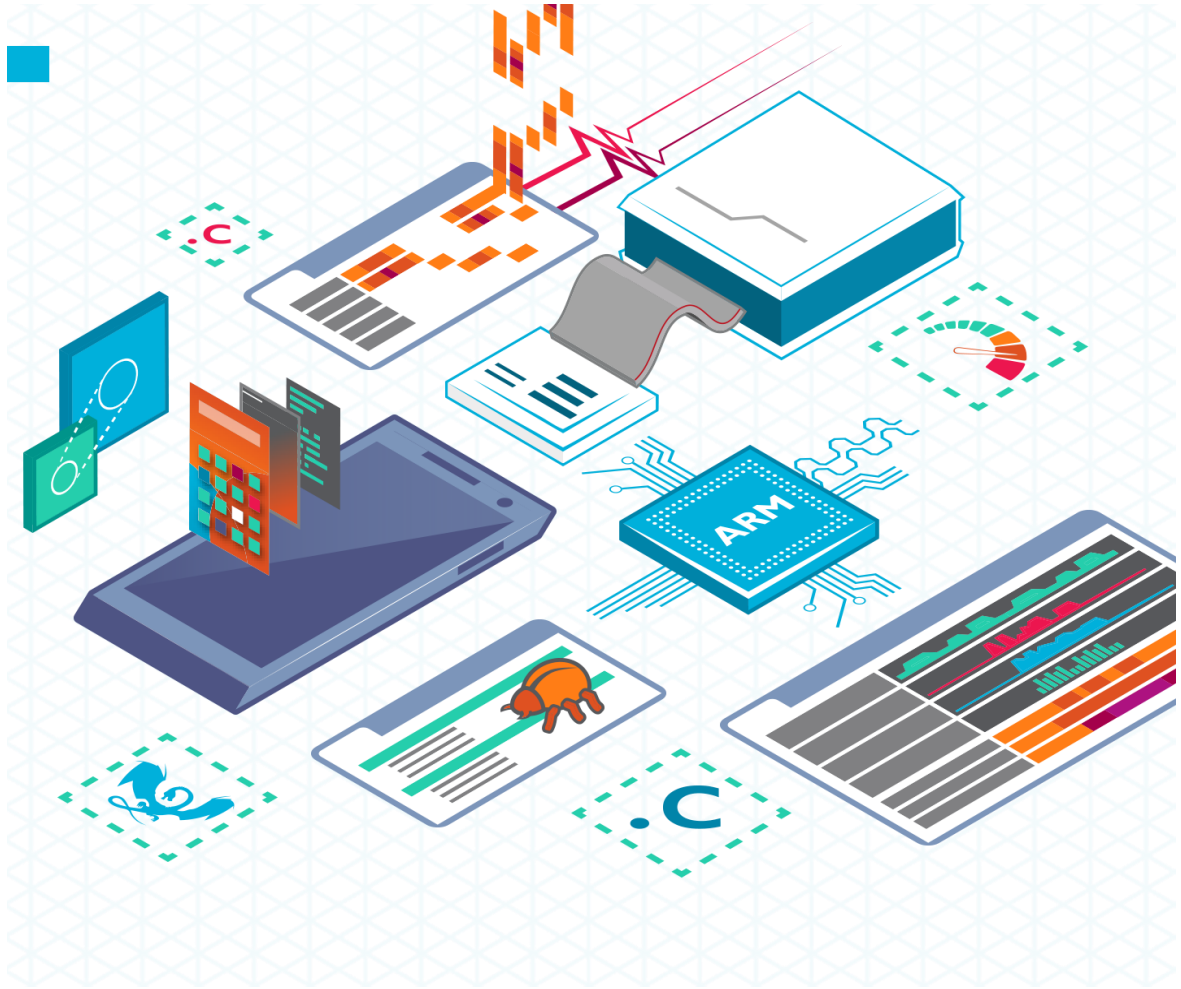
Profiling and Debugging OpenCL™ Applications with ARM® Development Tools

October 2014

Agenda

1. Introduction to GPU Compute
2. ARM® Development Solutions
3. Mali™ GPU Architecture
4. Using ARM DS-5 Streamline with OpenCL
5. Optimization tips



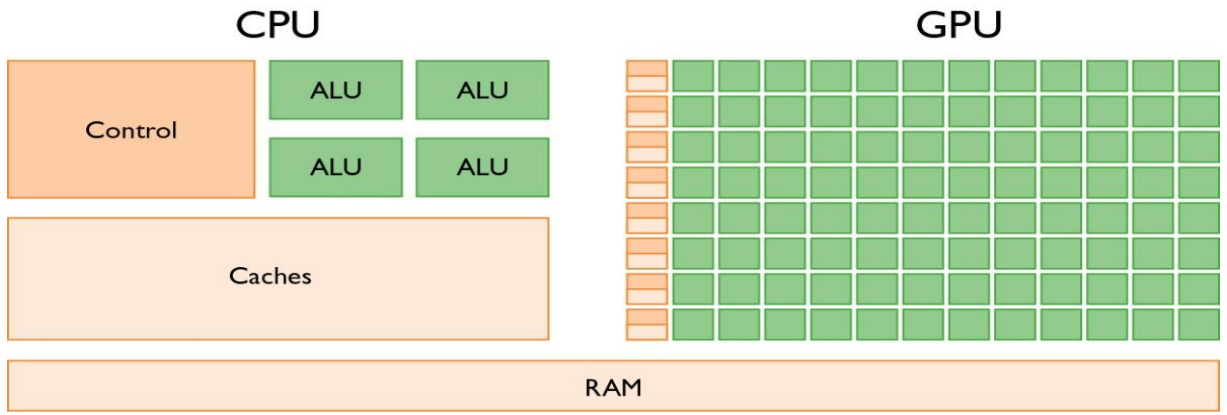
ARM

What is GPU Compute?

