

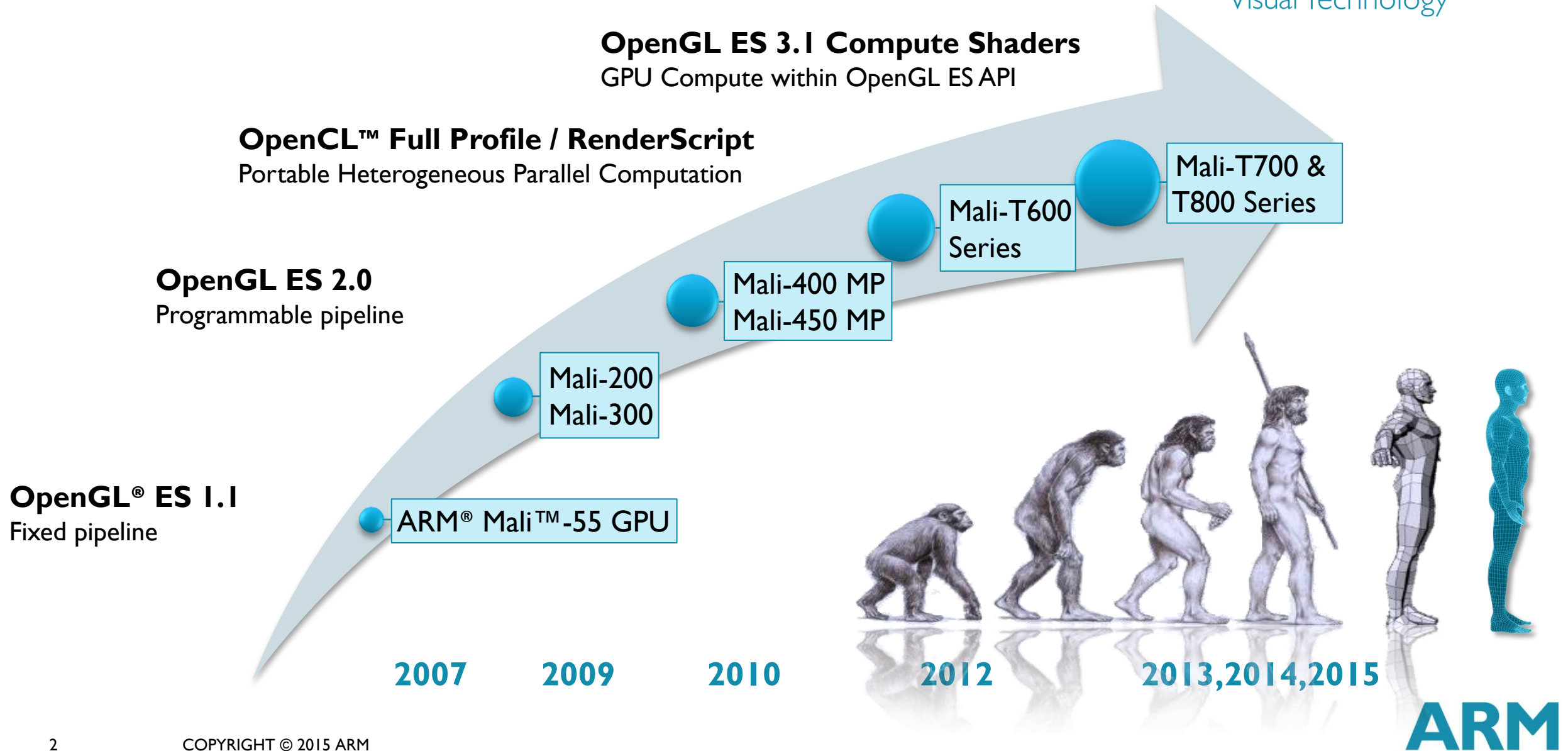
Measuring the Whole System

Holistic Profiling of CPU and GPU for Optimal Vision Applications on ARM Platforms

