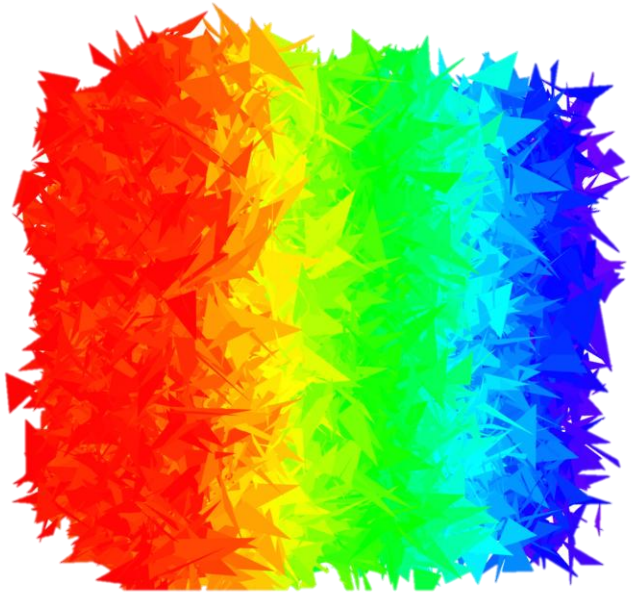
Performance Analysis and Optimization

ARM® Mali™ GPU Performance Counters in ARM DS-5 Streamline Performance Analyzer

Lorenzo Dal Col
Senior Software Engineer, ARM

Agenda

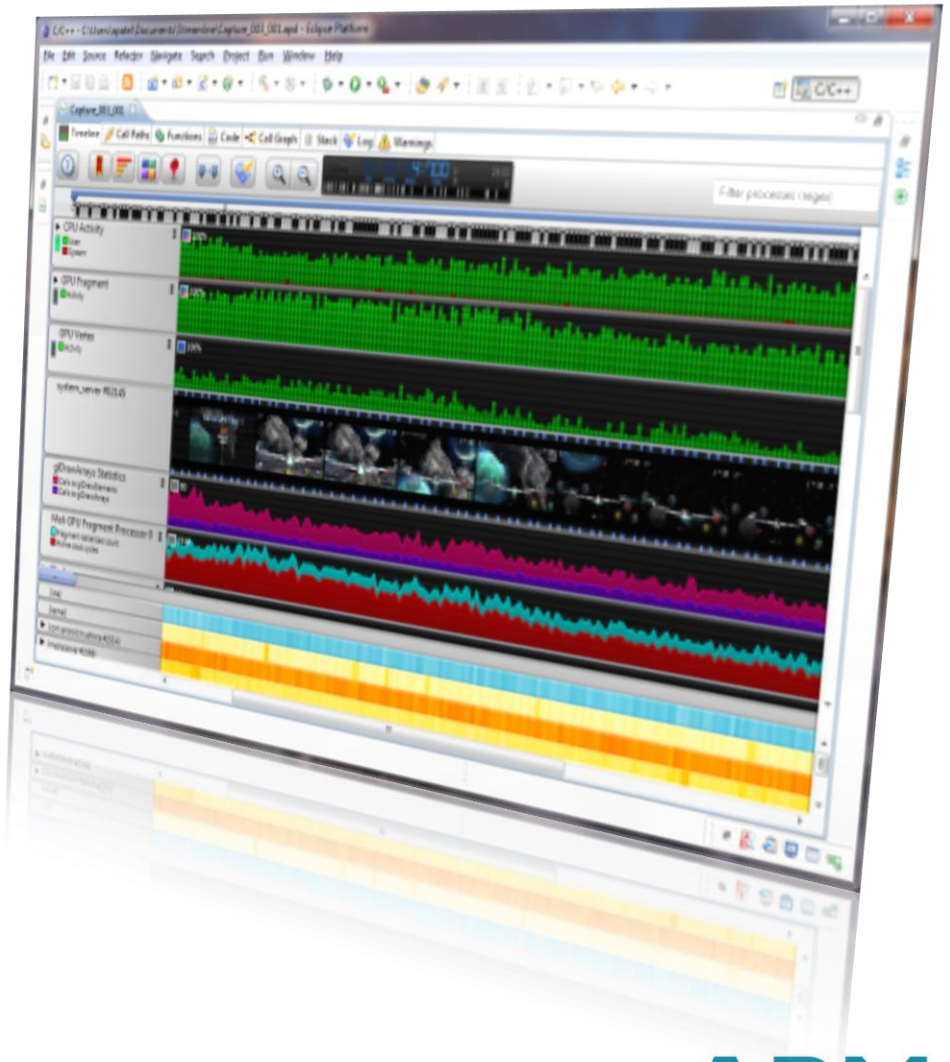
- Introduction to ARM® DS-5™ and Streamline™
- ARM Mali™ GPU Hardware Counters
- Sample Cases
  - CPU bound
  - Vertex bound
  - Fragment arithmetic bound
  - Fragment load & store bound
  - Bandwidth
- Q & A



ARM

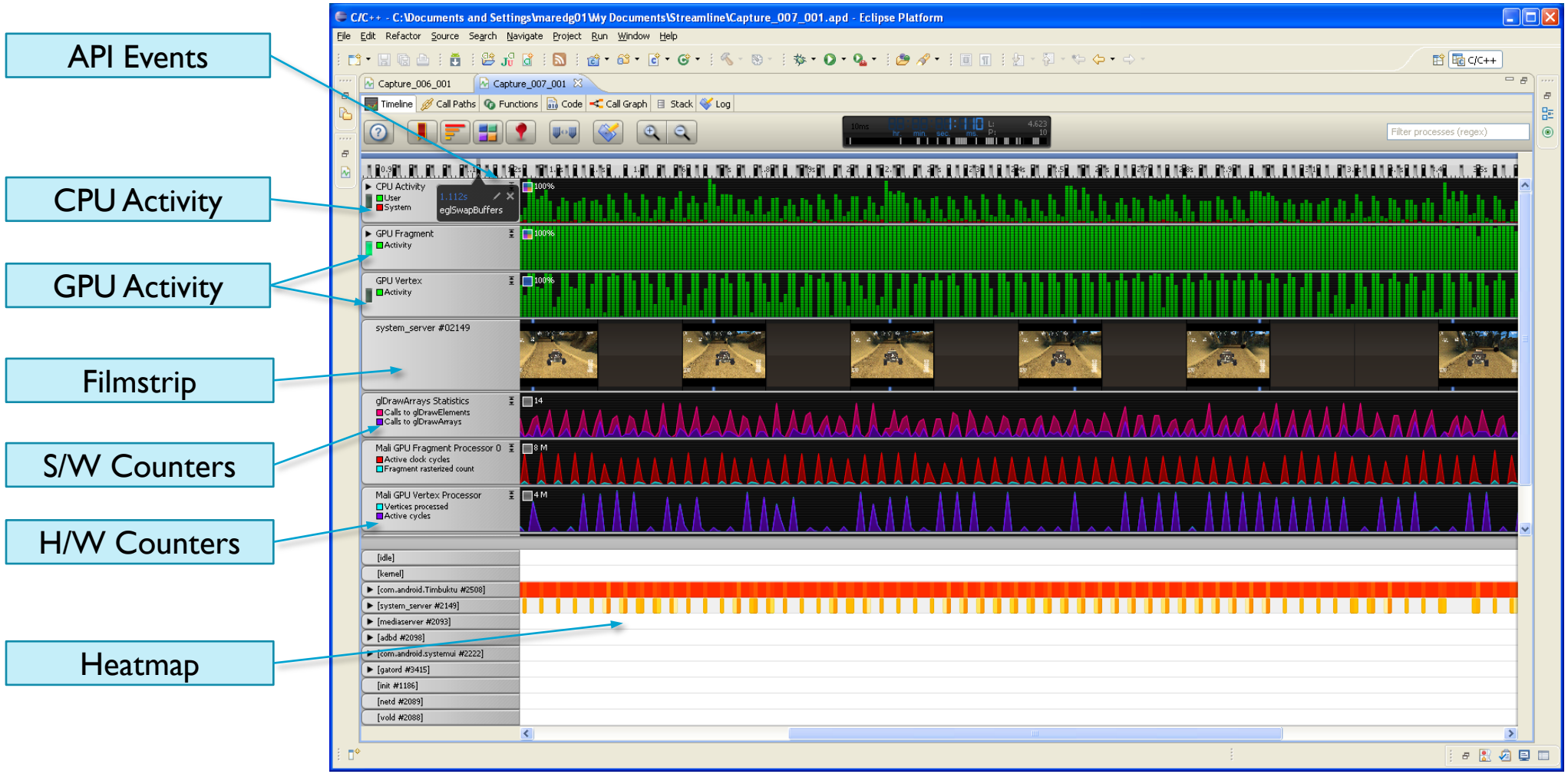
ARM® DS-5 Streamline Performance Analyzer