The GPU is now programmable through C-like languages

Cost-effective, efficient, and high-performance parallel computation

OS and applications



Computational accelerator or as a companion processor

- 2D/3D Graphics
- Image processing
- Multimedia
- Physics

ARM

The Evolution of Mobile GPU Compute

OpenGL ES 3.1 Compute Shaders
GPU Compute within graphics pipeline

OpenCL Full Profile/RenderScript
Portable Heterogeneous Parallel Computation

OpenGL ES 2.0
Programmable pipeline

OpenGL ES 1.1
Fixed pipeline

Mali-55

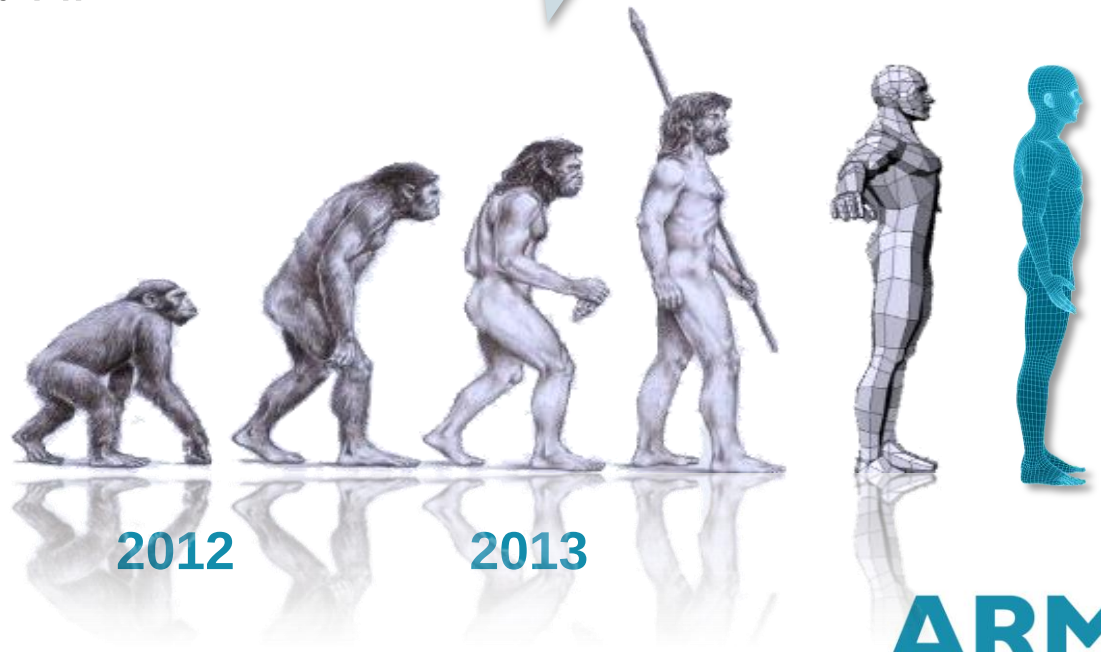
Mali-200
Mali-300

Mali-400 MP

Mali-T600
Series

Mali-T700
Series

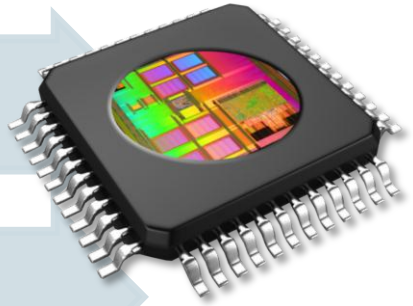
2007 2009 2010 2012 2013



ARM

GPU Compute: Improve Existing and Enable New Solutions

Increased system-level energy efficiency	• Complement CPU processing • Enable choice of best processor for the job
Better load-balance across system resources	• Use heterogeneous compute APIs designed for concurrency
Free up CPU resource	• Offload non-graphical computational tasks to GPU
Flexibility, portability and programmability	• Software solution leveraging CPU+GPU subsystem • Industry standard portable APIs
Improve user experience	• Remove computational barrier to improve visual quality, responsiveness, accuracy within existing compute & energy budgets
Reduce cost, risk and TTM	• Enable new applications using existing silicon design



ARM

ARM® Development Solutions

ARM

ARM Development Solutions Mission

- To help developers create innovative, robust and energy efficient ARM-based products

Tools for IP development and deployment

Facilitating software development on ARM

Strengthening the ARM tools ecosystem

From sensors

Internet of things

Safety standards

big.LITTLE™

GPU compute

Juno platform

AArch64

Multi-cluster

Network interconnect

to servers.

ARM

DS-5: Tools from Sensors to Servers

Everything ARM

Support for all ARM processor cores
Early support for new IP

End-to-End Solution

All tools you need,
from code generation to optimization

Track Record

Billions of products have shipped
over 20+ years based on ARM tools

ARM Expertise

Co-developed with ARM processors
Expert technical support

ARM

DS-5 Key Components

Compilation Tools

- ARM Compiler 5 and 6 C/C++ toolchains for bare-metal and RTOS
- Integrated Linaro GCC for ARM Linux

DS-5 Debugger

- Comprehensive device bring-up tools and s/w debugger for single- and multi-core platforms
- OS aware debug, on silicon, virtual platform and emulator

Streamline Analyzer

- CPU, GPU, interconnect performance and power analysis
- Time- and event-based profiling

BUILT ON eclipse

DS-5 IDE

- Powerful, customized editor based on industry standard Eclipse IDE 4.3
- Hundreds of compatible plugins