Tim Hartley

The Evolution of Mobile GPU Compute

ARM[®]MALI[™]
Visual Technology

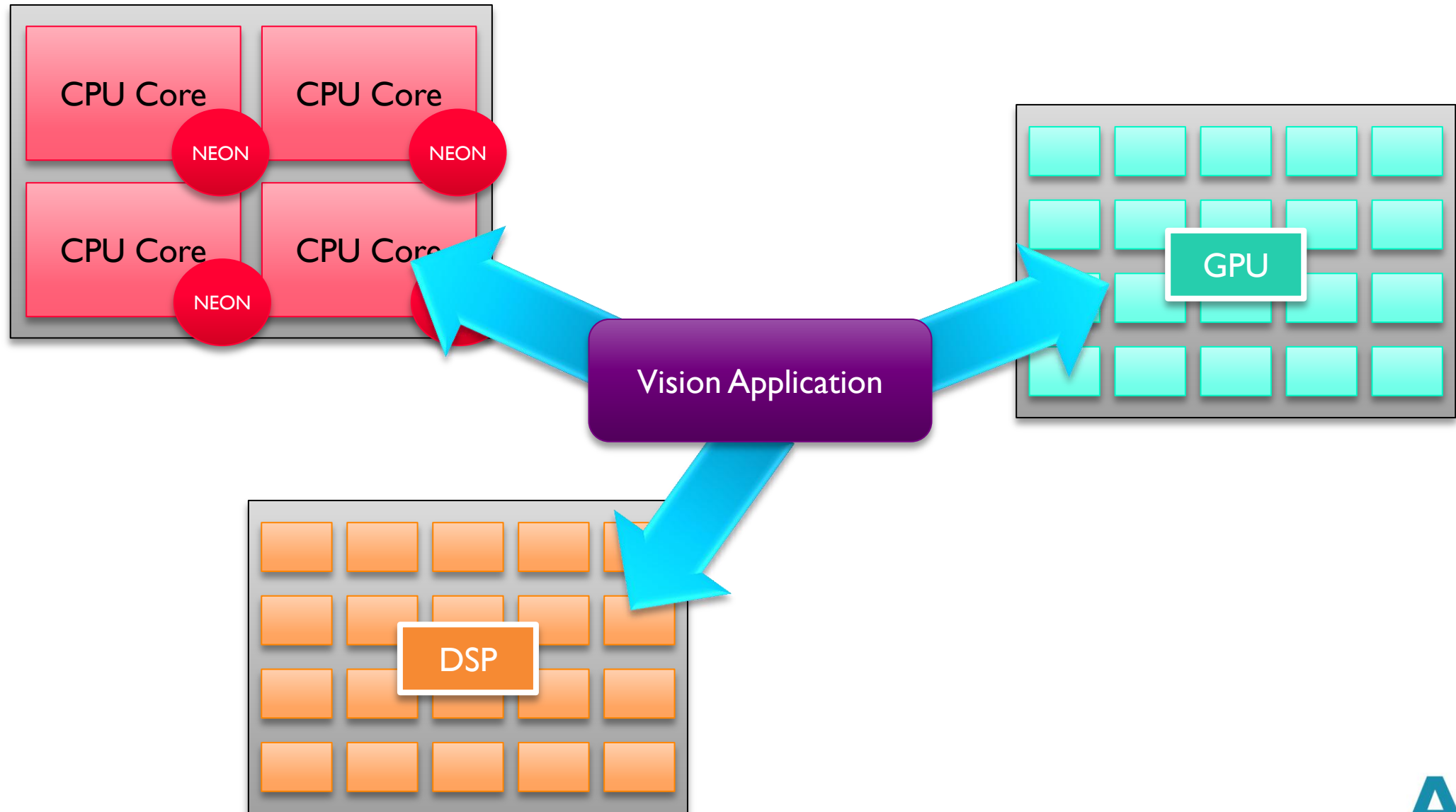


Measuring the Whole System

- Computer Vision will, for some time, succeed in using every drop of processing power we give it
 - And techniques in computer vision still evolving rapidly
- New, complex, sustained low power use cases
- Building computer vision applications an ever more complex process
 - The availability of more processors and processor types makes this even more so