- System wide performance analysis
  - Simultaneous visibility across ARM Cortex® processors & ARM Mali™ GPUs
  - Support for graphics and GPU Compute performance analysis on High-End GPUs
  - Timeline profiling of hardware counters for detailed analysis
  - Custom counters
  - Per-core/thread/process granularity
  - Frame buffer capture and display
- DS-5 toolchain with support for ARM Mali GPUs
- Optimize performance and power efficiency of gaming applications across the system



ARM

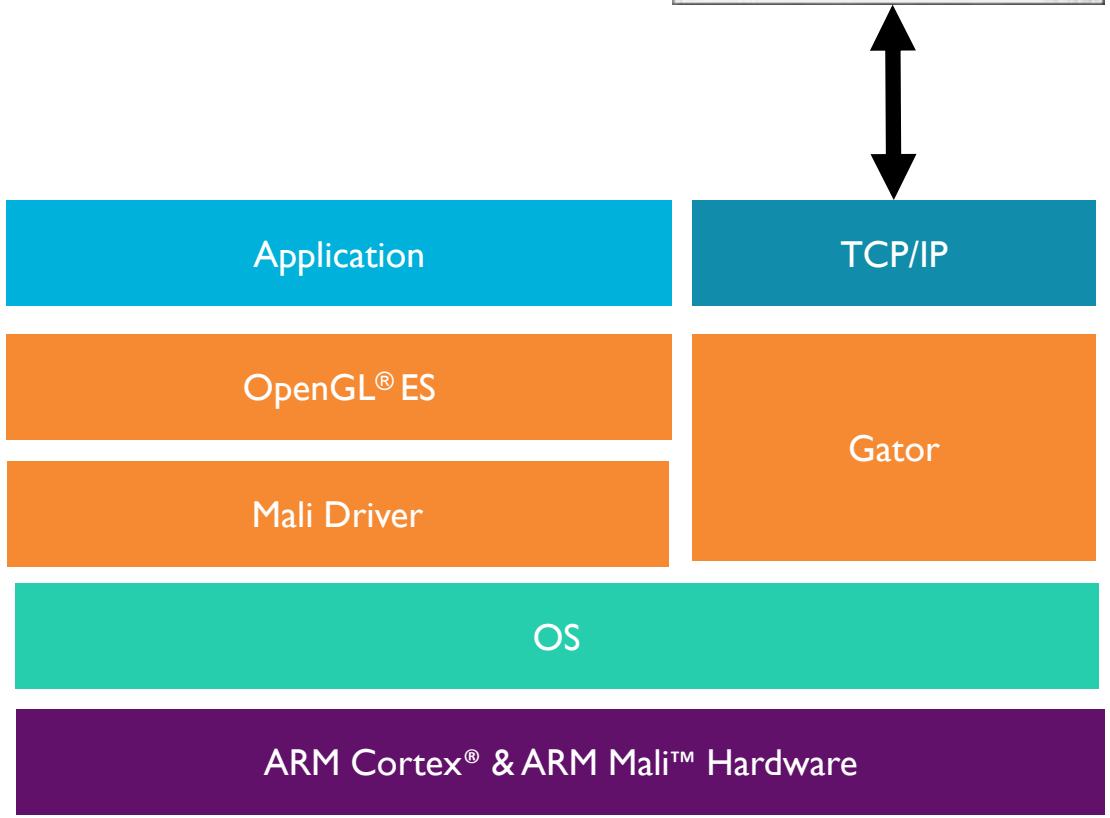
ARM® DS-5 Streamline Performance Analyzer



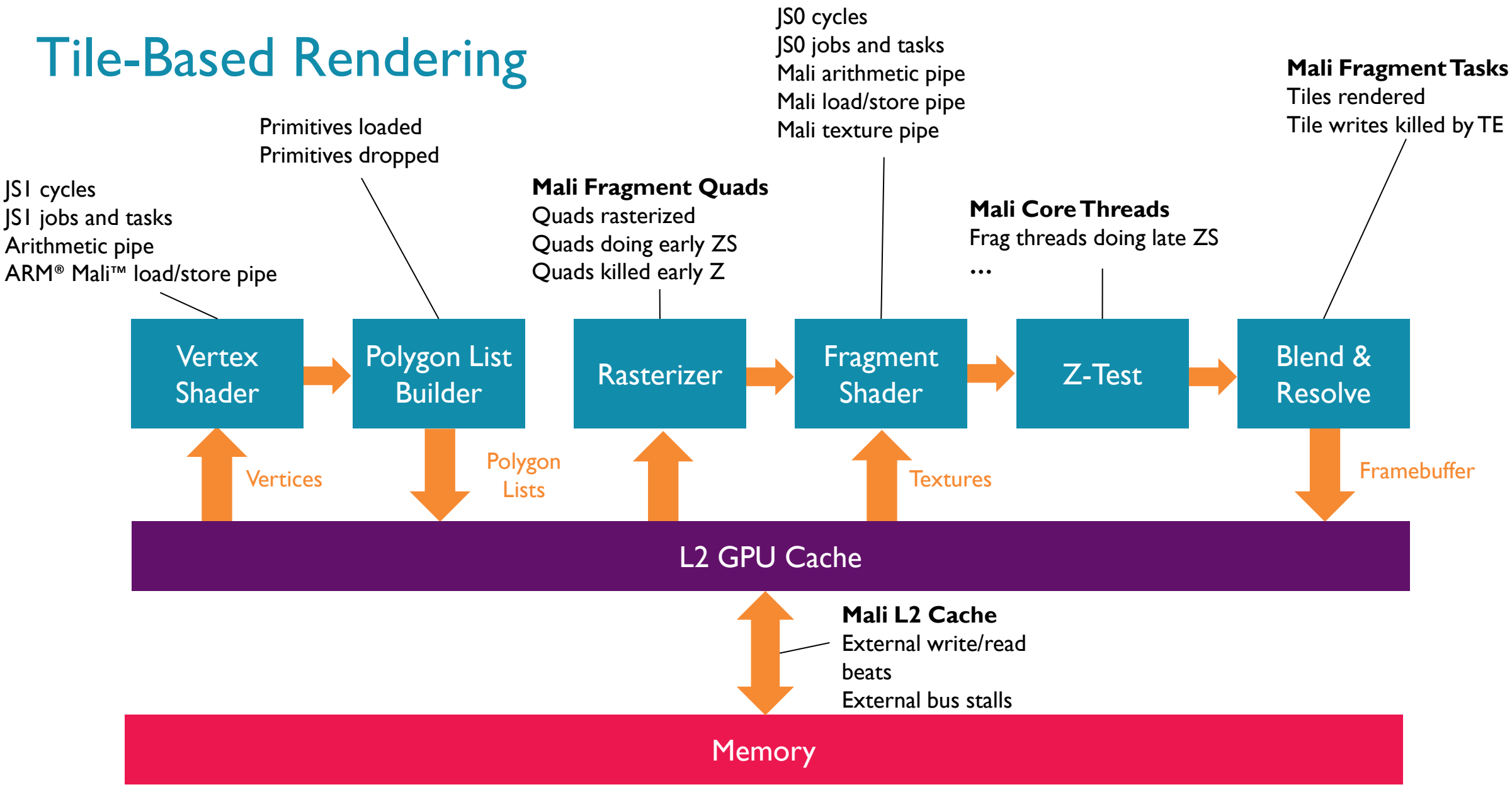
ARM® DS-5 Streamline Setup

- Gator interfaces with kernel & ARM Mali™ Drivers
  - Extract H/W & S/W counters
  - Extract frame buffer
  - Pass through events & annotations
  - Transmit data over TCP/IP to DS-5 tools
- Transparent to user application
  - Option to add annotations to user application for more advanced debugging
- Negligible performance impact
  - Zero impact when profiling disabled
  - Minimal impact in performance when enabled

