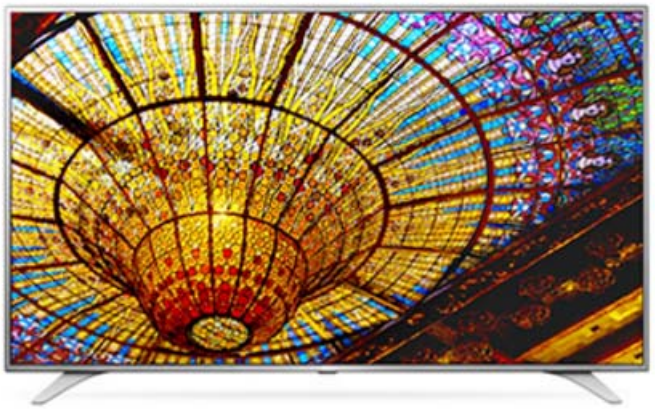
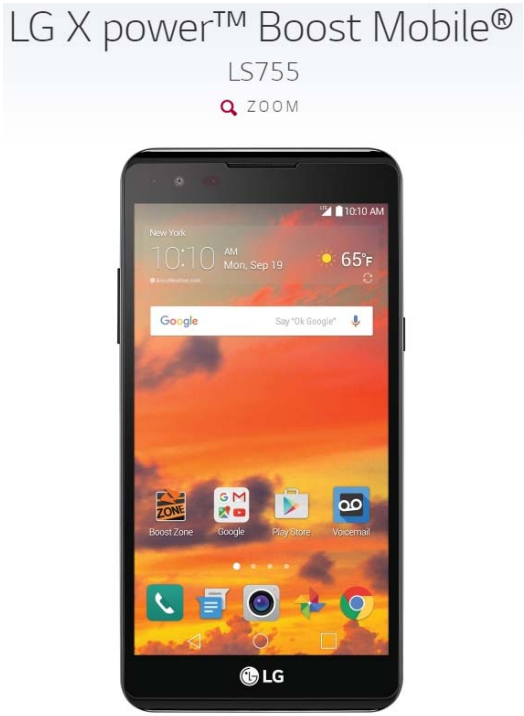
EXHIBIT D

U.S. Patent No. 7,633,506

LG / MediaTek Products

U.S. Patent No. 7,635,506: Claim 1

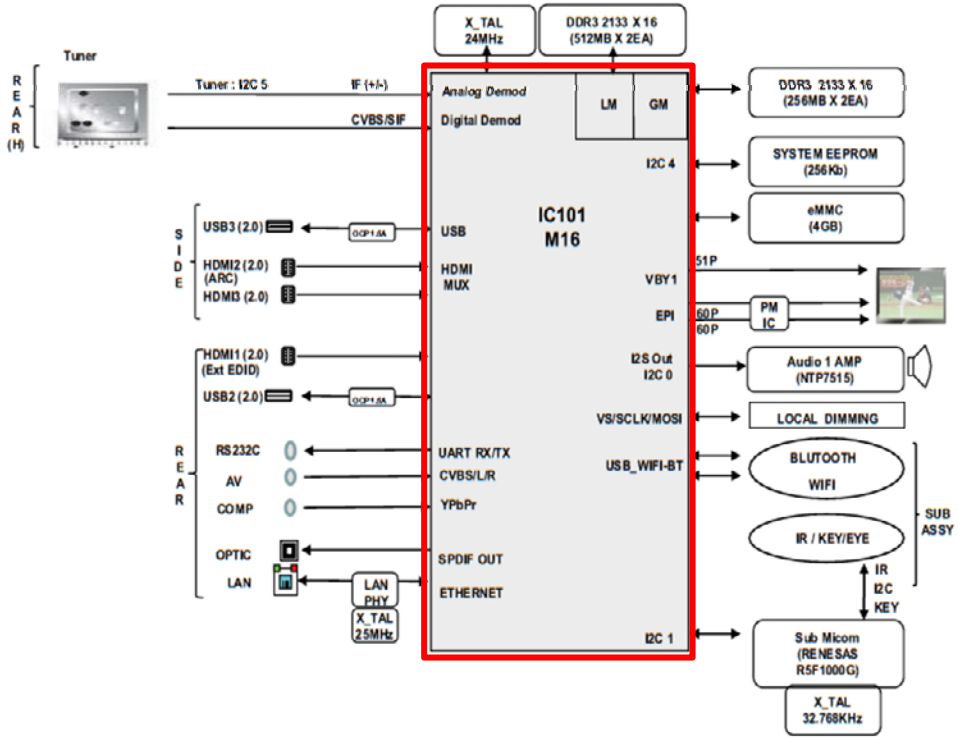
"1. A graphics chip comprising:"

1. A graphics chip comprising:	The LG 49UH6500 television and X Power LS755 phone (collectively, the "LG Products") include a graphics chip.  *See* http://www.lg.com/us/support-product/lg-49UH6500.  *See* http://www.lg.com/us/cell-phones/lg-LS755-x-power-boost-mobile.

U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"

The LG Products include one of the following System-on-Chips (SoCs): M16 and MediaTek MT6755M.



See LG LED TV Service Manual, Chassis: UA63J, Model: 43UH6500, p.28, *available at* https://lg.encompass.com/shop/model_research_docs/?file=/ZEN/sm/43UH6500UB.pdf.[1/]

[1/] The LG 49UH6500 television and the LG 43UH6500 television are part of the LG UH6500 Series televisions. *See* http://www.lg.com/us/support/products/documents/UH6500_Series_Spec_Sheet_Updated_10112016.pdf.

U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"

Technical Specifications

Carrier	Boost Mobile®
Display	5.3" (1280 x 720) HD TFT Display
Battery	4,100 mAh non-removable
Platform	Android 6.0.1 Marshmallow
Processor	MediaTek 1.8 GHz Octa-Core MT6755M

See http://www.lg.com/us/cell-phones/lg-LS755-x-power-boost-mobile.

The SoCs include one of the following ARM Mali graphics processing units (the "Mali GPUs"): T760 MP2, T760 MP4, and T860 MP2.

		M16
Smart Function	CPU	CA53 x4 1.1GHz / 1MB
	GPU	Mali T760 MP2 (650MHz)
	OSD	Separated 2K@60p
	HEVC	4K @60,10bit
	DDR	DDR3-2133/ DDR4-2400
	Audio DSP	HiFi3 Dual @370MHz

See LG LED TV Service Manual, Chassis: UA63J, Model: 43UH6500, p.123, *available at* https://lg.encompass.com/shop/model_research_docs/?file=/ZEN/sm/43UH6500UB.pdf.

MediaTek MT6755 Helio P10 Specs

Release	Q4 2015
Process	28nm
Apps CPU	8x Cortex-A53, up to 2.0GHz
GPU	ARM Mali-T860 MP2 at 700 MHz

See http://cnoemphone.com/blog/mediatek-mt6755-helio-p10-specs-benchmark-and-smartphone-list.

U.S. Patent No. 7,635,506: Claim 1

"1. A graphics chip comprising:"

	The Mali GPUs share substantially similar structure, function, and operation.