ARM

Streamline Analyzer

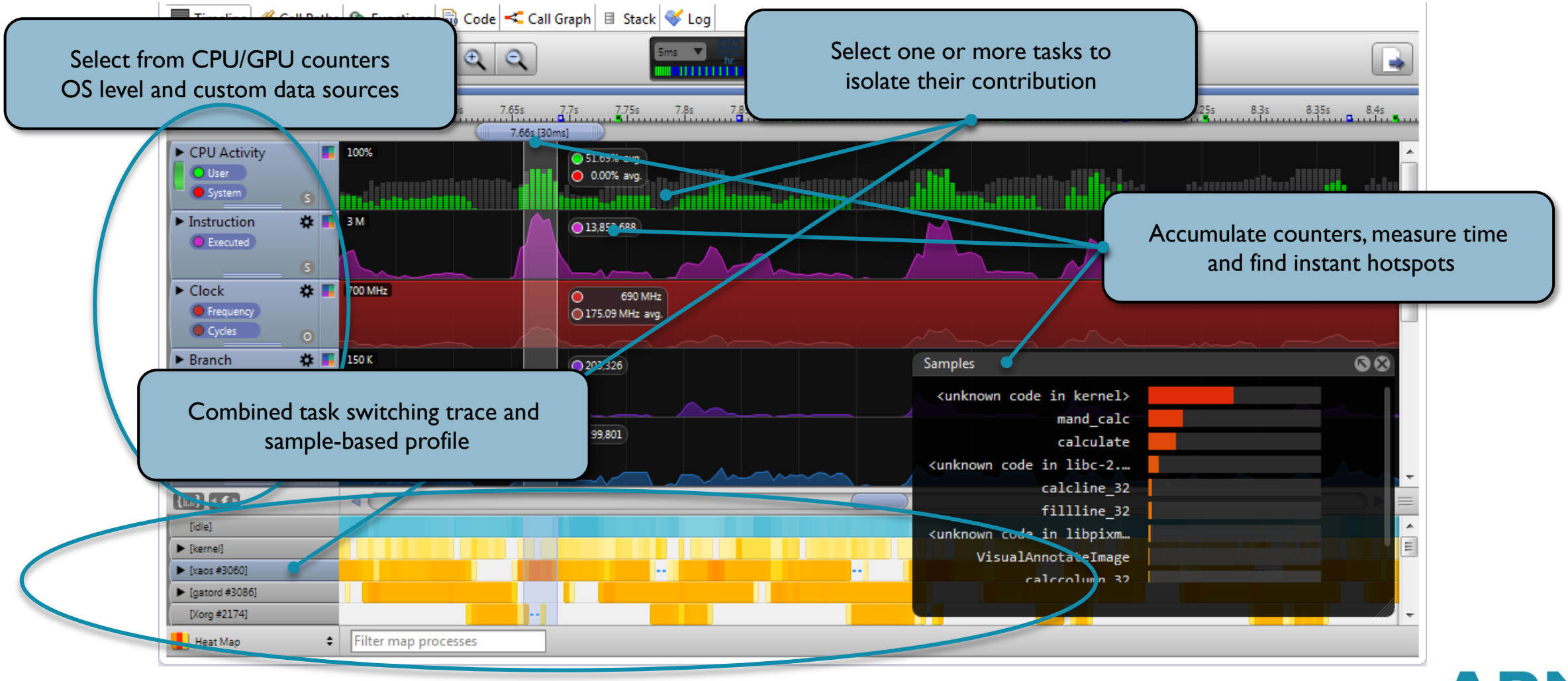
Debug and optimize system performance and power

- H/W and S/W performance data
- Task/thread execution sequence tracing
- Process-to-source CPU and event profiling
- Multicore utilization mapping
- Customizable data sources
- For Linux and Android
  - OpenGL® ES and OpenCL ® analysis on Mali GPU
  - No debug or trace probe required
  - Streaming data option for long captures
- For embedded
  - Negligible overhead, based on DWT and ITM
  - Compatible with Cortex-M3 and Cortex-M4