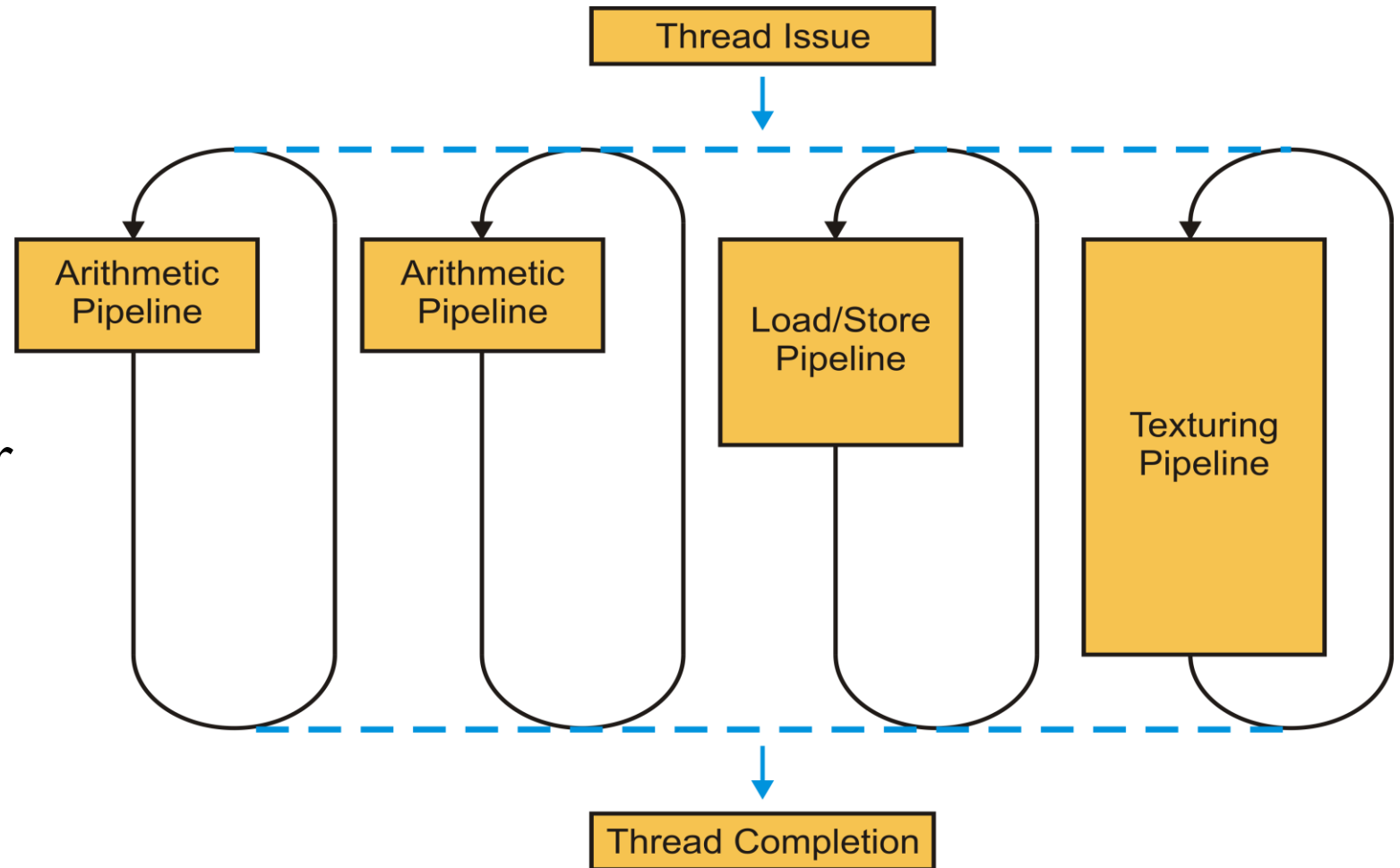
Capturing and analyzing accurate and effective measurements from platforms plays a vital role in achieving optimal performance