U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"

High performance

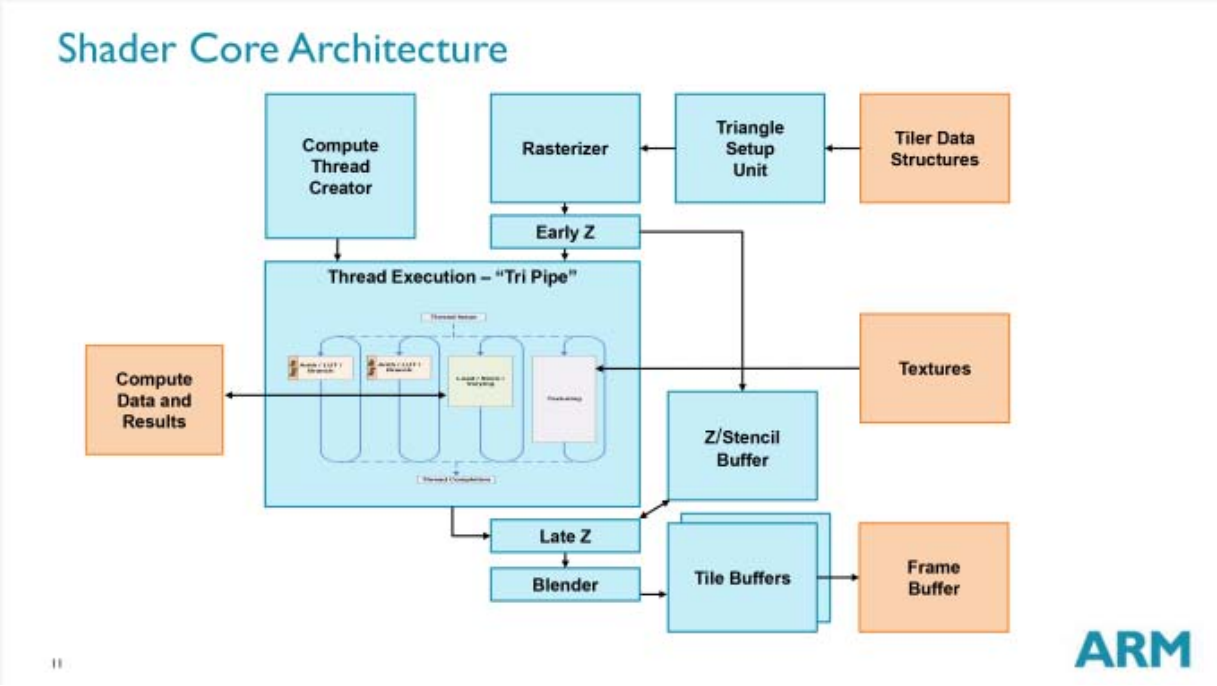
With stunning graphics capabilities, ARM Mali High Performance GPUs combine GPU Compute functionality with micro-architecture enhancements and system-wide, bandwidth-saving technology to bring energy efficiency to advanced mobile and consumer devices. GPU Compute solutions enable each task to be executed on the most suitable processor within the system. The resultant efficiencies guarantee superior graphics performance and extended battery life.

High performance GPUs:

- Mali-G71
- Mali-T860 & T880
- Mali-T760
- Mali-T628
- Mali-T624

See http://www.arm.com/products/graphics-and-multimedia/mali-gpu.

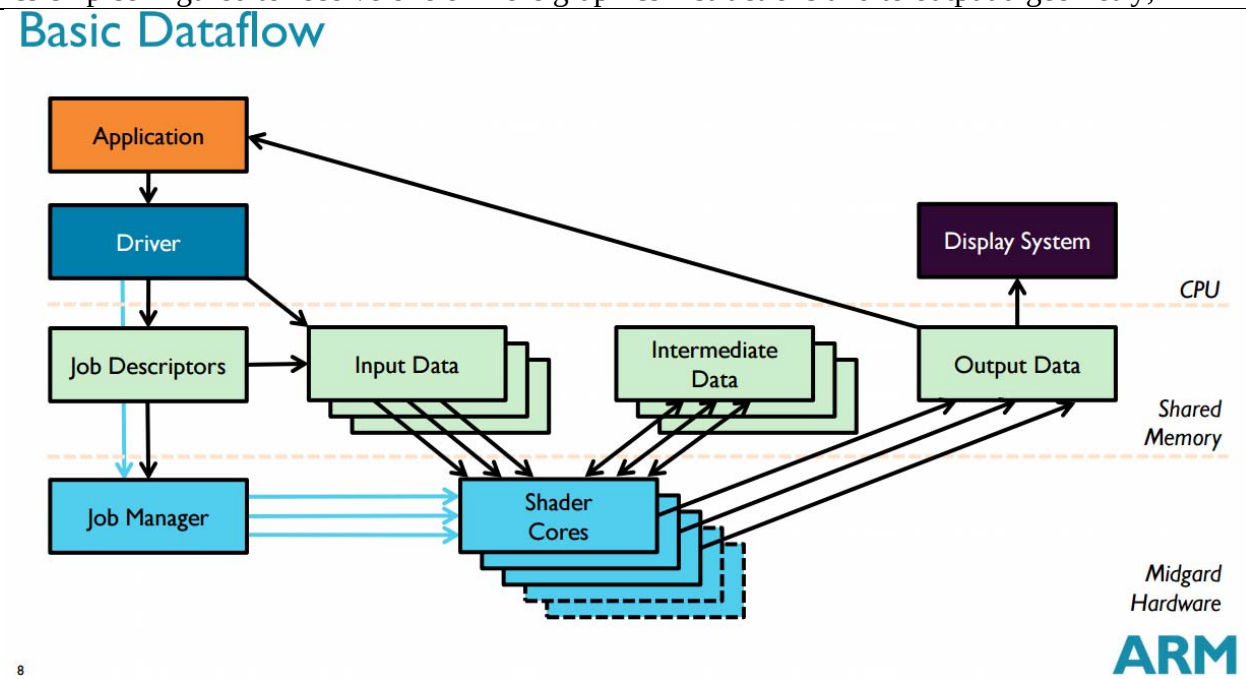
"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;	The LG Products include a front-end in the graphics chip configured to receive one or more graphics instructions and to output geometry. For example, as depicted below, the ARM Mali GPU includes, *inter alia*, tri-pipe shader core ("Shader"), triangle setup unit ("TSU"), rasterizer ("Rasterizer"), early-Z ("Early-Z"), z/stencil ("Z/Stencil), tile buffer ("Tile Buffer"), and late z ("Late Z") circuitry. Moreover, each shader includes a texturing unit ("TA"), load store unit ("LSU"), and arithmetic pipeline ("ALU") circuitry. **The Midgard Architecture** Diving in to the Mali architecture, we'll start with a high level overview of the architecture. What we're looking at here is a single Midgard shader core, which despite the "shader" name actually contains a whole lot more. A shader core in this context contains the actual shader core within one of Midgard's "tri pipe" shader blocks, but also contains a triangle setup unit, rasterizer, Z & stencil hardware, a ROP/blender, tiling hardware, and a compute thread creator specifically for feeding a tri pipe with compute workloads.  *See* http://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/4. The Mali GPU includes a front-end configured to receive one or more graphics instructions. For example,

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

	"Mali GPUs use data structures and hardware functional blocks[.]" Moreover, "shaders specify the vertex and fragment processing operations." **1.6.1 About the OpenGL ES 3.x pipeline** Mali GPUs use data structures and hardware functional blocks to implement the OpenGL ES graphics pipeline. In the OpenGL ES 3.x pipeline, the shaders specify the vertex and fragment processing operations. The application must provide a pair of shaders for each draw call. A vertex shader defines the vertex processing operations and a fragment shader defines the fragment processing operations. The vertex shader is executed once per vertex, and the fragment shader is executed once per fragment. Most of the semantics that are associated with data flowing through the pipeline are abstracted into the following special variables that are declared in the shaders: • Generic vertex attributes. Generic vertex attributes replace all vertex data, such as position, normal vector, texture coordinates, and colors. • Varying variables. All outputs from the vertex shader, except for position and point size, are abstracted into varying variables. These variables are interpolated across the primitive and are available to the fragment shader. • Uniform variables. All global states that are required by vertex and fragment processing, such as transformation matrices, light positions, material properties, texture stage constants, and texture bindings, are abstracted into uniform variables. The application sets the values of these variables. The following figure shows a simplified OpenGL ES graphics pipeline: *See* http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf.