ARM DS-5 Toolchain with Streamline



Tile-Based Rendering



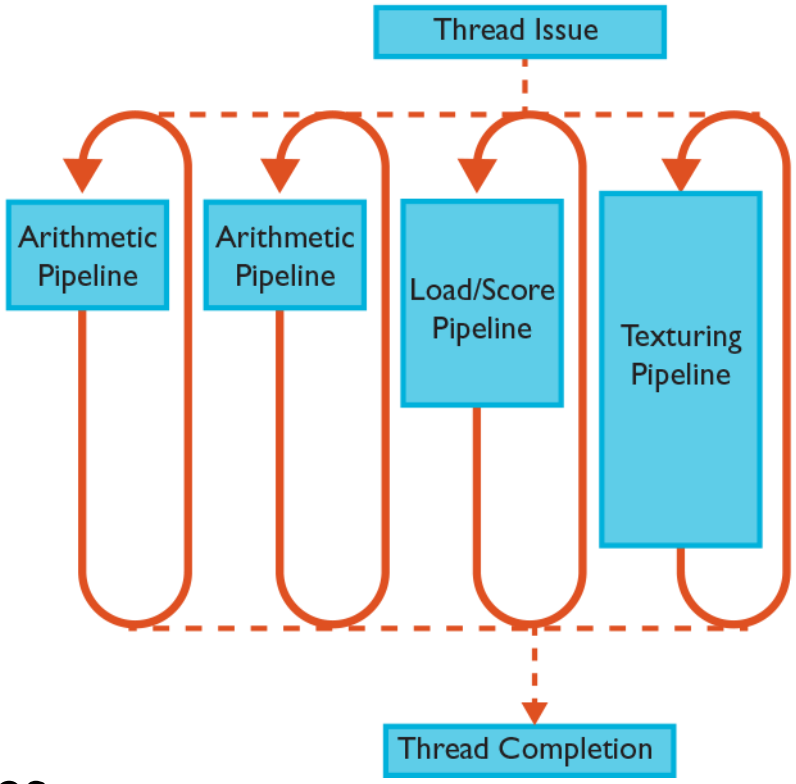
ARM

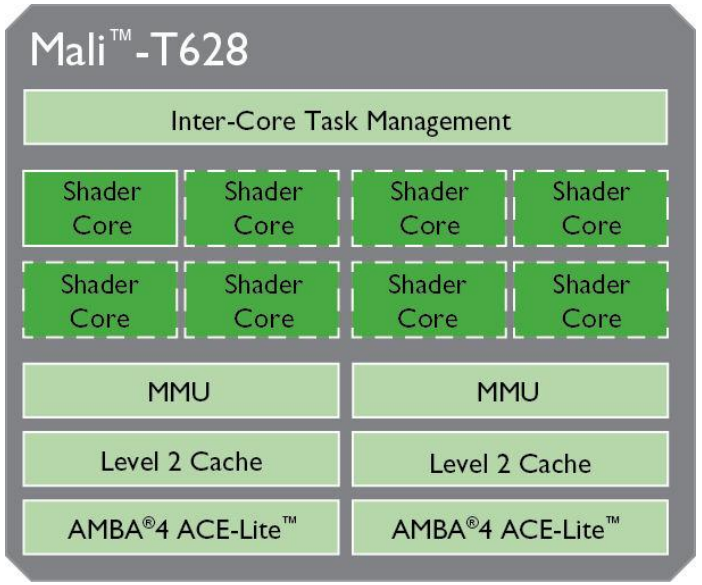
ARM® Mali™-T628 GPU Tripipe

Tripipe Cycles

- Arithmetic instructions
  - Math in the shaders
- Load & Store instructions
  - Uniforms, attributes and varyings
- Texture instructions
  - Texture sampling and filtering

- Instructions can run in parallel
- Each one can be a bottleneck
- There are two arithmetic pipelines so we should aim to increase the arithmetic workload





ARM

ARM® Mali™-T600 GPU Series Counters

Mali Job Manager Cycles	
GPU cycles	Number of cycles the GPU was active
IRQ cycles	Number of cycles the GPU had a pending interrupt
JS0 cycles	Number of cycles JS0 (fragment) was active
JS1 cycles	Number of cycles JS1 (vertex/tiler/compute) was active
JS2 cycles	Number of cycles JS2 (compute) was active
Mali Job Manager Work	
JS0 jobs	Number of Jobs (fragment) completed in JS0
JS0 tasks	Number of Tasks completed in JS0
JS1 jobs	Number of Jobs (vertex/tiler/compute) completed in JS1
JS1 tasks	Number of Tasks completed in JS1
JS2 jobs	Number of Jobs (compute) completed in JS2
JS2 tasks	Number of Tasks completed in JS2
Mali Core Cycles	
Tripipe cycles	Number of cycles the Tripipe was active
Fragment cycles	Number of cycles fragment processing was active
Compute cycles	Number of cycles vertex\compute processing was active
Fragment cycles waiting for tile	Number of cycles spent waiting for a physical tile buffer
Mali Arithmetic Pipe	
A instructions	Number of instructions completed by the A-pipe (normalized per pipeline)
Mali Load/Store Pipe	
LS instructions	Number of instructions completed by the LS-pipe
LS instruction issues	Number of instructions issued to the LS-pipe, including restarts
Mali Texture Pipe	
T instructions	Number of instructions completed by the T-pipe
T instruction issues	Number of instructions issued to the T-pipe, including restarts
Cache misses	Number of instructions in the T-pipe, recirculated because of cache miss

Mali Core Threads	
Fragment threads	Number of fragment threads started
Dummy fragment threads	Number of dummy fragment threads started
Compute threads	Number of vertex\compute threads started
Frag threads doing late ZS	Number of threads doing late ZS test
Frag threads killed late ZS	Number of threads killed by late ZS test
Mali Fragment Primitives	
Primitives loaded	Number of primitives loaded from tiler
Primitives dropped	Number of primitives dropped because out of tile
Mali Fragment Quads	
Quads rasterized	Number of quads rasterized
Quads doing early ZS	Number of quads doing early ZS test
Quads killed early Z	Number of quads killed by early ZS test
Mali Fragment Tasks	
Tiles rendered	Number of tiles rendered
Tile writes killed by TE	Number of tile writes skipped by transaction elimination
Mali Load/Store Cache	
Read hits	Number of read hits in the Load/Store cache
Read misses	Number of read misses in the Load/Store cache
Write hits	Number of write hits in the Load/Store cache
Write misses	Number of write misses in the Load/Store cache
Atomic hits	Number of atomic hits in the Load/Store cache
Atomic misses	Number of atomic misses in the Load/Store cache
Line fetches	Number of line fetches in the Load/Store cache
Dirty line evictions	Number of dirty line evictions in the Load/Store cache
Snoops in to LSC	Number of coherent memory snoops in to the Load/Store cache
Mali L2 Cache	
External write beats	Number of external bus write beats
External read beats	Number of external bus read beats
Cache read hits	Number of reads hitting in the L2 cache
Write hits	Number of writes hitting in the L2 cache
Write snoops	Number of write transaction snoops
Read snoops	Number of read transaction snoops
External bus stalls (AR)	Number of cycles a valid read address (AR) is stalled by the external interconnect
External bus stalls (W)	Number of cycles a valid write data (W channel) is stalled by the external interconnect