Modern Computer Vision Applications



Inside an ARM Mali Midgard Core

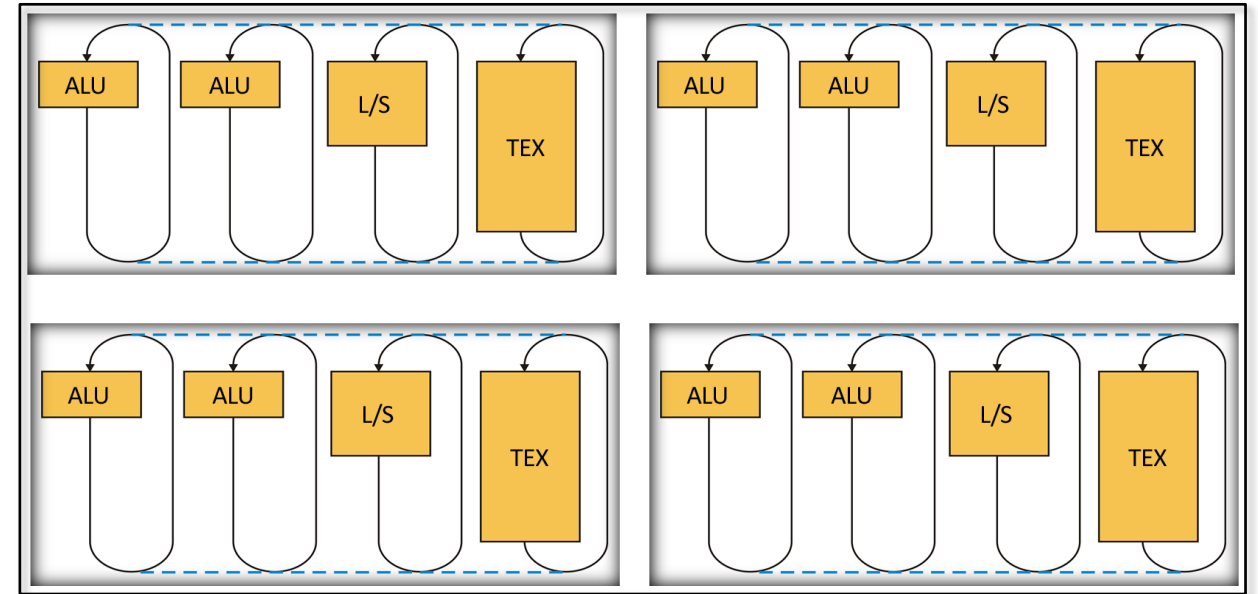
- SIMD: Several components per operation
 - 128-bit registers
- VLIW: Several operations per instruction word
 - Some operations are “free”
- Built in function library
 - Accelerated in hardware