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"



See http://malideveloper.arm.com/downloads/ARM_Game_Developer_Days/PDFs/2-Mali-GPU-architecture-overview-and-tile-local-storage.pdf.

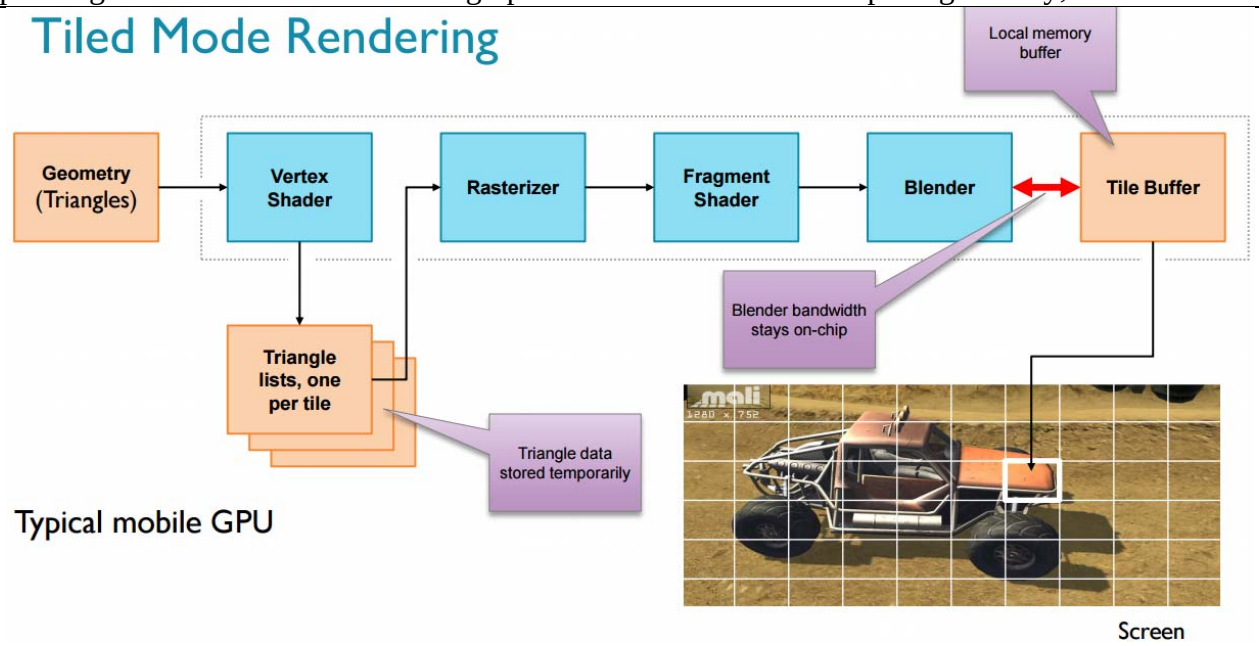
The front-end outputs a geometry. For example, "[t]he Mali GPU generates primitives starting from the vertices." Moreover, the "vertex processor…[a]ssembles vertices of graphics primitives" and "[b]uilds polygon lists." Furthermore, "[t]he output of vertex processing includes…[t]he position of the vertex in the output frambuffer" and "[a]dditional data, such as the color of the vertex after lighting calculations."

U.S. Patent No. 7,633,506: Claim 1

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

	1.6.2 Primitive assembly The Mali GPU generates primitives starting from the vertices. A point contains one vertex, a line contains two vertices, a triangle contains three vertices. Vertices can be shared between multiple primitives, depending on the draw mode. If geometry or tessellation shaders are present, then vertices can generate a variable number of primitives. **1.6.3 Vertex processing** The vertex data provided by the application is read one vertex at a time, and the shader core runs a vertex shader program for each vertex. This shader program performs: • Lighting. • Transforms. • Viewport transformation. • Perspective transformation. The shader core or vertex processor also perform the following processing: • Assembles vertices of graphics primitives. • Builds polygon lists. The output from vertex processing includes: • The position of the vertex in the output framebuffer. • Additional data, such as the color of the vertex after lighting calculations. *See* http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf.

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

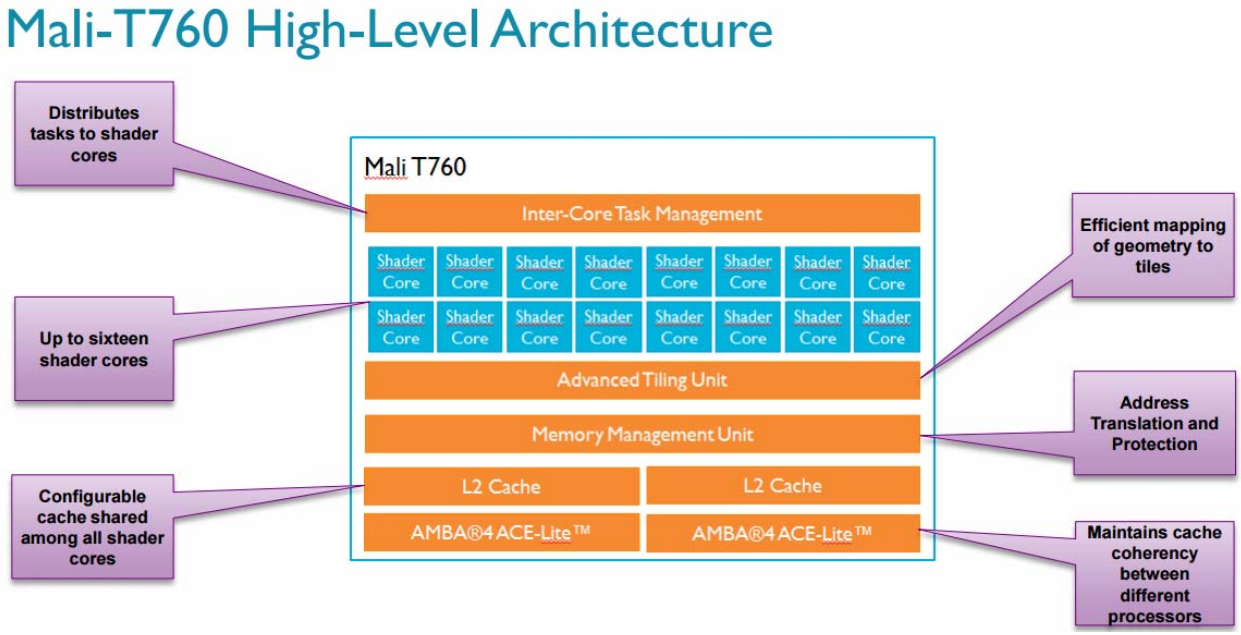


See ARM, Midgard GPU Architecture, p.7, available at http://malideveloper.arm.com/downloads/ARM_Game_Developer_Days/PDFs/2-Mali-GPU-architecture-overview-and-tile-local-storage.pdf.

