anti-Aliasing (MSAA). This means that both full rate fragment blending, and the resolving of pixels when using 4xMSAA, are natively supported.

4. Index-Driven Geometry Pipeline

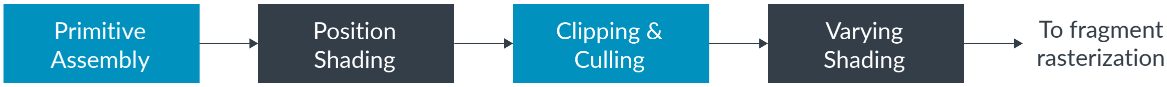
Bifrost introduces an Index-Driven Vertex Shading (IDVS) geometry processing pipeline. Earlier Mali GPUs processed all vertex shading tasks before culling primitives, which could result in wasted computation and bandwidth for the vertices that are only used in culled triangles. The following diagram shows the geometry pipeline that is used in earlier Mali GPUs:

Figure 4-1: Earlier Mali GPU geometry processing pipeline



By contrast, the following diagram shows how the IDVS pipeline starts by building primitives and submitting shading in primitive order, which splits the shader in half. Next, position shading occurs before clipping and culling, then varying shading runs for all, non-culled, visible vertices:

Figure 4-2: Bifrost IDVS geometry processing pipeline



The IDVS pipeline flow provides the following key optimizations:

- Position shading is only submitted for small batches of vertices where at least one vertex in each batch is referenced by the index buffer. This allows vertex shading to jump spatial gaps in the index buffer which are never referenced.
- Varying shading removes some redundant computation and bandwidth. This is because varying shading is only submitted for primitives that survive the clip-and-cull phase.

Also, partially deinterleaving packed vertex buffers ensures that you maximize the benefit from the IDVS geometry flow. Using two packed buffers, place attributes contributing to position in one, and attributes contributing to non-position varyings in the other. Placing the attributes in this way ensures that non-position varyings are not pulled into the cache for vertices that are culled.

5. Next steps

This guide introduced the fundamental principles of the Bifrost Shader Core. If you are interested in previous Mali architecture generations, you can read our guides:

- The Valhall shader core
- The Midgard shader core
- The Utgard shader core

To learn more about the architecture and the products in each architecture, visit the GPU architecture page.