$$T = \max(A_0, A_1, LS, Tex)$$

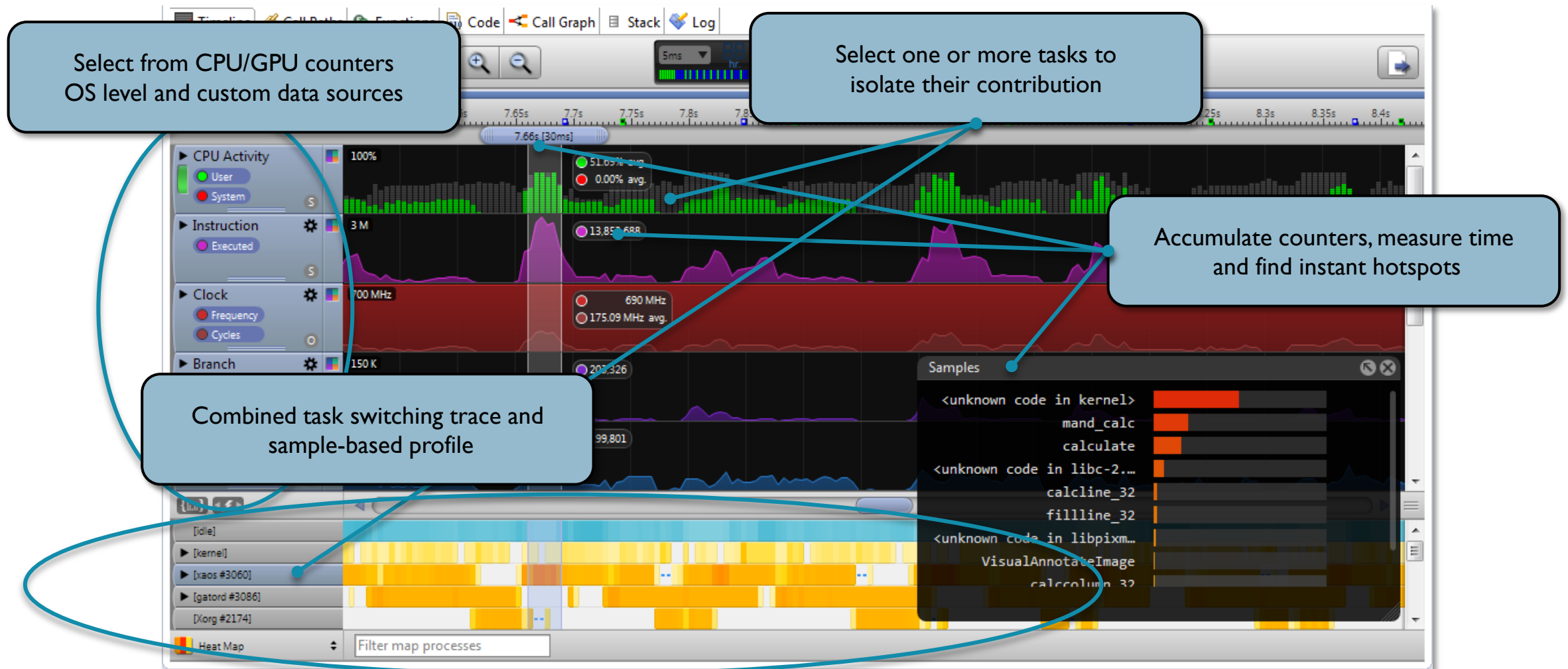
Hardware Counters

- Counters per core
 - Active cycles
 - Pipe activity
 - L1 cache
- Counters for the GPU
 - Active cycles
 - L2 caches
 - MMU
- Accessed through DS-5 Streamline
 - Timeline of all hardware counters, and more
 - Explore the execution of the full application
 - Zoom in on details



DS-5 Streamline

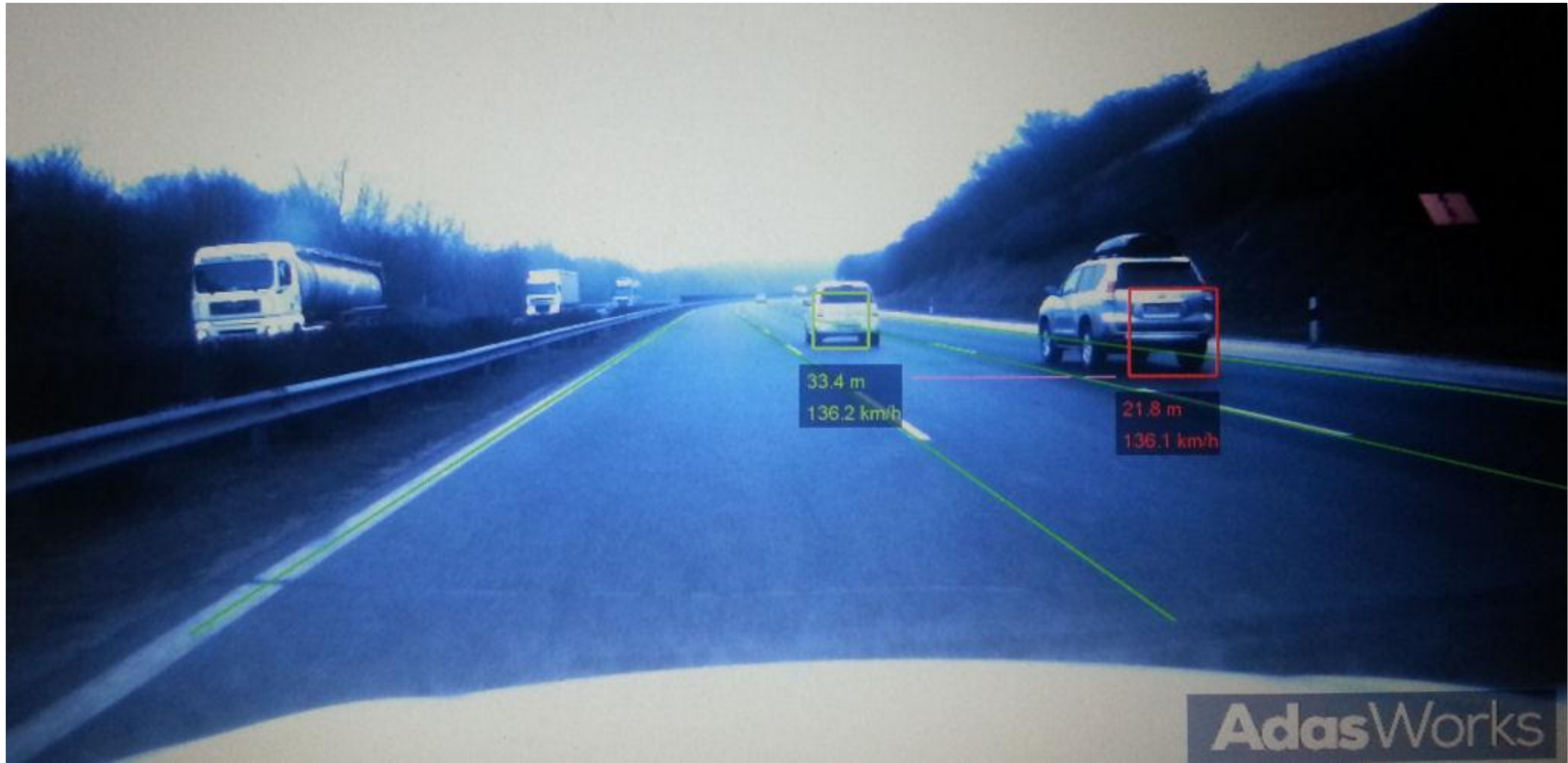
Identify hotspots and system bottlenecks at a glance