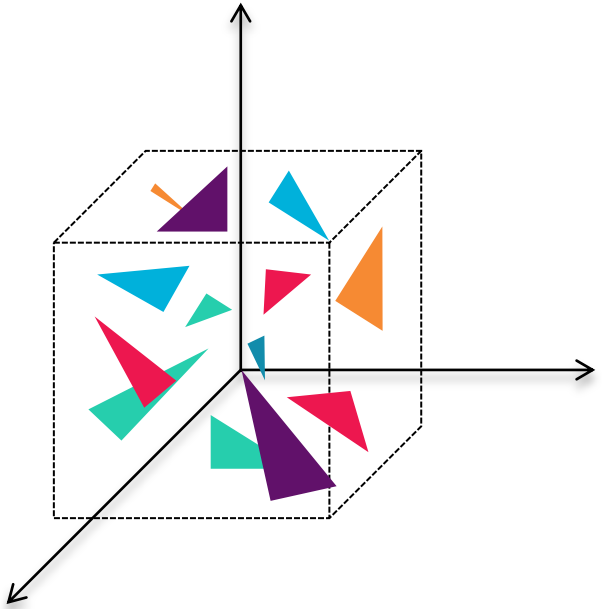
ARM

Sample Case

Study the Counters with the ComplexCube

- SimpleCube sample from the ARM® Mali™ OpenGL® ES SDK for Android™

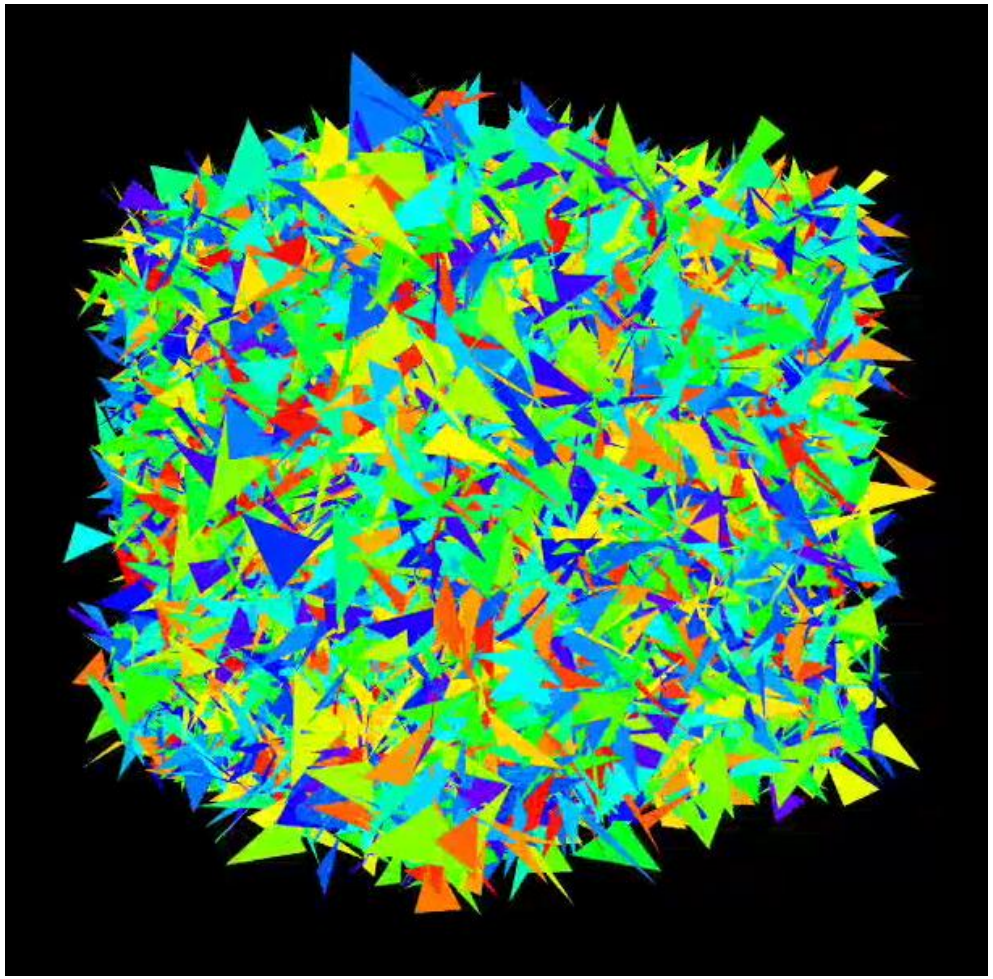
- Draw triangles randomly distributed in a cube space
- Space: -3.0 to +3.0 in x, y, z
- Size of the triangles from 0.0 to 0.25
- By tuning the app we can observe different behaviors



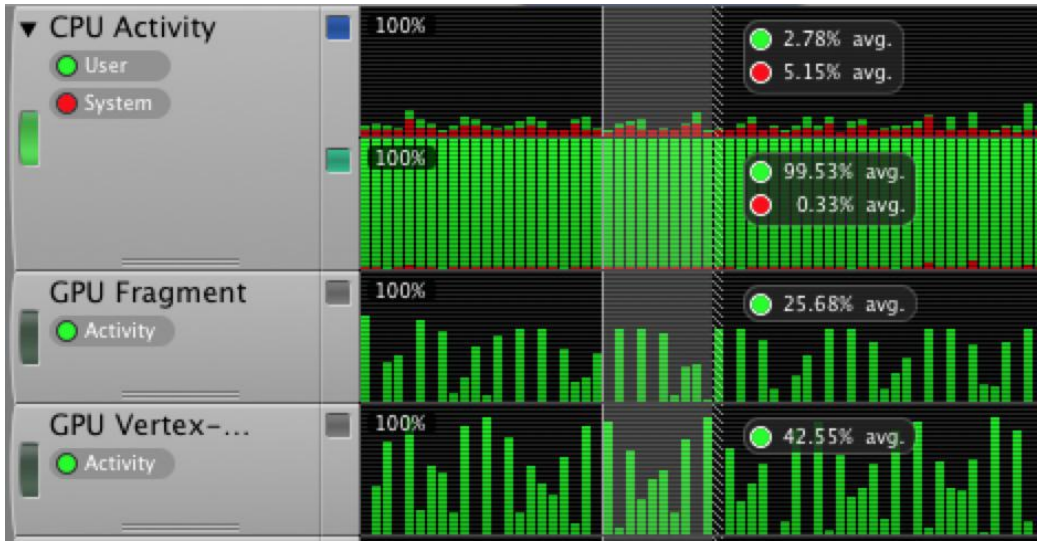
ARM

CPU Bound Case

Running ComplexCube with one draw call per triangle



- 20,000 triangles (60,000 vertices)
- One triangle for each draw call
  - 20,000 draw calls per frame
- Overhead for calling glDrawArrays once every three vertices