Furthermore, the Mali GPUs include an Inter-Core Task Management module that distributes tasks to shader cores.

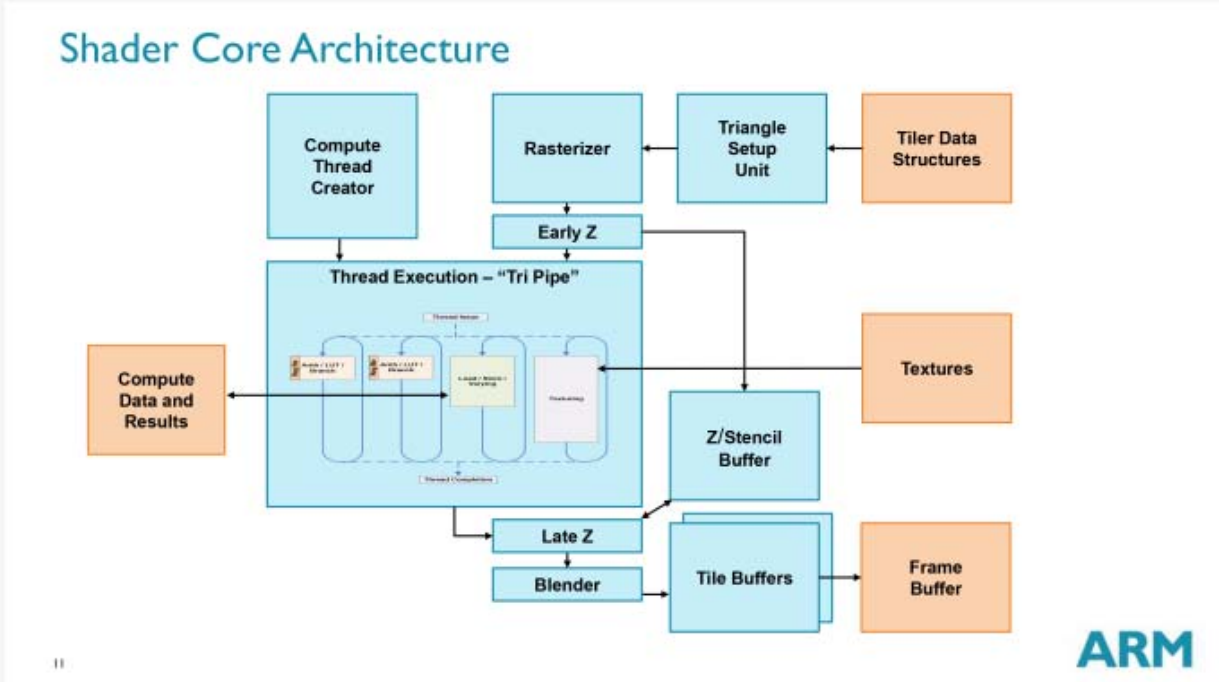
U.S. Patent No. 7,633,506: Claim 1

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"



See ARM, Midgard GPU Architecture, p.4, available at http://malideveloper.arm.com/downloads/ARM_Game_Developer_Days/PDFs/2-Mali-GPU-architecture-overview-and-tile-local-storage.pdf.

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;	The LG Products include a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer. For example, the Mali GPU include the Shader, TSU, Rasterizer, Early-Z, Z/Stencil, Tile, Late Z stages (collectively, the "Pixel Processing Stage"). The Midgard Architecture Diving in to the Mali architecture, we'll start with a high level overview of the architecture. What we're looking at here is a single Midgard shader core, which despite the "shader" name actually contains a whole lot more. A shader core in this context contains the actual shader core within one of Midgard's "tri pipe" shader blocks, but also contains a triangle setup unit, rasterizer, Z & stencil hardware, a ROP/blender, tiling hardware, and a compute thread creator specifically for feeding a tri pipe with compute workloads.  *See* http://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/4.

U.S. Patent No. 7,633,506: Claim 1

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

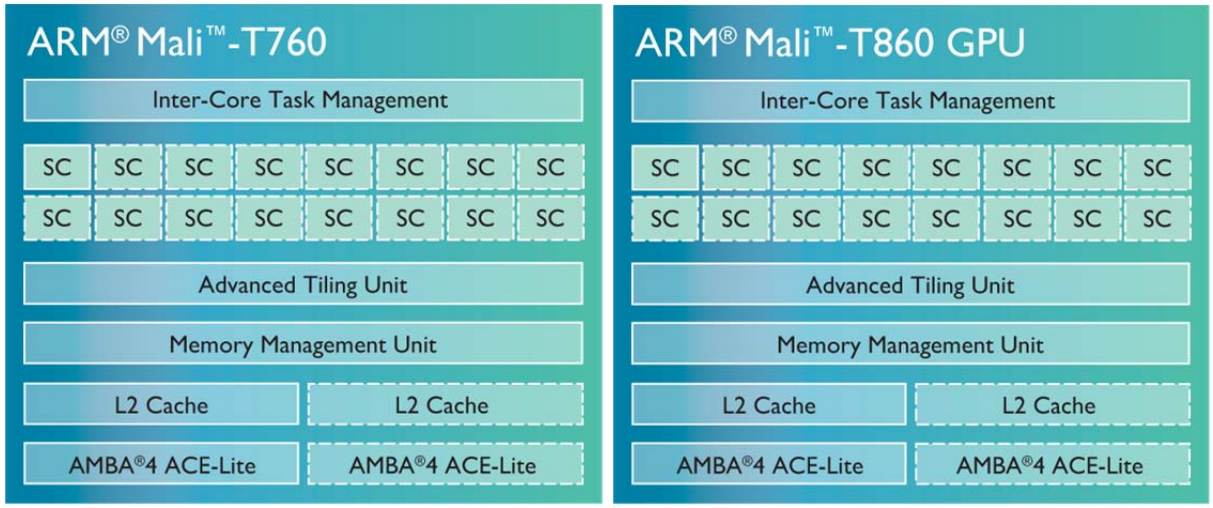
	The Mali GPU includes a back-end in the graphics chip configured to receive said geometry and to process said geometry. For example, the Pixel Processing Stage includes the TSU and the Rasterizer. The TSU provides "coefficients" to the rasterizer, which "applies equations to create fragments." Moreover, the Rasterizer includes circuitry in which "[e]ach primitive is divided into fragments so that there is one or more fragments for each pixel covered by the primitive." **1.6.4 Rasterization** Each primitive is divided into fragments so that there is one or more fragments for each pixel covered by the primitive. **Reads data** Reads the state information, polygon lists, and transformed vertex data. These are processed in a triangle setup unit to generate coefficients. **Rasterizes polygons** The rasterizer takes the coefficients from the triangle setup unit and applies equations to create fragments. **Interpolation** The properties at individual vertices are interpolated for each fragment produced by rasterization. *See* http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf. The geometry is processing into one or more final pixels to be placed in a frame buffer. For example, the Pixel Processing Stage processes the fragments, and ultimately "[t]he resulting pixel color is placed into the corresponding location in the frambuffer."

U.S. Patent No. 7,633,506: Claim 1

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

	1.6.5 Fragment processing Each fragment is assigned a color. This stage usually involves one or more texture look-ups. The fragment shader defines the fragment processing operations and it is executed once per fragment. **1.6.6 Framebuffer operations** The resulting pixel color is placed into the corresponding location in the framebuffer. Depending on the render states set for the draw call, the fragment is subjected to a series of tests and combination operations on route to the location. The tests include: • Depth testing. Compares the new fragment depth value to the depth value of the fragment that is previously written to this pixel, and discards it if it fails the depth test comparison. • Blending. Calculates the resulting pixel color as a combination of the fragment color and the existing pixel color. • Scissoring. Restricts rendering to a certain area of the framebuffer, and discards the fragment if it is located outside that area. *See* http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf.