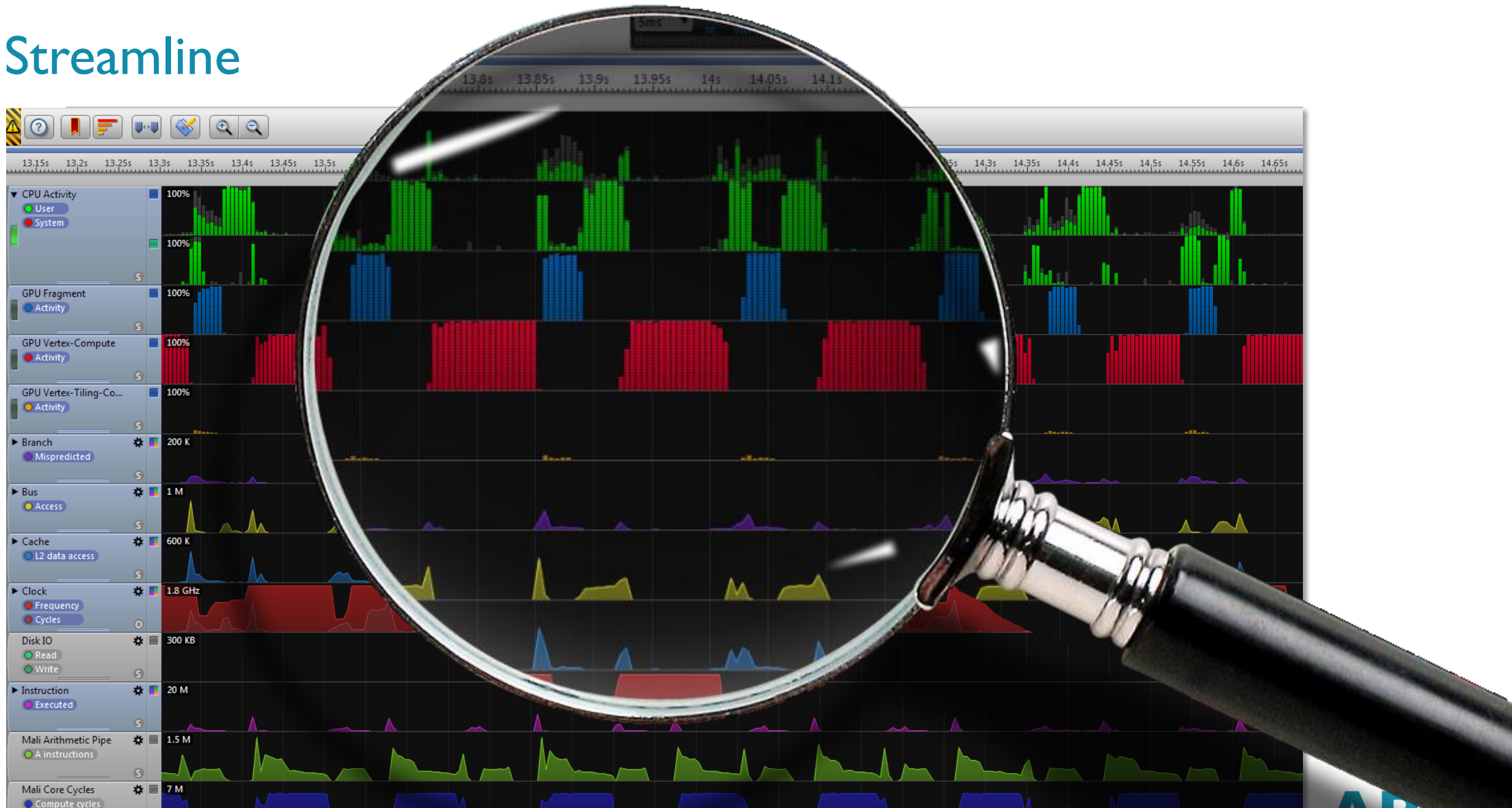
Example: Complex Computer Vision Application