ARM

CPU Profiling

Find the Culprit in the Application Code

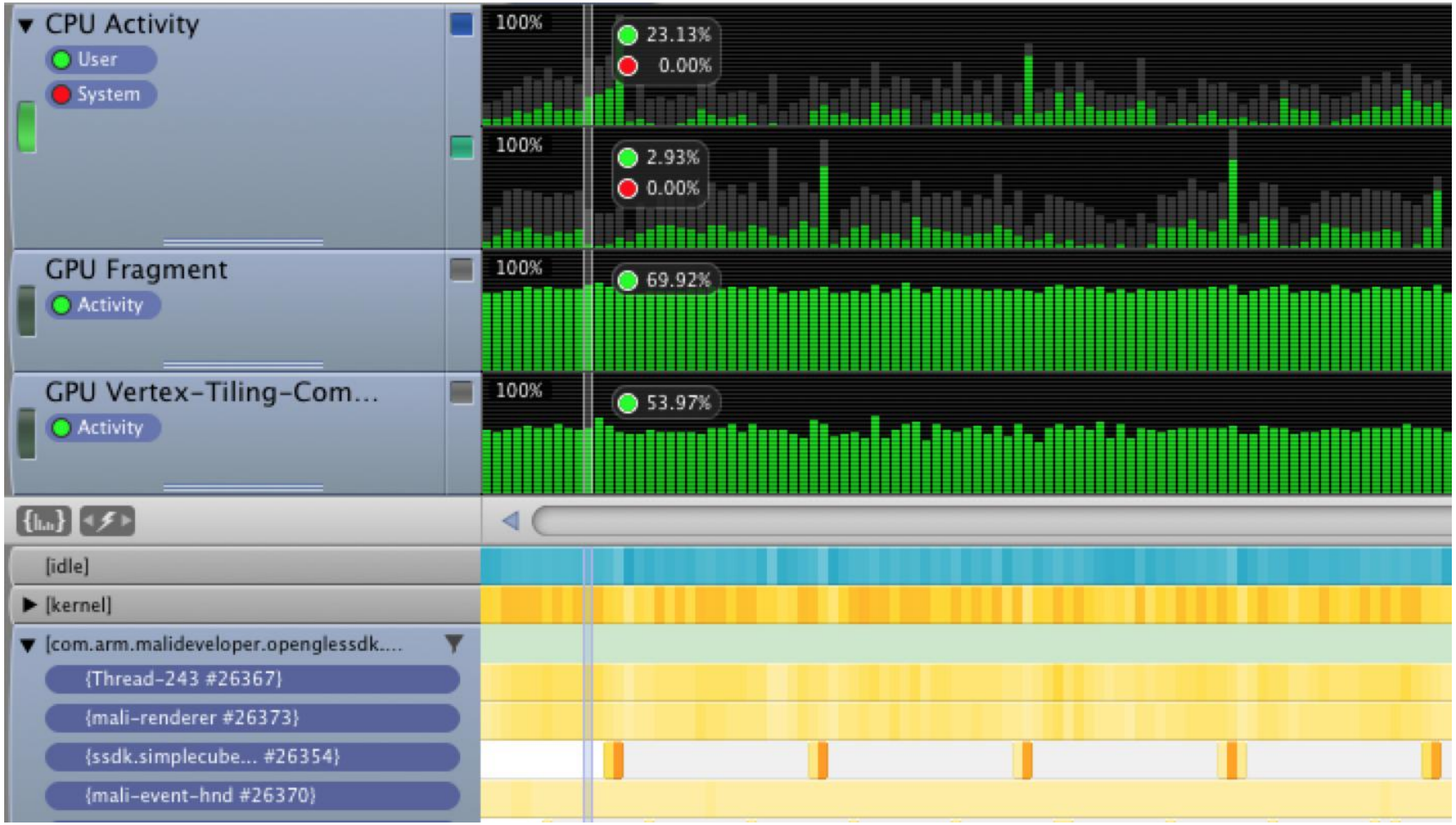
Self	% Self	Process	% Process	Total	Stack	[Process]/{Thread}/Code
		3,720	100.00%	48.92%	0	[com.arm.malideveloper.openglessdk.simplecube #2…
		3,583	96.32%	47.12%	0	{Thread-180 #25403}
2,217	59.60%	2,217	59.60%	29.16%	0	<unknown code in libGLES_mali.so>
577	15.51%	577	15.51%	7.59%	0	<unknown code in libc.so>
471	12.66%	471	12.66%	6.19%	0	<unknown code in kernel>
187	5.03%	187	5.03%	2.46%	0	<unknown code in libGLES_mgd.so>
69	1.85%	69	1.85%	0.91%	64	renderFrame()
53	1.42%	53	1.42%	0.70%	0	<unknown code in libGLESv2.so>
6	0.16%	6	0.16%	0.08%	0	.plt [libNative.so]
1	0.03%	1	0.03%	0.01%	0	<unknown code in libbinder.so>
1	0.03%	1	0.03%	0.01%	0	<unknown code in libdvm.so>
1	0.03%	1	0.03%	0.01%	0	<unknown code in libui.so>

14	20.29%	451	`for(int i = 0; i < nTriangles * 3; i += 9)`
		452	`{`
27	39.13%	453	`    glUniformMatrix4fv(projectionLocation, 1, GL_FALSE, projectionMatrix);`
19	27.54%	454	`    glUniformMatrix4fv(modelViewLocation, 1, GL_FALSE, modelViewMatrix);`
9	13.04%	455	`    glDrawArrays(GL_TRIANGLES, i, 9);`

ARM

Batch Draw Calls

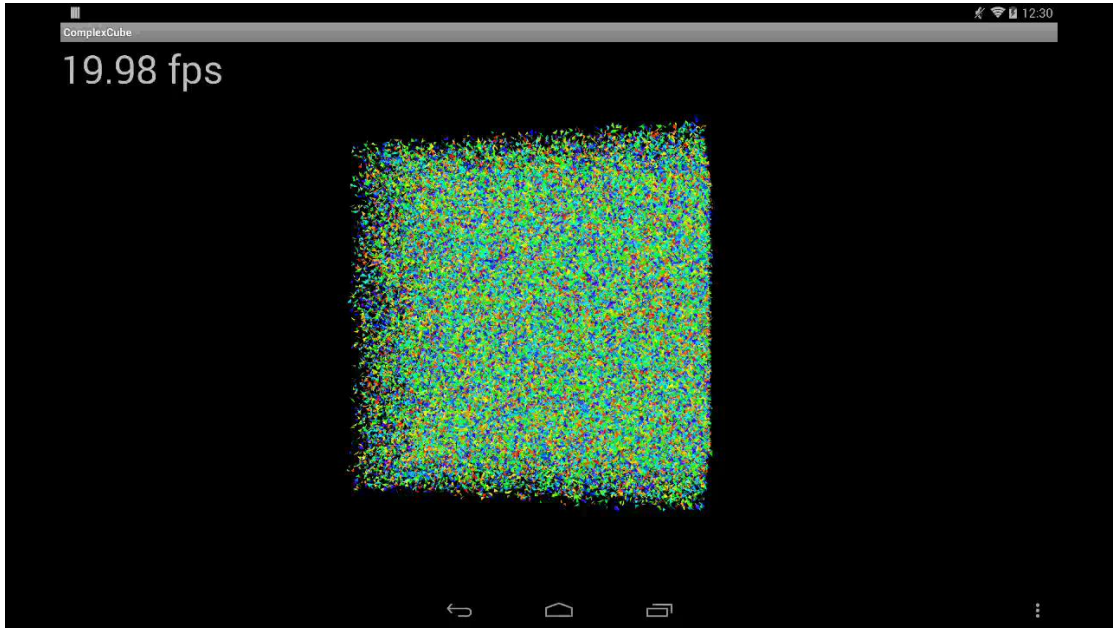
Draw 20,000 Triangles with a Single Draw Call