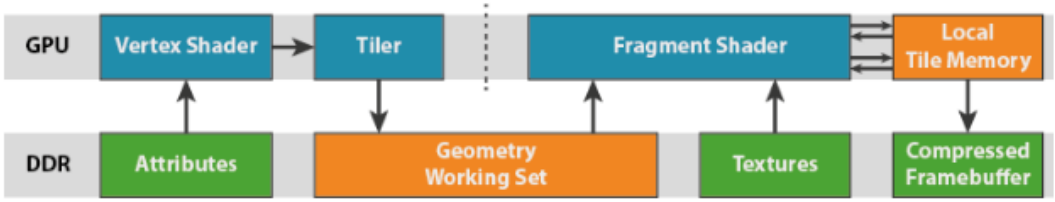
"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"

wherein said back-end in the graphics chip comprises multiple parallel pipelines;	The LG Products include a back-end in the graphics chip that comprises multiple parallel pipelines.
	
	For example, as depicted below, the Mali GPU includes multiple parallel SCs.
	
	 
	
	See http://www.arm.com/products/multimedia/mali-gpu/high-performance/mali-t760.php; https://www.arm.com/products/multimedia/mali-gpu/high-performance/mali-t860-t880.php.
	
	The Mali Approach
	
	The Mali GPU family takes a very different approach, commonly called tile-based rendering, designed to minimize the amount of power hungry external memory accesses which are needed during rendering. As described in the first blog in this series, Mali uses a distinct two-pass rendering algorithm for each render target. It first executes all of the geometry processing, and then executes all of the fragment processing. During the geometry processing stage, Mali GPUs break up the screen into small 16x16 pixel tiles and construct a list of which rendering primitives are present in each tile. When the GPU fragment shading step runs, each shader core processes one 16x16 pixel tile at a time, rendering it to completion before starting the next one. For tile-based architectures the algorithm equates to:
	
	`foreach( tile )`
	`  foreach( primitive in tile )`
	`  foreach( fragment in primitive in tile )`
	`  render fragment`

"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"

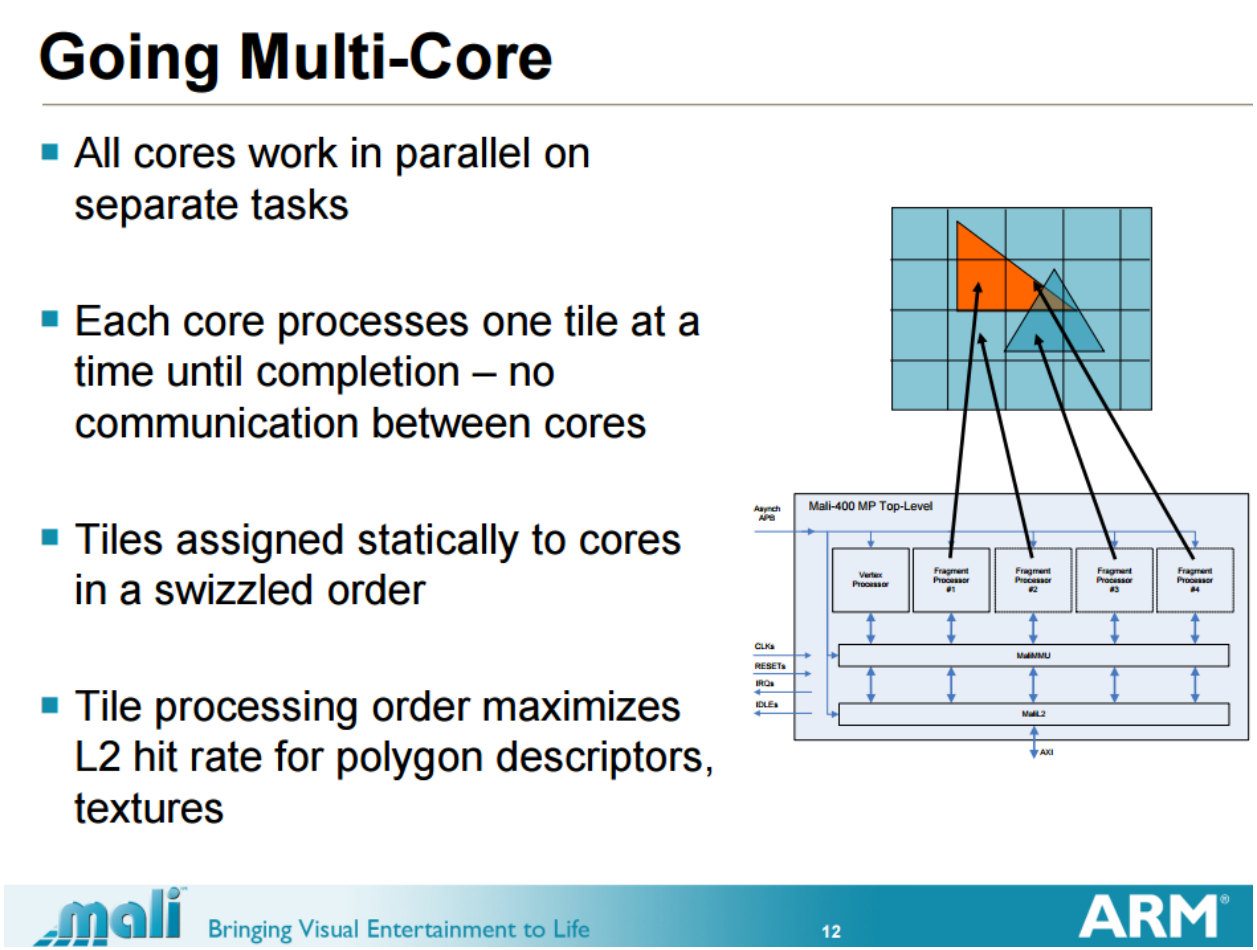
	See The Mali GPU: An Abstract Machine, Part 2 - Tile-based Rendering, *available at* https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.

"wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and"

wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and	The geometry is determined to locate in a portion of an output screen defined by a tile. For example, the Mali GPUs implement tiled-based rendering. On information and belief, the tile based rendering of the Mali GPU is substantially similar to the Mali-400 GPU. **The Mali Approach** The Mali GPU family takes a very different approach, commonly called tile-based rendering, designed to minimize the amount of power hungry external memory accesses which are needed during rendering. As described in the first blog in this series, Mali uses a distinct two-pass rendering algorithm for each render target. It first executes all of the geometry processing, and then executes all of the fragment processing. During the geometry processing stage, Mali GPUs break up the screen into small 16x16 pixel tiles and construct a list of which rendering primitives are present in each tile. When the GPU fragment shading step runs, each shader core processes one 16x16 pixel tile at a time, rendering it to completion before starting the next one. For tile-based architectures the algorithm equates to: 01. foreach(tile) 02. foreach(primitive in tile) 03. foreach(fragment in primitive in tile) 04. render fragment 05. As a 16x16 tile is only a small fraction of the total screen area it is possible to keep the entire working set (color, depth, and stencil) for a whole tile in a fast RAM which is tightly coupled with the GPU shader core. **Tile-based Renderer Data Flow**  *See* The Mali GPU: An Abstract Machine, Part 2 - Tile-based Rendering, *available at* https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.

"wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and"

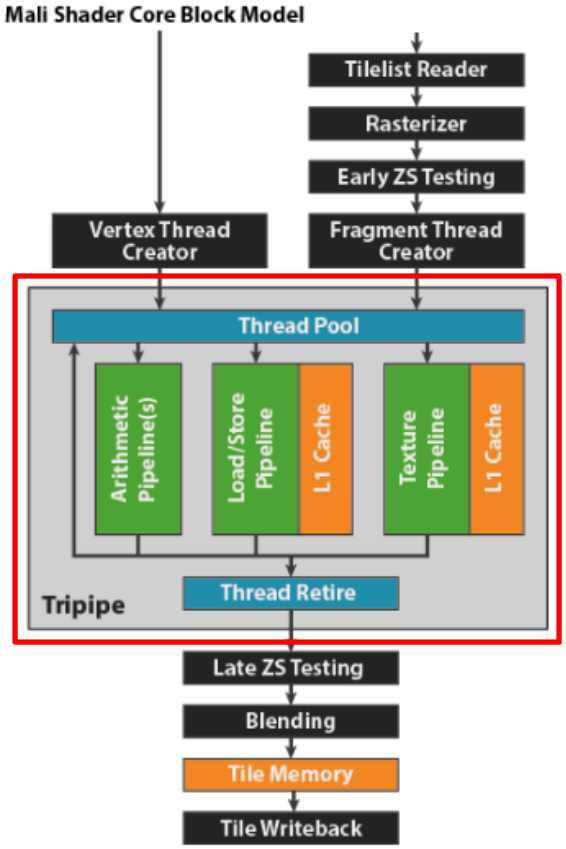
Furthermore, as depicted below, the geometry is determined to locate in a portion of an output screen defined by a tile.



See http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf.

U.S. Patent No. 7,633,506: Claim 1

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading.	Each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading. For example, each Shader includes a programmable Tri-Pipe, which includes the TA, LSU, and ALU. **The Midgard Shader Core** The Mali shader core is structured as a number of fixed-function hardware blocks wrapped around a programmable "tripipe" execution core. The fixed function units perform the setup for a shader operation - such as rasterizing triangles or performing depth testing - or handling the post-shader activities - such as blending, or writing back a whole tile's worth of data at the end of rendering. The tripipe itself is the programmable part responsible for the execution of shader programs.  *See* The Mali GPU: An Abstract Machine, Part 3 - The Midgard Shader Core, https://community.arm.com/groups/arm-mali-graphics/blog/2014/03/12/the-mali-gpu-an-abstract-machine-part-3--the-shader-core.

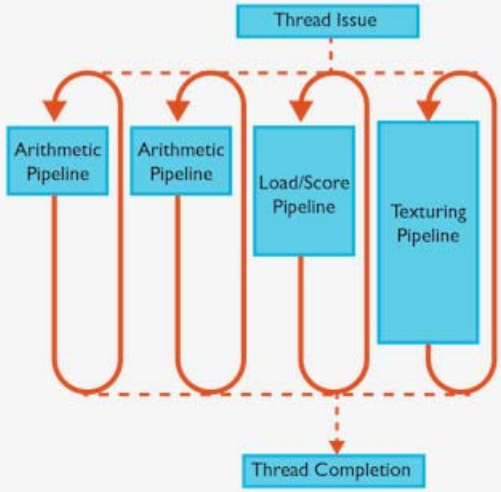
U.S. Patent No. 7,633,506: Claim 1

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

	The ALU is designed to “strike a closer balance between shading and texturing.”

U.S. Patent No. 7,633,506: Claim 1

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."



As we've stated before, for our purposes we're primarily looking at the Mali-T760. On the T760 ARM uses 2 ALU blocks per tri pipe, which is the most common configuration that you will see for Midgard. However ARM also has Midgard designs that have 1 ALU block or 4 ALU blocks per tri pipe, which is one of the reasons why seemingly similarly GPUs such as T760, T720, and T678 can look so similar and yet behave so differently.

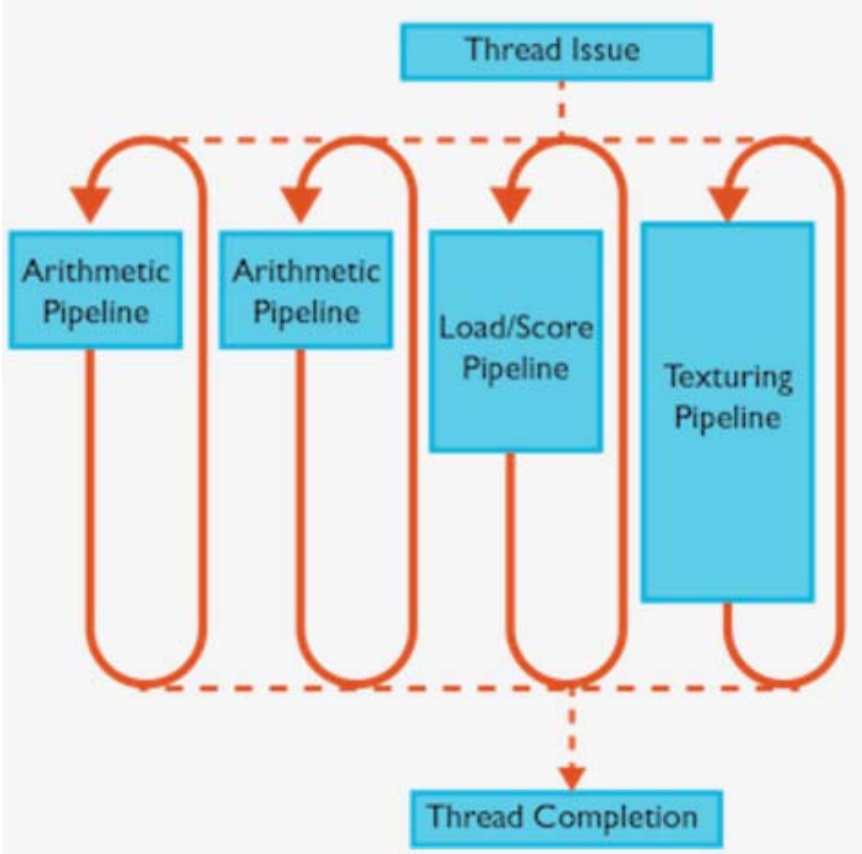
ARM Mali Midgard Arithmetic Pipeline Count (Per Core)	
T628	2
T678	4
T720	1
T760	2

Without being fully exhaustive, among various Midgard designs T628 and T760 are 2 ALU designs, while T720 is a 1 ALU design, and T678 is a 4 ALU design.

As one would expect, the different number of arithmetic pipelines per tri pipe has a knock-on effect on performance in all aspects, due to the changing ratio between the number of arithmetic pipelines and the number of load/store units and texture units. T678, for example, would be fairly shader-heavy, whereas the 2 ALU designs strike a closer balance between shading and texturing. Among the various Midgard designs ARM has experimented with several configurations, and with the T700 series they have settled on 2 ALU designs for the high-end T760 and 1 ALU for the mid-range T720 (although ARM likes to point out that T720 has some further optimizations just for this 1 ALU configuration).

See http://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/4.

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

	In particular, the shading operations comprise texture operations. For example, the SC includes a texture pipeline, load/store pipeline and a plurality of arithmetic pipelines. The "texture pipeline (T-pipe) is responsible for all memory access to do with textures. The texture pipeline can return one bilinear filtered texel per clock; trilinear filtering requires us to load samples from two different mipmaps in memory." *See* https://community.arm.com/groups/arm-mali-graphics/blog/2014/03/12/the-mali-gpu-an-abstract-machine-part-3--the-shader-core.  *See* http://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/4.