ARM

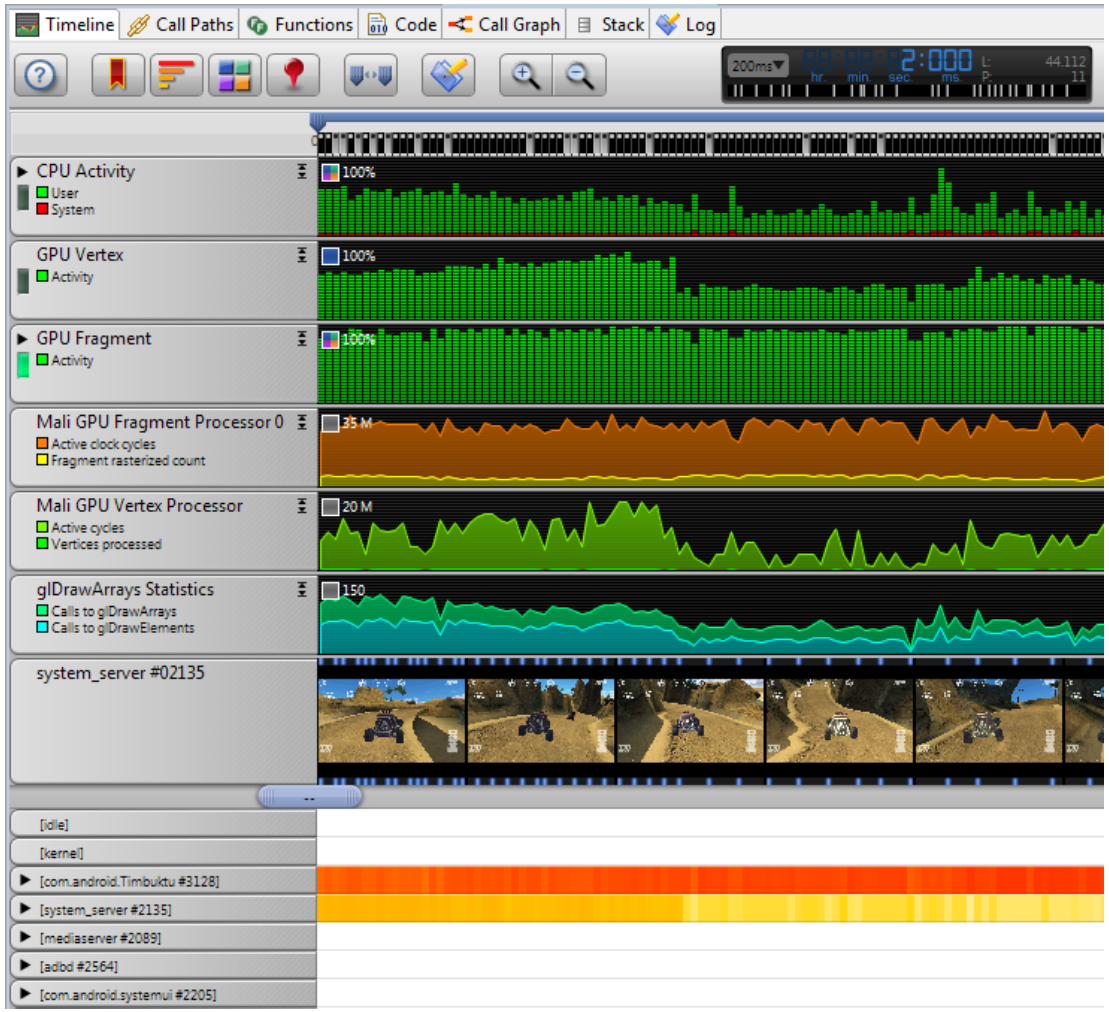
Timeline: Heat Map

Identify hotspots and system bottlenecks at a glance

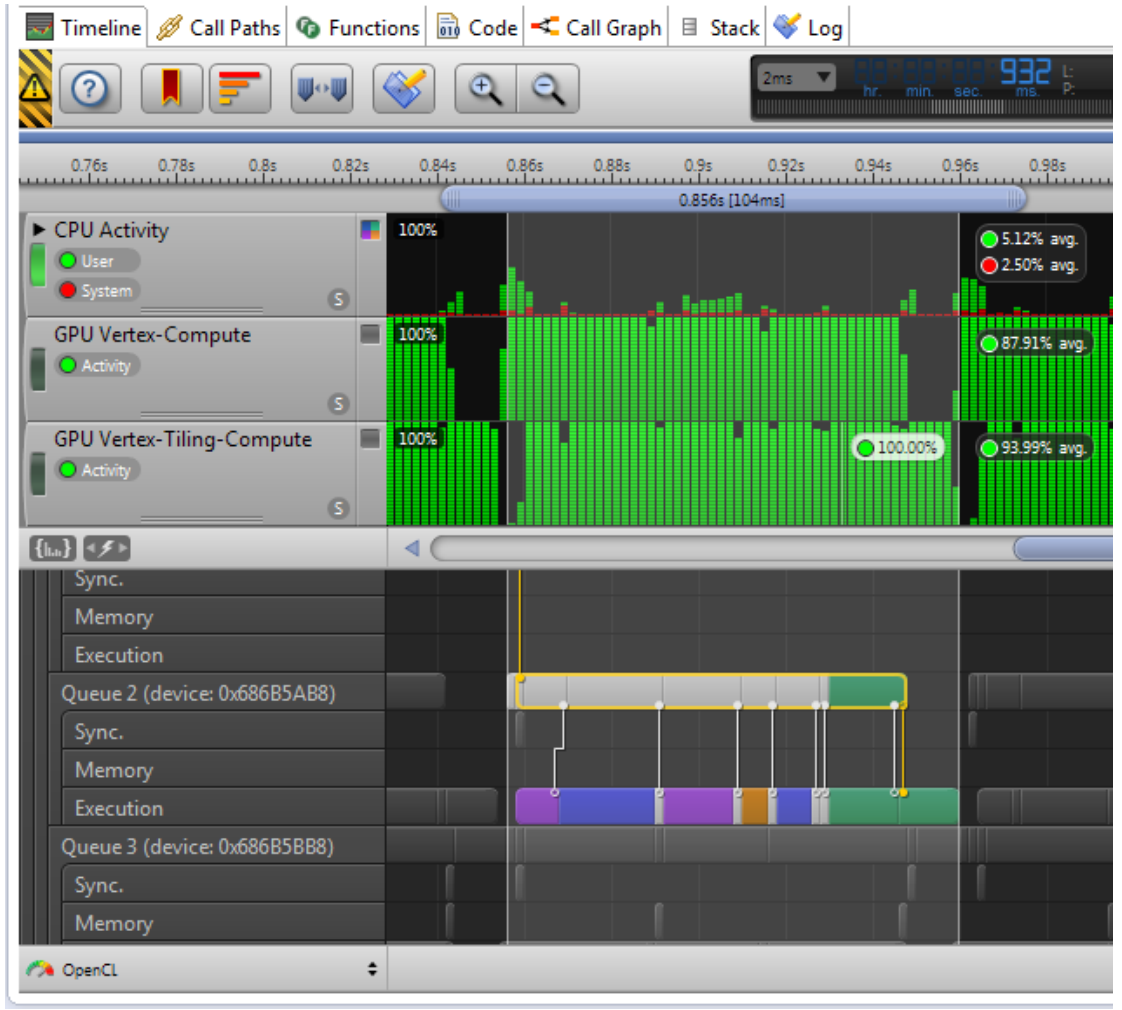


Mali GPU Analysis

OpenGL ES



OpenCL Compute

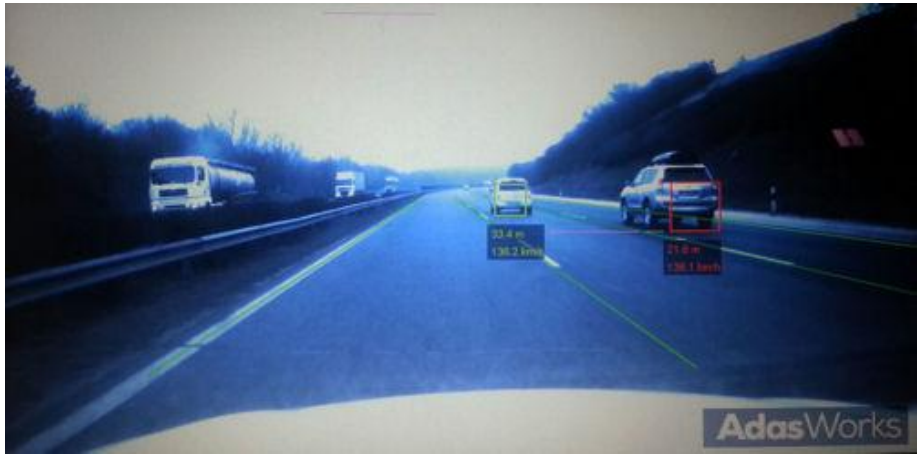


A Practical Example: Optimizing an ADAS Application

ARM

ADAS Case Study

- Traffic Lane Detection Algorithm
  - Chain of filters running on the GPU
  - Analyzed using ARM DS-5 Streamline
