The Bifrost Shader Core

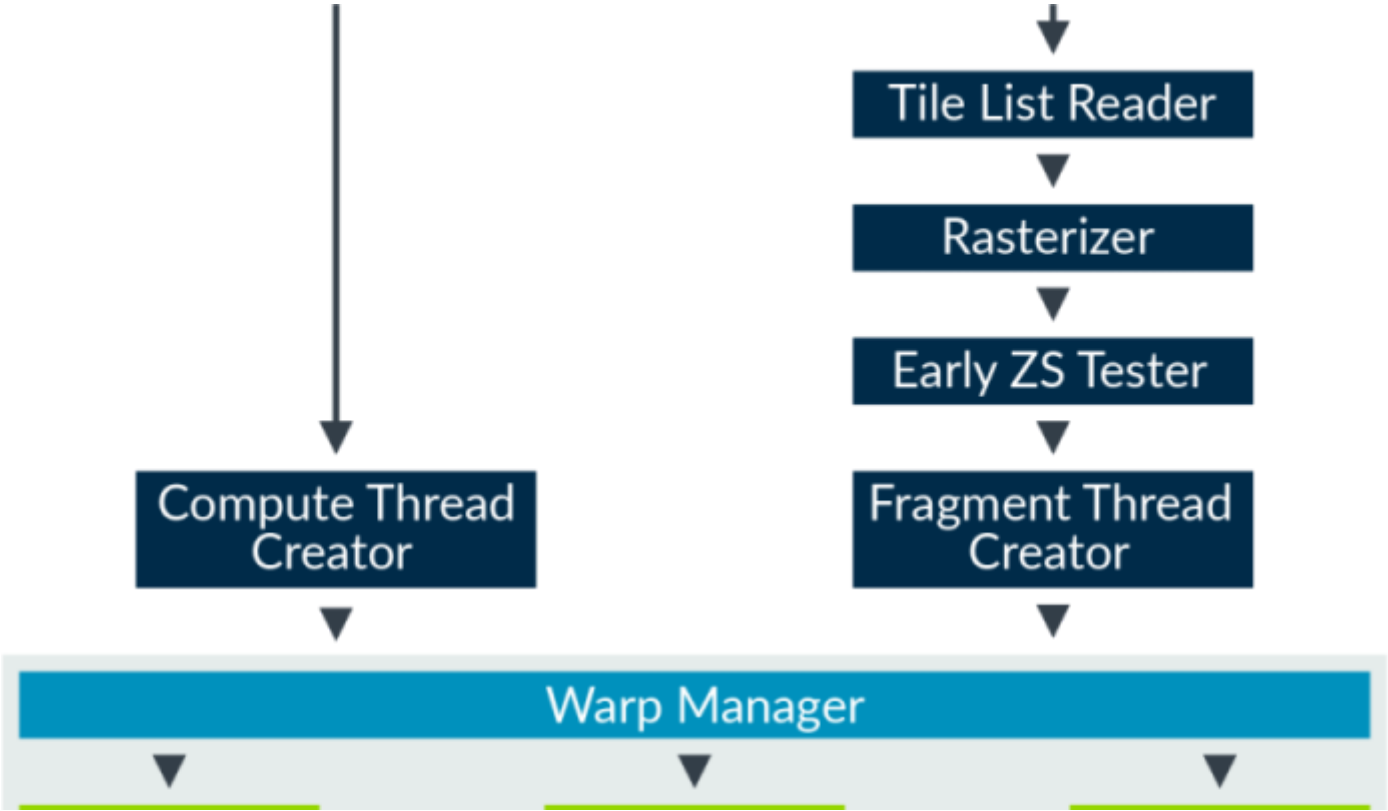
← Previous Section

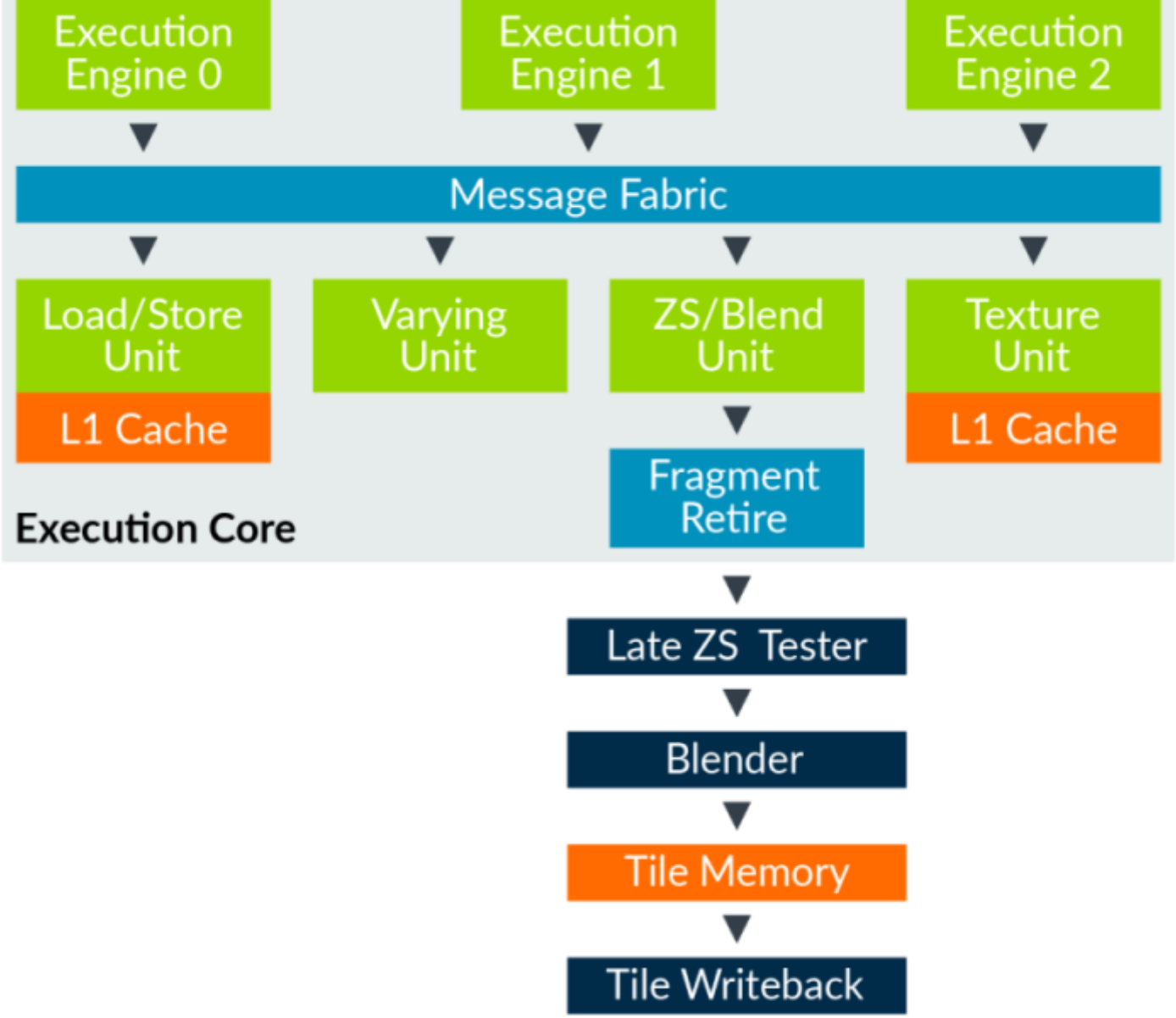
Next Section →

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:





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:

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:

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.

← Previous Section

Next Section →