U.S. Patent No. 7,633,506: Claim 1

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

7.2.2 About Mali GPU architectures

Mali GPUs use a SIMD architecture. Instructions operate on multiple data elements simultaneously.

The peak throughput depends on the hardware implementation of the Mali GPU type and configuration.

The Mali GPUs contain 1 to 16 identical shader cores. Each shader core supports up to 384 concurrently executing threads.

Each shader core contains:

- One to four arithmetic pipelines.
- One load-store pipeline.
- One texture pipeline.

See "ARM Guide to Unity" Version 2.1 page 7-64 available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf (accessed 10/27/2016).

The arithmetic pipeline ("ALU") performs texture operations. For example, the "ALU pipeline can read/write to 32 128-bit registers" including "texture pipeline results" from the texture pipe.

Registers

The ALU pipeline can read/write to 32 128-bit registers, which can be divided into 4 32-bit (highp in GLSL) components (one vec4) or 8 16-bit (mediump) components (two vec4's). Some of the registers, however, are dedicated to special purposes (see below) and are read-only or write-only.

Special Registers

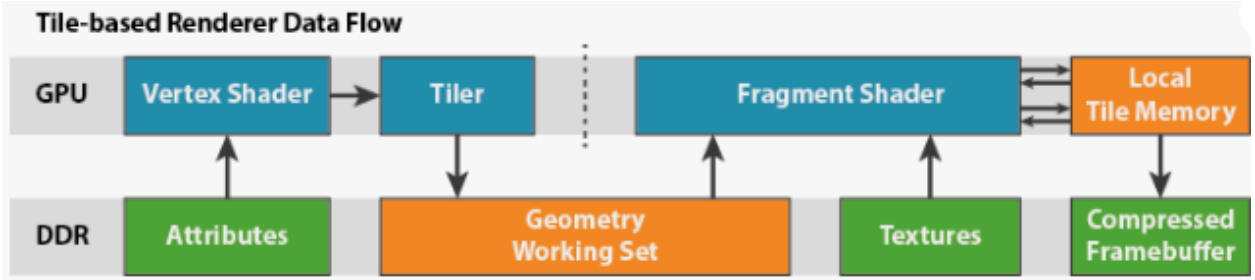
```
r24 - can mean "unused" for 1-src instructions, or a pipeline register
r26 - inline constant
r27 - load/store offset when used as output register
r28-r29 - texture pipeline results
r31.w - conditional select input when written to in scalar add ALU
```

r0 - r23 is divided into two spaces: work registers and uniform registers. A configurable number of registers can be devoted to each; if there are N uniform registers, then r0 - r(23-N) are work registers and r(24-N)-r23 are uniform registers.

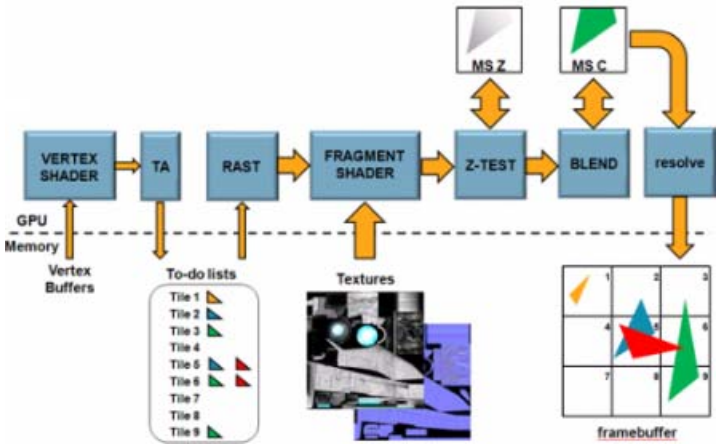
See http://limadriver.org/T6xx+ISA/.

The ALU also performs color operations. For example, the "Mali [GPU] only has to write the color data for a single tile back to memory at the end of the tile." *See* https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."



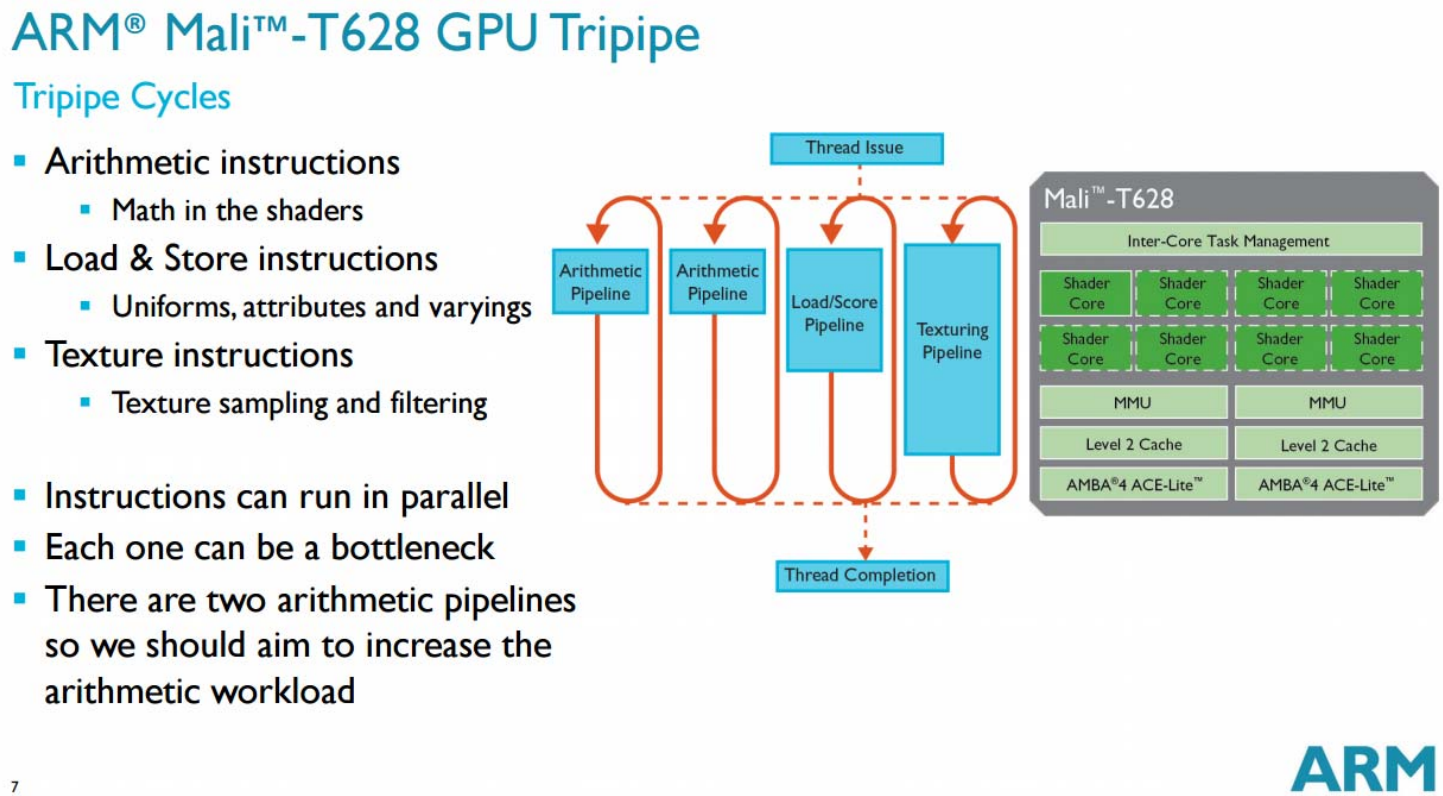
See https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.