  - Significant gaps in GPU activity easily identified
  - Trace images sent back to developer
- Updated Version from Developer
  - Better pipelining between CPU & GPU
  - 2 x performance improvement overall

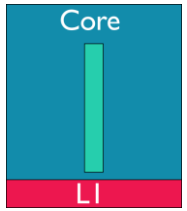


ARM

OpenCL Execution Model on Mali GPUs

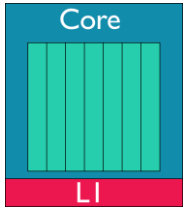
Work Item

Registers, PC, SP, Private Stack

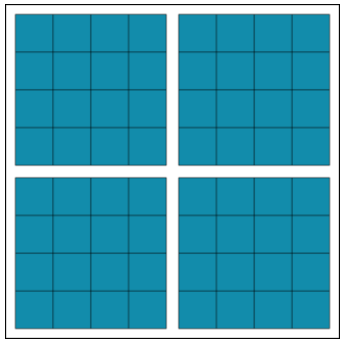


Work Group

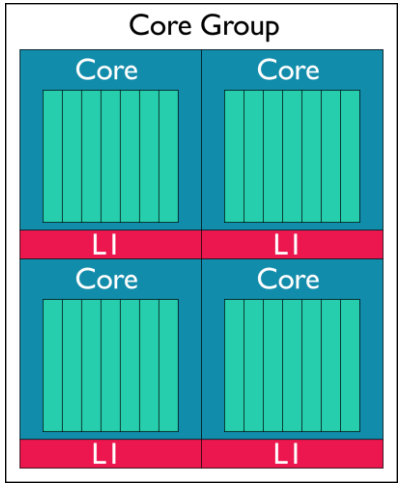
Barriers, Local Atomics,
Cached Local Memory



ND Range



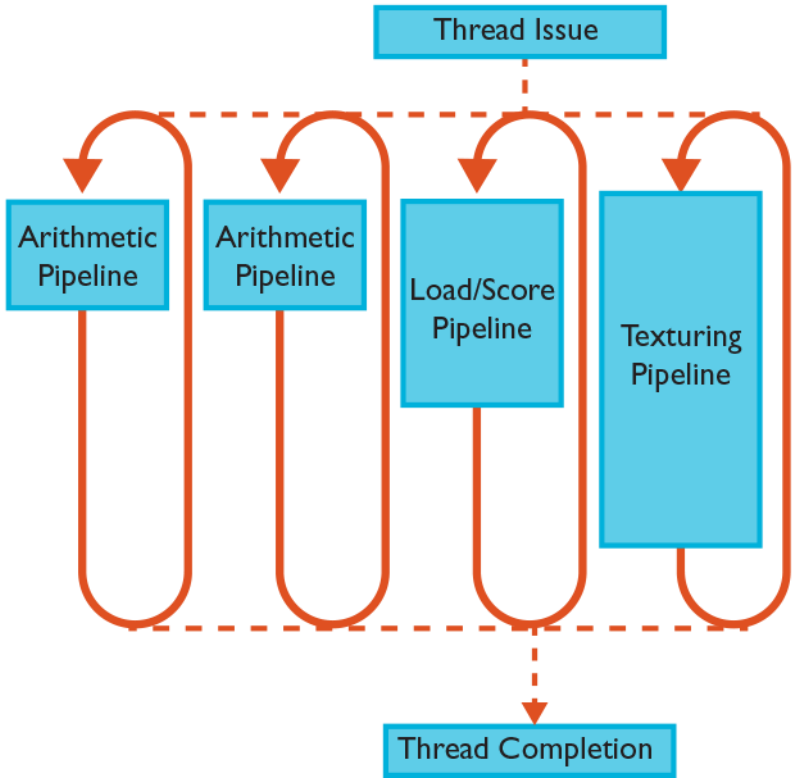
Global Atomics,
Cached Global Memory

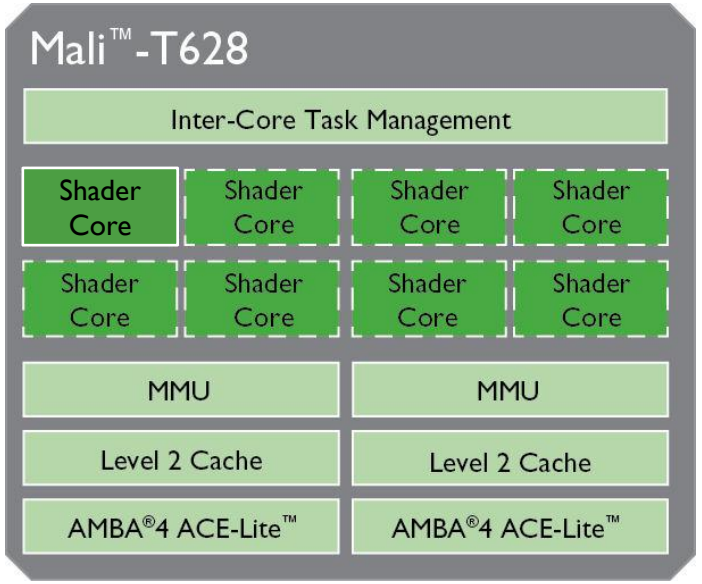


ARM

Inside a Mali GPU Core

- SIMD: Several components per operation
  - 128-bit registers
- VLIW: Several operations per instruction word
  - Some operations are "free"
- Built in function library
  - Accelerated in hardware