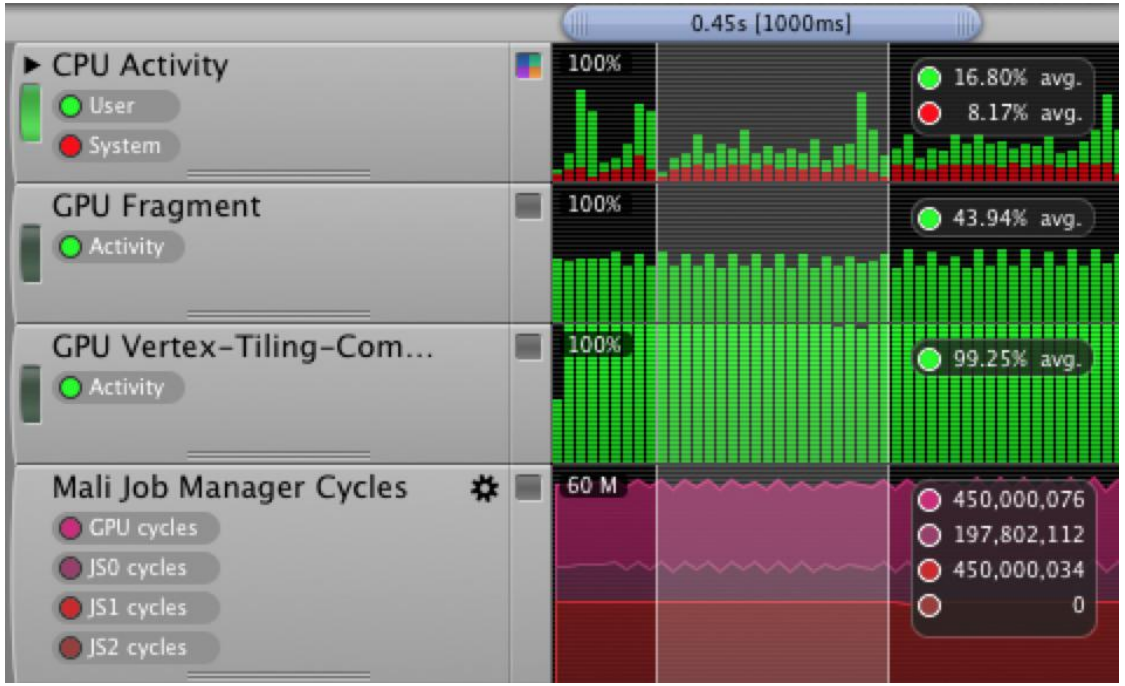
ARM

GPU Vertex Bound Case

100,000 Very Small Triangles



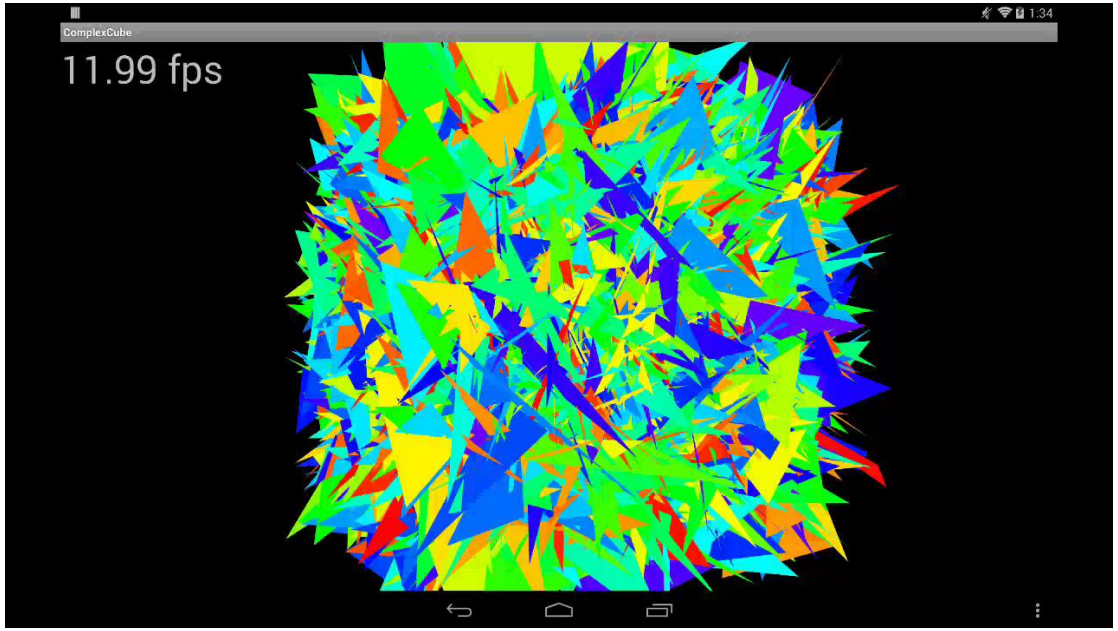
- Job Slot 1 cycles == GPU cycles
- GPU is running Vertex Shader for 99.25% of the time



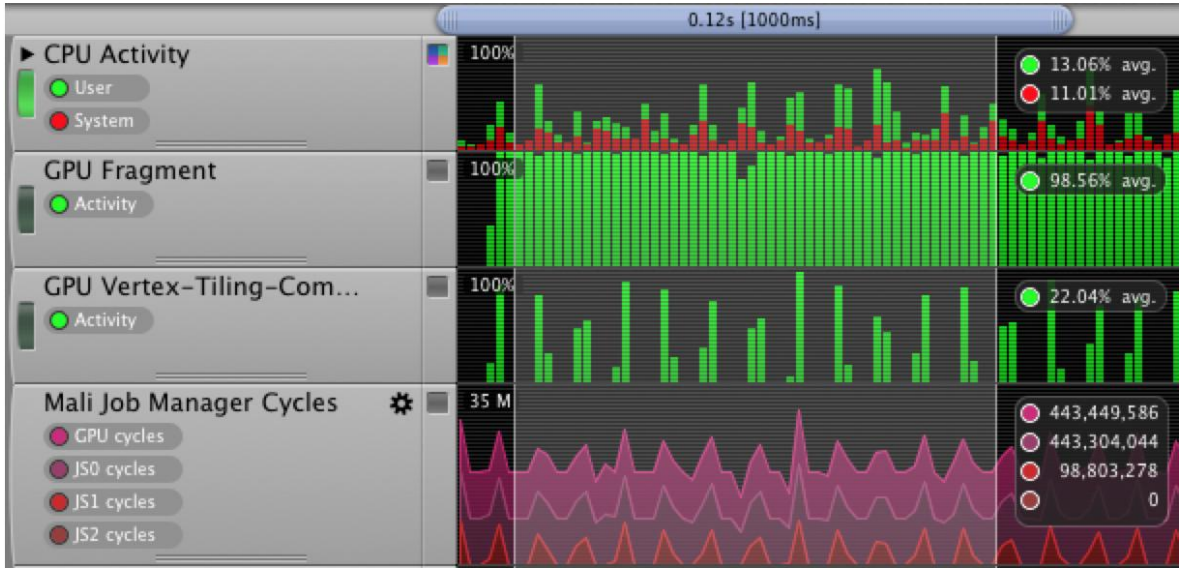
ARM

GPU Fragment Bound Case

20,000 Big Triangles



- Job Slot 0 cycles == GPU cycles
- GPU is running Fragment Shader for 98.56% of the time