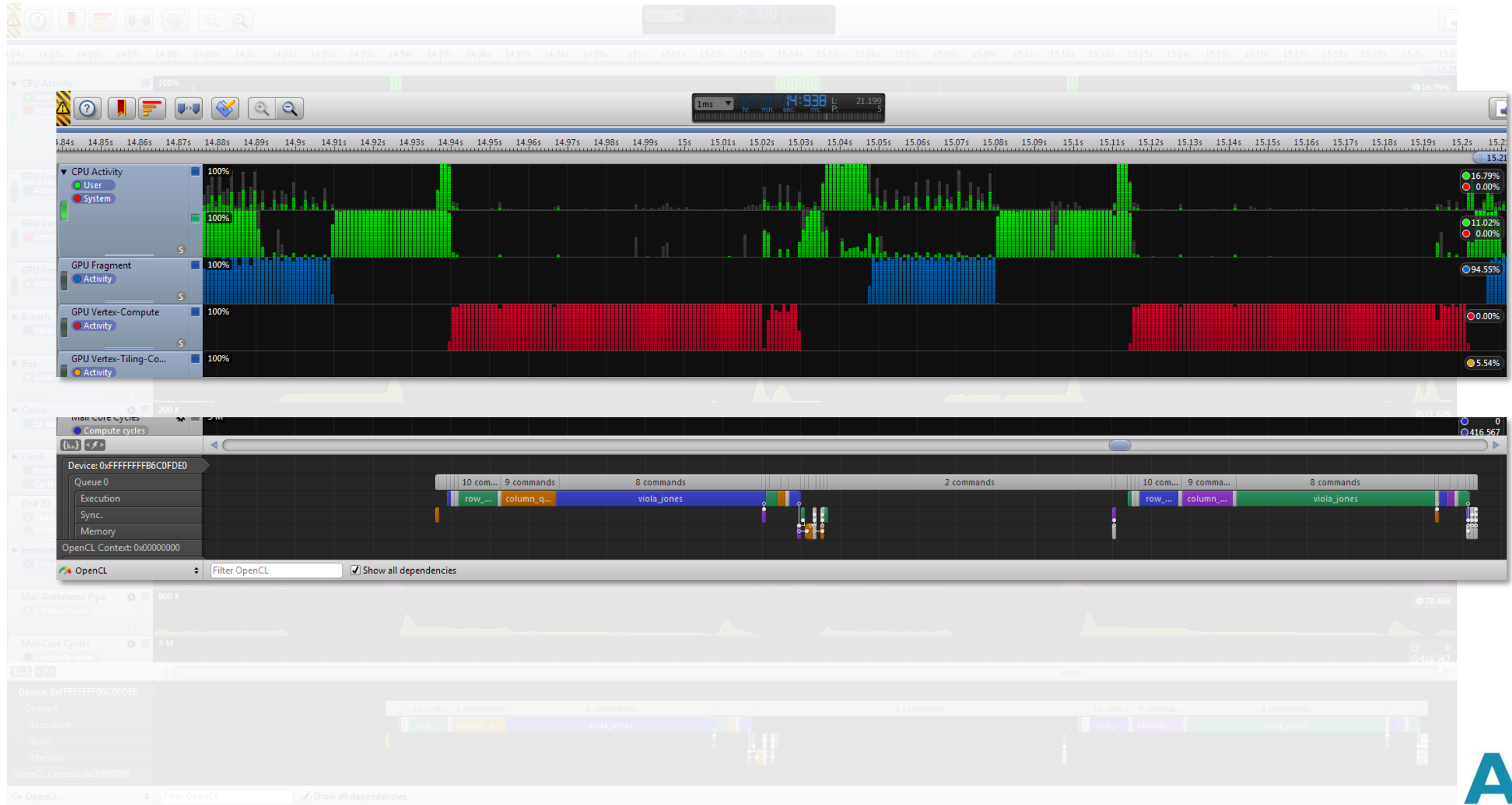
Lane and Car Detection