See https://community.arm.com/groups/arm-mali-graphics/blog/2012/08/17/how-low-can-you-go-building-low-power-low-bandwidth-arm-mali-gpus.

Moreover, the ALU is responsible for performing the "[m]ath in the shaders[.]"

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."



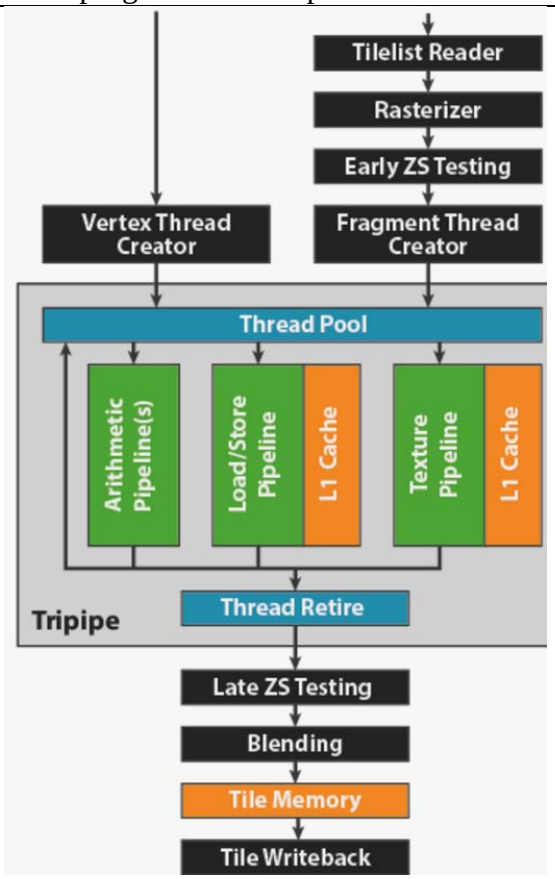
7

See http://malideveloper.arm.com/downloads/GDC14/Weds/11.15amStreamlineMaliHWCounters.pdf.

Additionally, "there are three classes of execution pipeline in the tripipe design: one handling arithmetic operations, one handling memory load/store and varying access, and one handling texture access. There is one load/store and one texture pipe per shader core, but the number of arithmetic pipelines can vary depending on which GPU you are using; most silicon shipping today will have two arithmetic pipelines." *See* https://community.arm.com/groups/arm-mali-graphics/blog/2014/03/12/the-mali-gpu-an-abstract-machine-part-3--the-shader-core.

See http://malideveloper.arm.com/downloads/ARM_Game_Developer_Days/PDFs/2-Mali-GPU-architecture-overview-and-tile-local-storage.pdf.

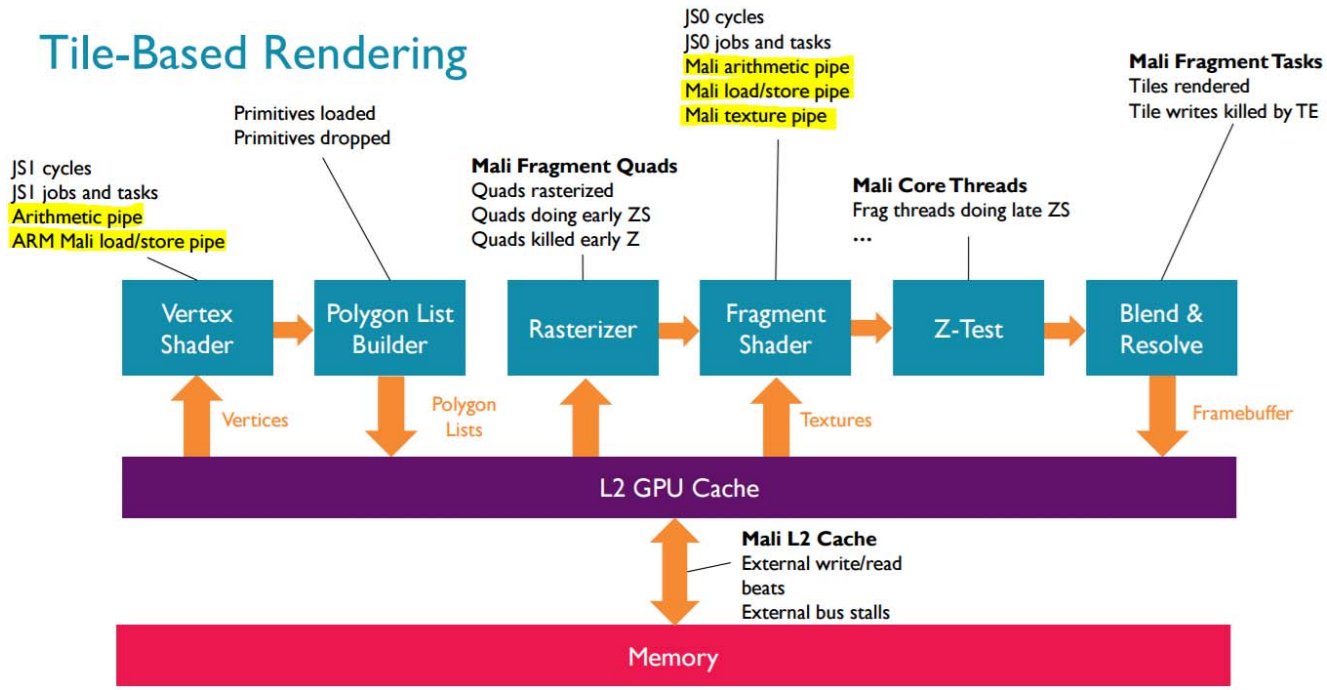
"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

	 *See* https://community.arm.com/groups/arm-mali-graphics/blog/2014/03/12/the-mali-gpu-an-abstract-machine-part-3--the-shader-core.

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

	1.1 About Mali GPUs ARM produces the following families of Mali GPUs: **Mali Midgard GPUs** Mali Midgard GPUs include the following: • Mali-T600 series. • Mali-T720. • Mali-T760. • Mali-T820. • Mali-T830. • Mali-T860. • Mali-T880. **Mali Utgard GPUs** The Mali Utgard GPUs include the following: • Mali-300. • Mali-400 MP. • Mali-450 MP. • Mali-470 MP. **Note** The Mali Utgard GPUs do not support OpenCL. Mali GPUs can have one or more shader cores. Each shader core contains one or more *Arithmetic Logic Units* (ALUs). The ALUs are based on a *Single Instruction Multiple Data* (SIMD) architecture. Instructions operate on multiple data elements simultaneously. Mali GPUs run data processing tasks in parallel that contain relatively little control code. Mali GPUs typically contain many more processing units than application processors. This enables Mali GPUs to compute at a higher rate than application processors, without using more power. *See* "ARM Guide to Unity" Version 2.1 page 1.1 available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf (accessed 10/27/2016).

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."



See ARM, How to Optimize Your Mobile Game with ARM Tools and Practical Examples, p.33, http://malideveloper.arm.com/downloads/GDC15/How%20to%20Optimize%20Your%20Mobile%20Game%20with%20ARM%20Tools%20and%20Practical%20Examples.pdf.