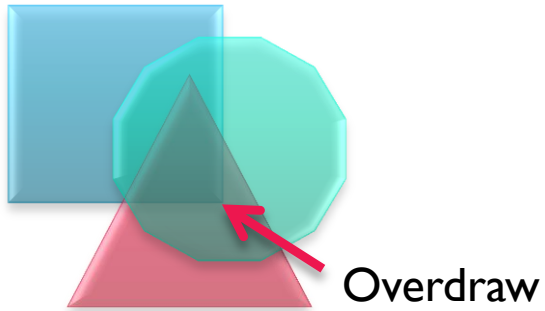


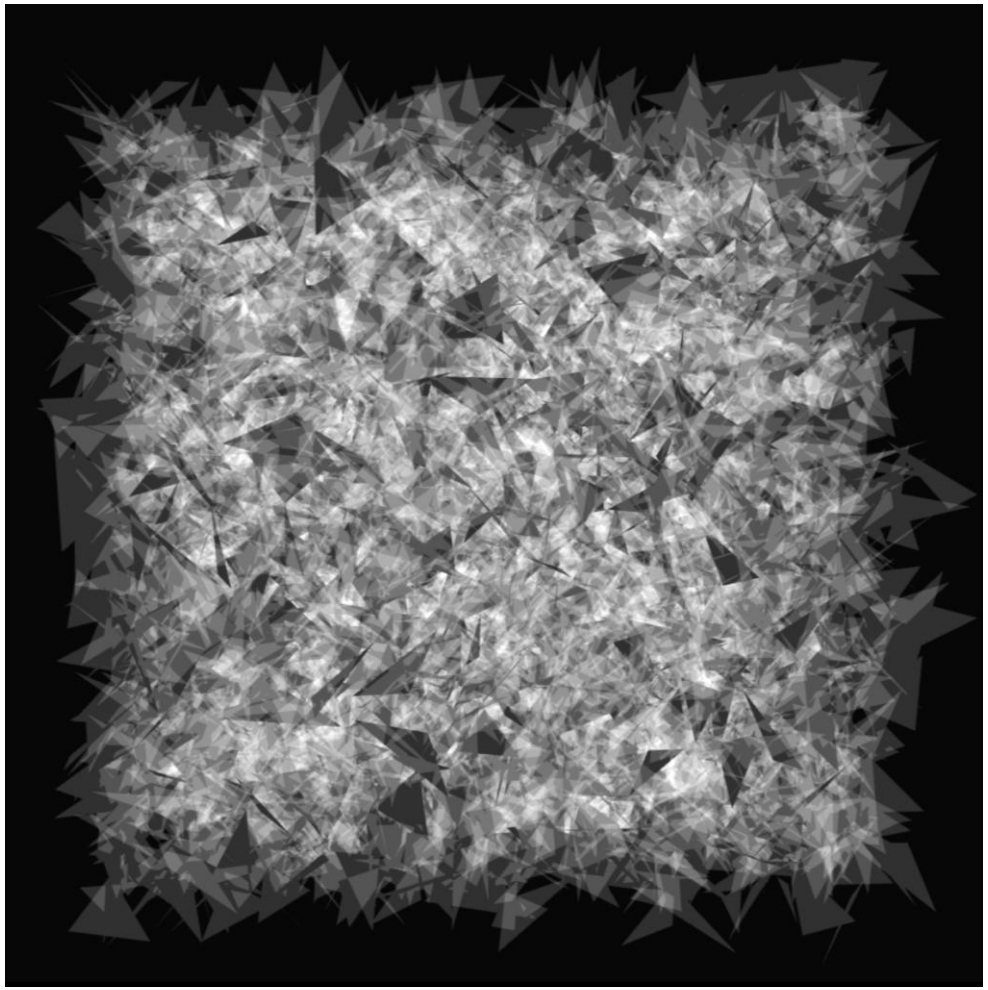
ARM

Overdraw

The Order You Draw Triangles Matters!!!

- This is when you draw to each pixel on the screen more than once
- Drawing your objects front to back instead of back to front reduces overdraw
- Limiting the amount of transparency in the scene can help

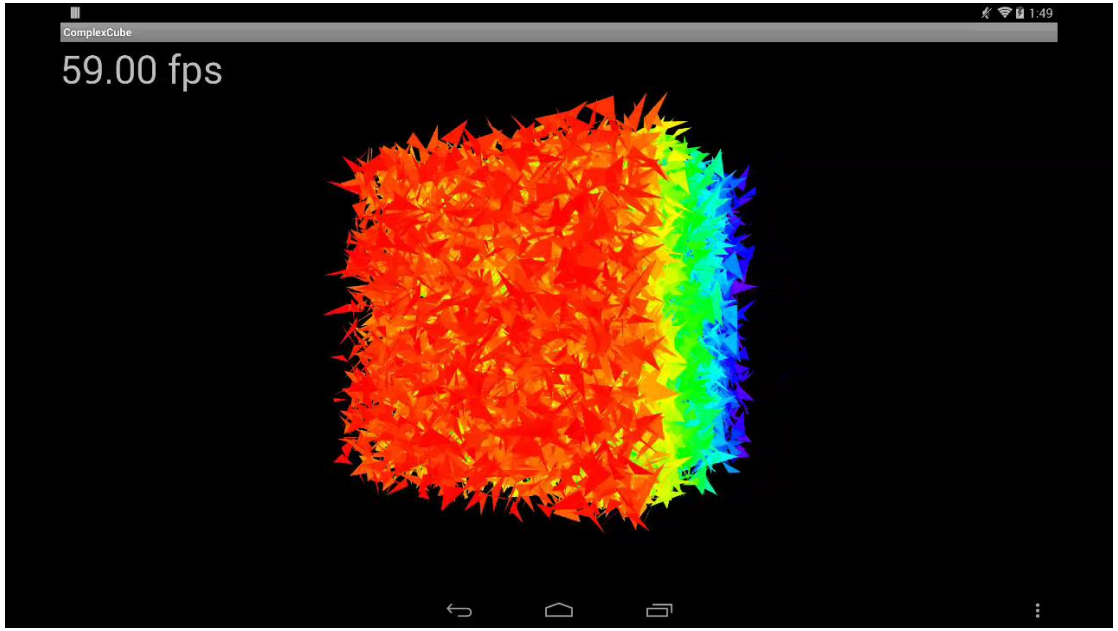




ARM

Overdraw

Detect High Overdraw and Sort Triangles



- Sort the objects in the scene
  - Front to back
  - Finally, semi-transparent objects
- If you know what face an object is going to show in the scene, even sort triangles
  - Don't just sort the indices in glDrawElements, but also sort the vertices in the buffer, for better caching
- ComplexCube with 20,000 sorted triangles, from red to blue
- We expect less overdraw when the cube is drawn with the red face in the front

ARM

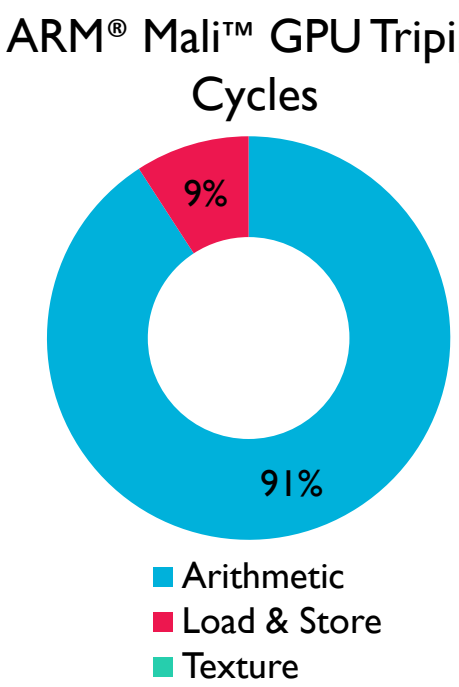
Early Z-test and Overdraw

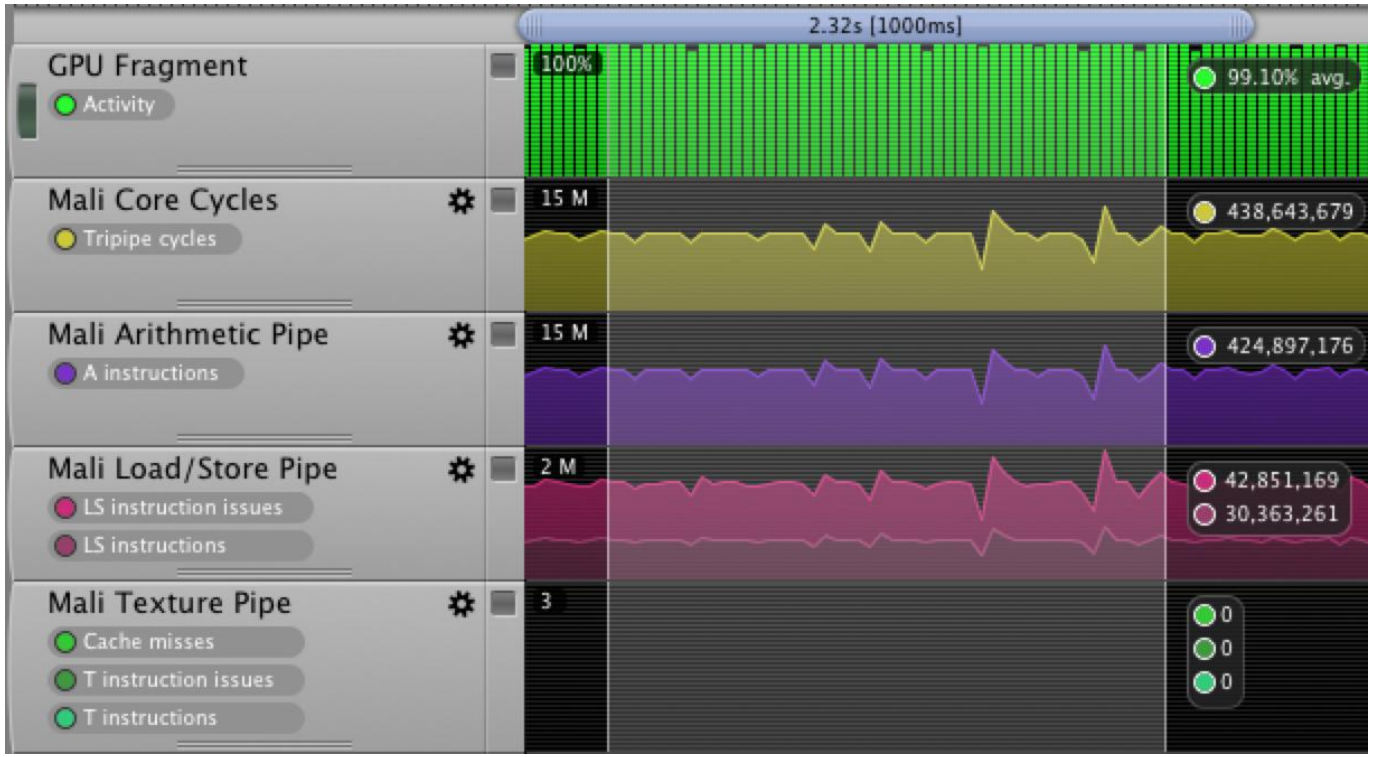
ComplexCube with 20,000 Sorted Triangles