ARM

Hardware Counters

- Counters per core
  - Active cycles
  - Pipe activity
  - L1 cache
- Counters per core group
  - L2 caches
  - MMU
- Counters for the GPU
  - Active cycles
- Accessed through Streamline
  - Timeline of all hardware counters, and more
  - Explore the execution of the full application
  - Zoom in on details

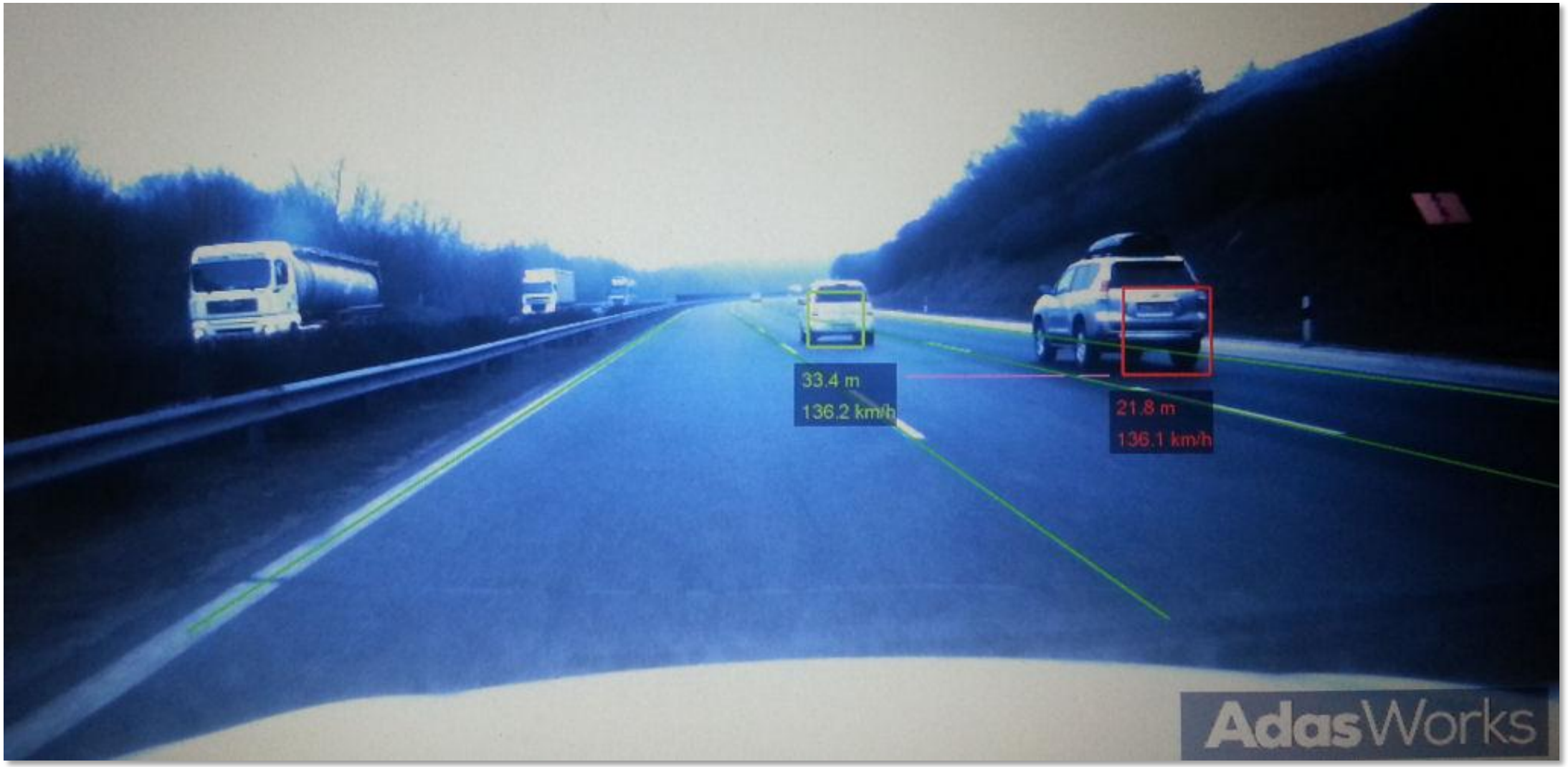
Streamline

ARM

Streamline

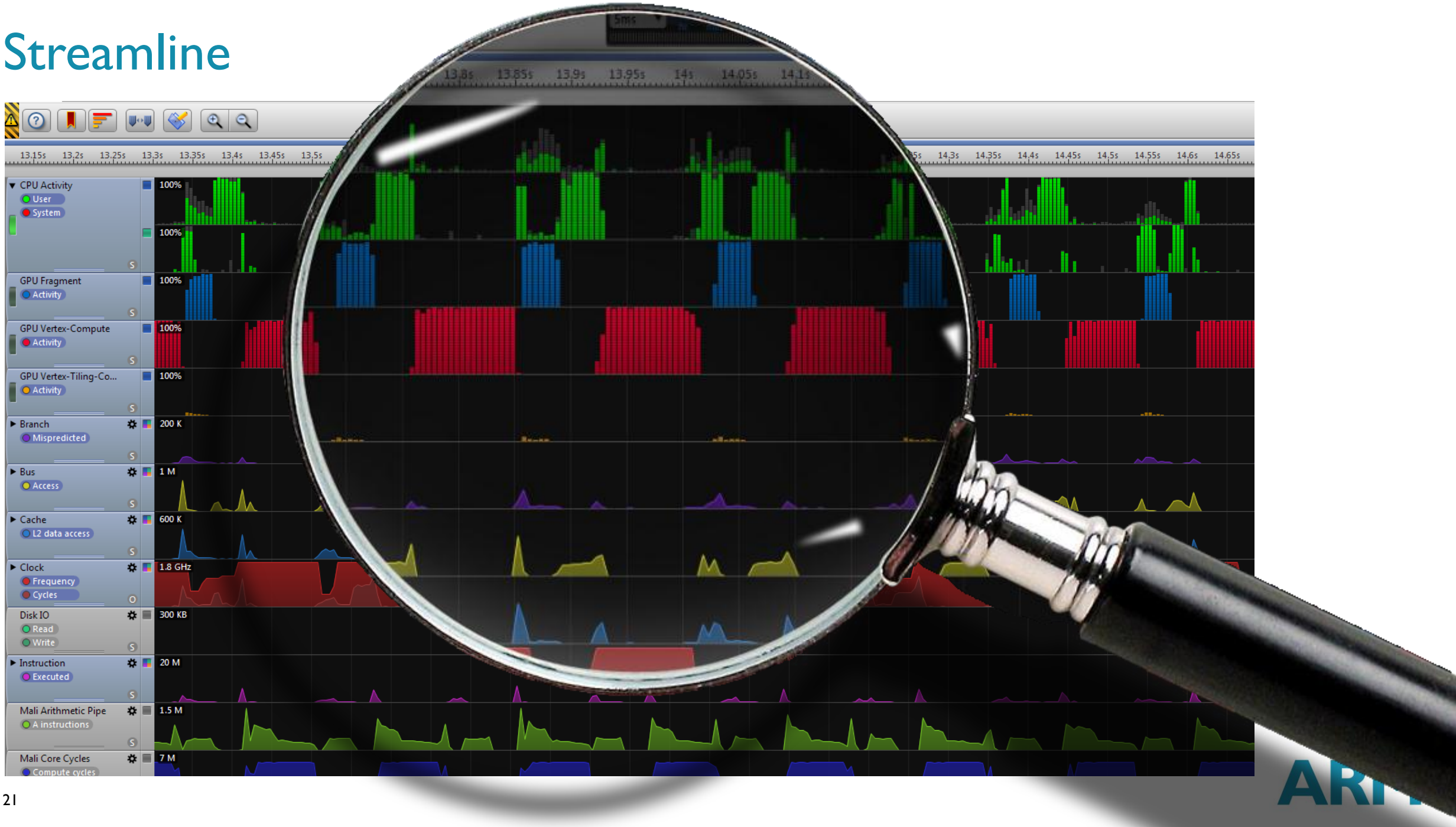

Mali Compute Threads
Compute tasks
Compute threads started
Mali Core Cycles
Compute cycles
Fragment cycles
Fragment cycles w
Tripipe cycles
Mali Arithmetic Pi
A instructions
Mali Load/Store
LS instruction
LS instruction
Mali Texture Pipe
T instructions
Mali Load/Store Ca
Atomic hits
Atomic misses
Dirty line evictions
Line fetches
Read hits
Read misses
Snoops in to LSC
Write hits
Write misses
Mali Core Threads
Fragment threads
450
167,600
151,029