ARM

Arithmetic Bound

- Arithmetic
- Load & Store
- Texture



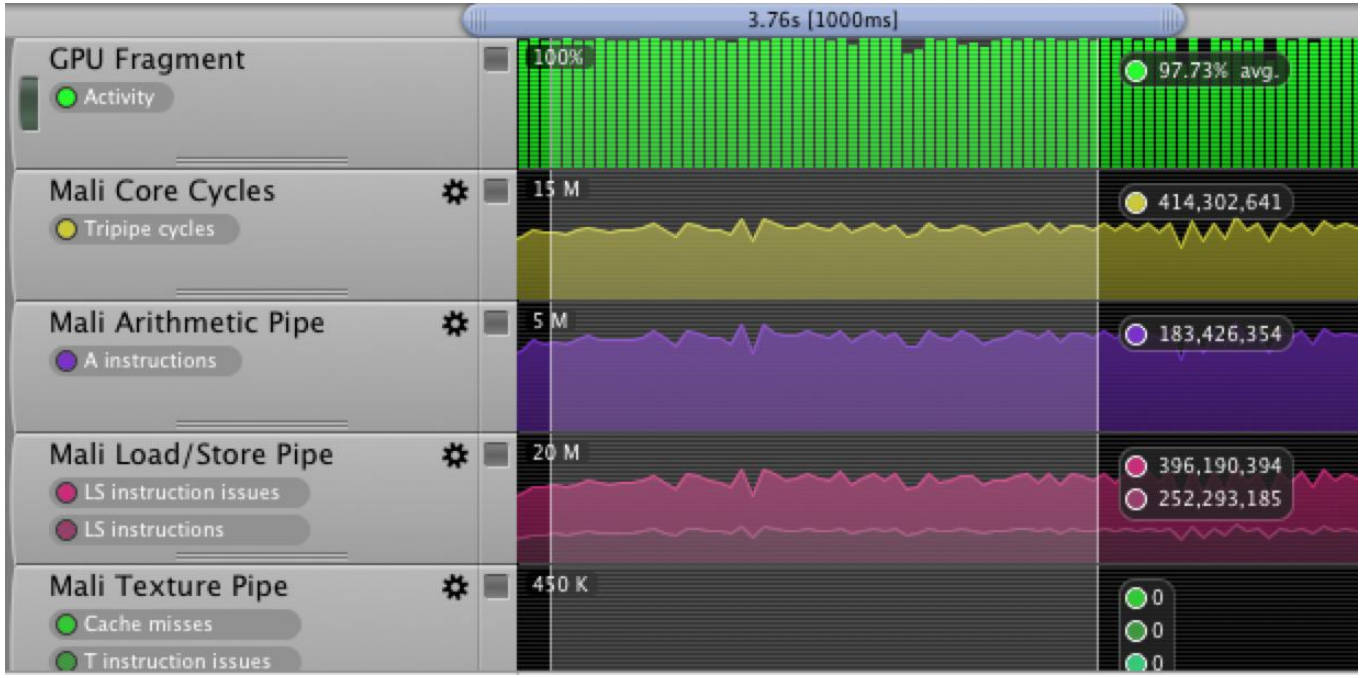



```
void main()
{
    vec3 a = fragColour;
    a = sqrt(a) * a / log(a)/sin(a)/cos(a)/tan(a)*sqrt(a);
    a = a * vec3(0.00001);
    gl_FragColor = vec4(fragColour + a, 1.0);
}
```

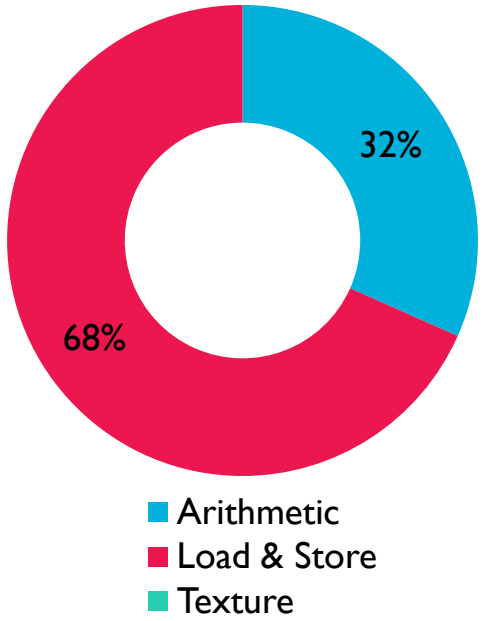

ARM

Load and Store Bound

- Vertex Attributes
- Uniforms
- Varyings



ARM® Mali™ GPU Tripipe Cycles



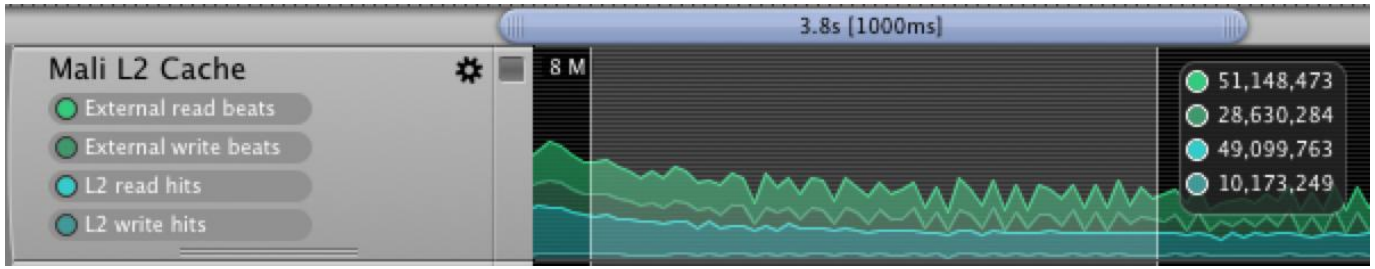

```
varying vec3 over1;
varying vec3 over2;
varying vec3 over3;
varying vec3 over4;
varying vec3 over5;
void main()
{
    vec3 a = fragColour;
    a = fragColour + over1 + over2 + over3 + over4 + over5;
    gl_FragColor = vec4(fragColour + a, 1.0);
}
```


ARM

Bandwidth Counters

ComplexCube with 20,000 Triangles and High Number of Varyings

- When creating embedded graphics applications bandwidth is a scarce resource
  - A typical embedded device can handle around 5GB/s of bandwidth
  - A typical desktop GPU can do in excess of 100GB/s
- Involves just two ARM® Streamline™ counters
  - External Bus Read Beats
  - External Bus Write Beats
- In our case:
  - (51.1M + 28.6M) * 16 bytes = 1.18 GB/s



Bandwidth in Bytes = (External Bus Read Beats + External Bus Write Beats) * Bus Width

ARM

Thank You