Lane and Car Detection

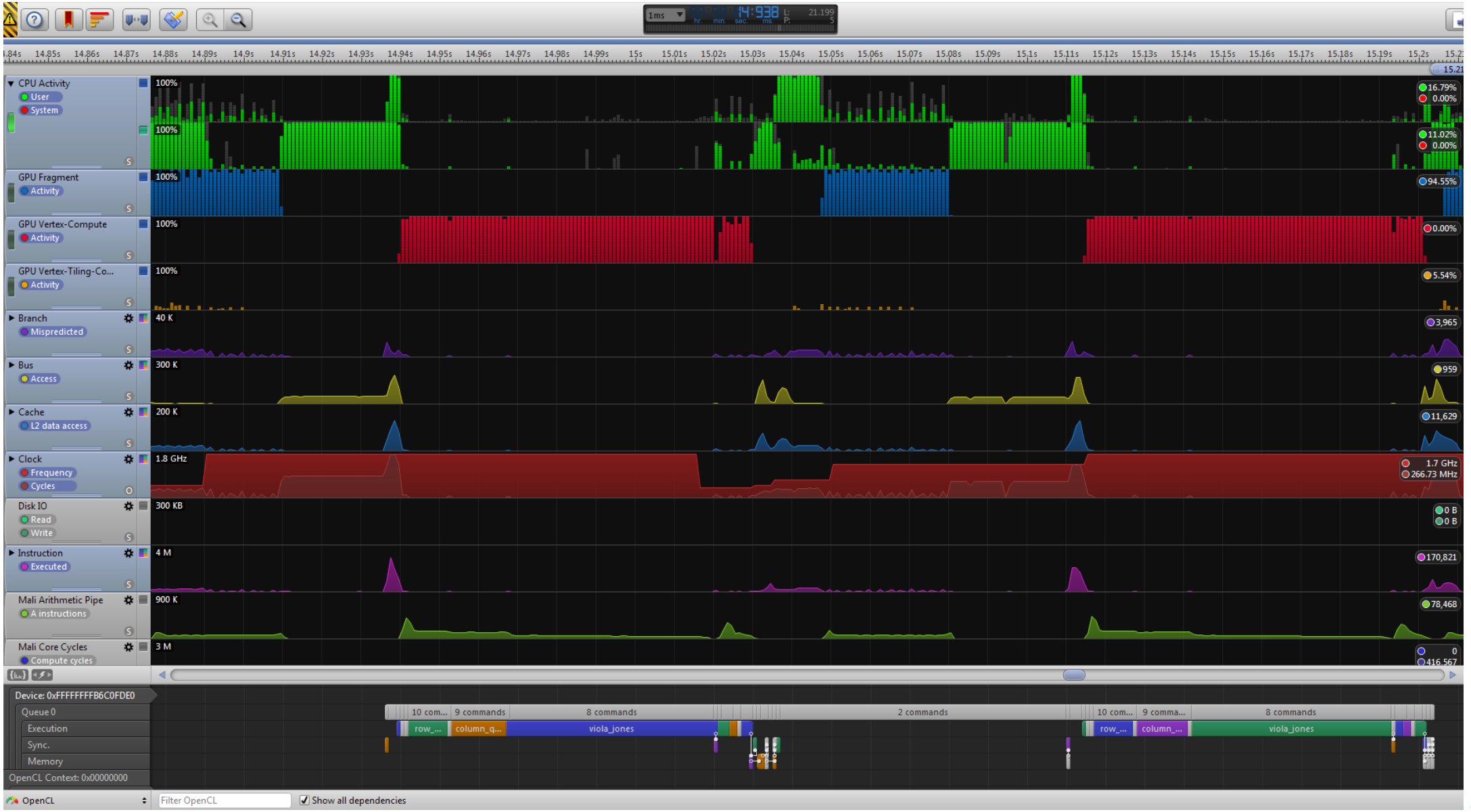


ARM

Streamline

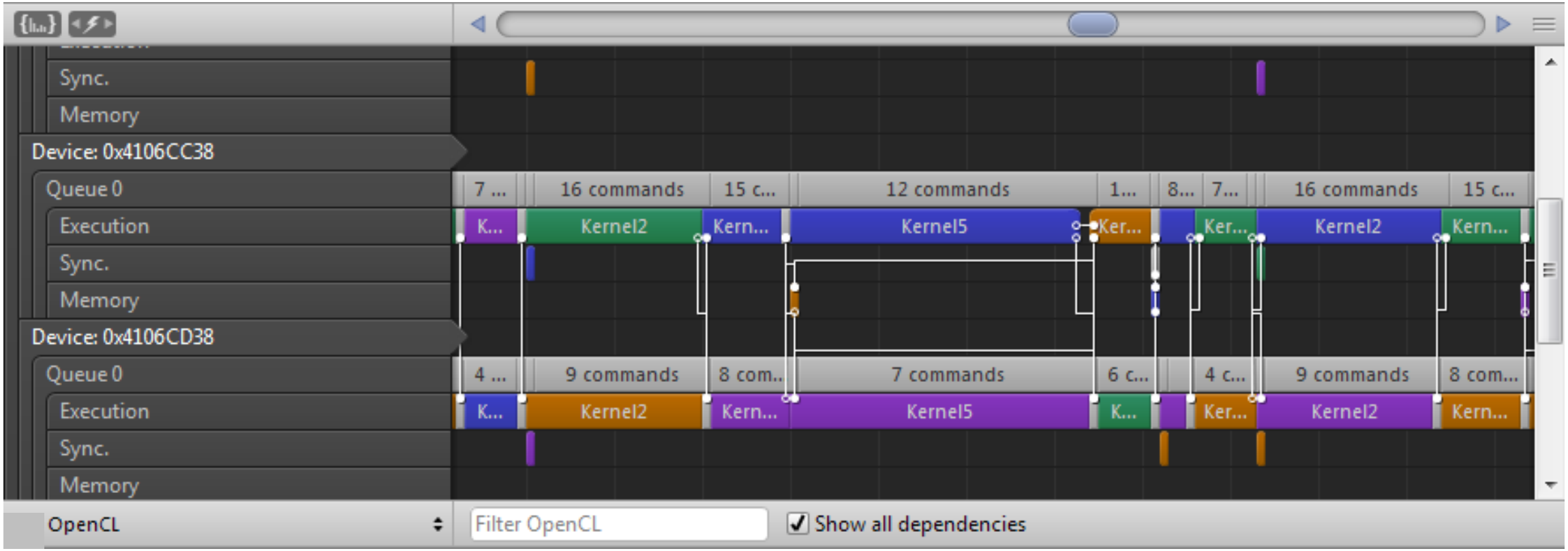

CPU Activity
User
System
100%
100%
GPU Fragment
Activity
100%
GPU Vertex-Compute
Activity
100%
GPU Vertex-Tiling-Co...
Activity
100%
Branch
Mispredicted
200 K
Bus
Access
1 M
Cache
L2 data access
600 K
Clock
Frequency
Cycles
1.8 GHz
Disk IO
Read
Write
300 KB
Instruction
Executed
20 M
Mali Arithmetic Pipe
A instructions
1.5 M
Mali Core Cycles
Compute cycles
7 M

Streamline: OpenCL Timeline



Streamline: OpenCL Timeli

Streamline: OpenCL Timeline



ARM

Optimizing Memory Traffic

- Only one programmer-controlled memory
  - Many transparent caches
- Memory copying takes time
  - It can easily dominate over kernel execution time
- Use appropriate memory allocation schemes
- Avoid synchronization points
  - Cache maintenance has a cost as well
- Streamline to the rescue
  - Visualize when kernels are executed
  - Many more features

Hiding Pipeline Latency

- Needs enough threads
  - Limited by register usage
- When there are issues
  - Few instructions issued per cycle
  - Spilling of values to memory
- Symptoms
  - Low Max Local Workgroup Size in OpenCL™
  - Few instructions issued per cycle in limiting pipe
- Remedy
  - Smaller types → More values per register
  - Splitting kernels

Finding Processing Bottlenecks

- Host application or kernel execution
  - Avoid memory copying
  - Avoid cache flushes
- Which pipe is important?
  - Operations in other pipes incur little or no runtime cost
- Saving operations or saving registers
  - How much register pressure can we handle, and still hide the latencies?
- How well are we using the caches
  - Are instructions spinning around the LS pipe waiting for data?

Thank You

Any Questions?

malideveloper.arm.com

ds.arm.com

community.arm.com