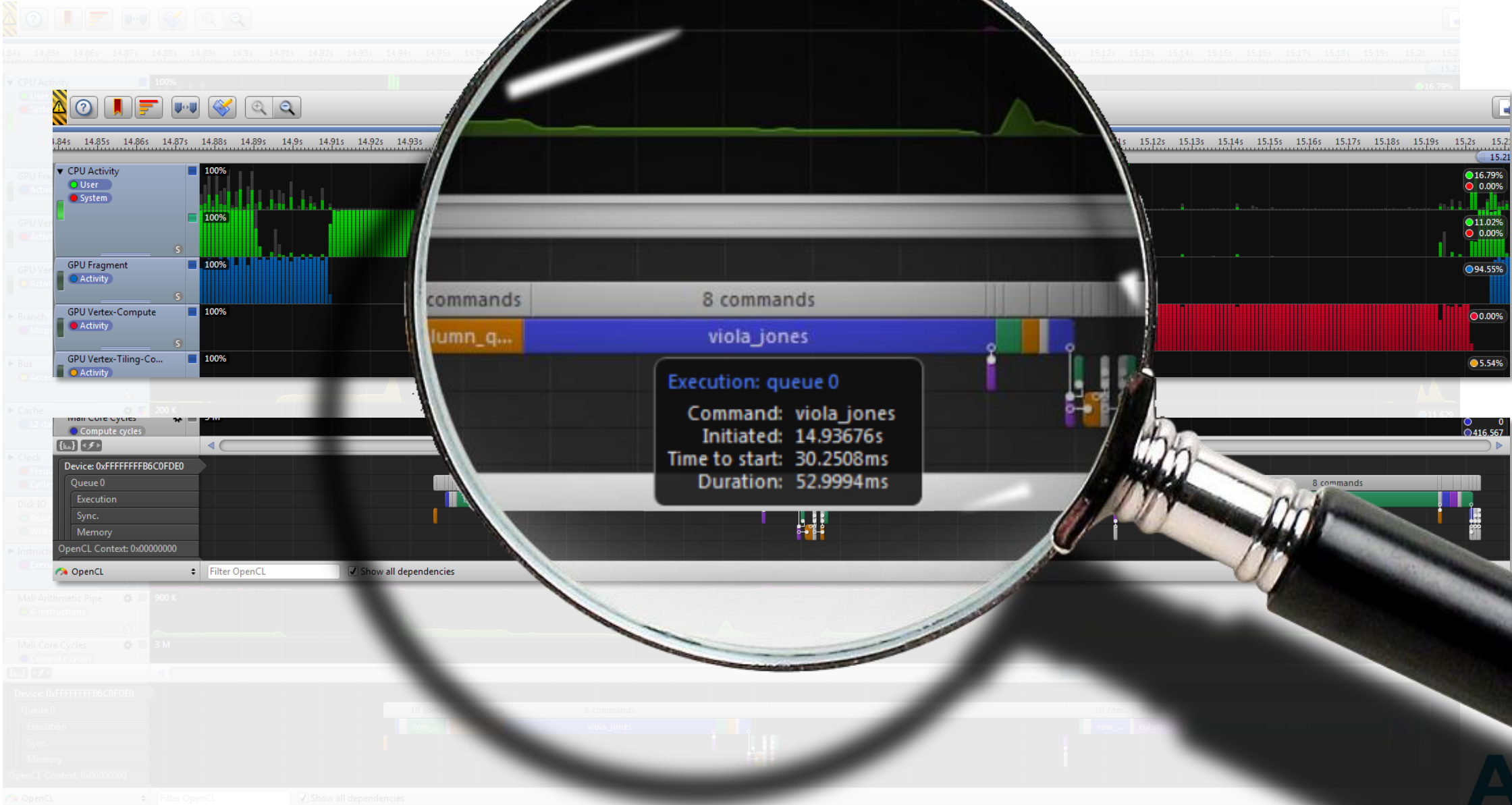
Streamline



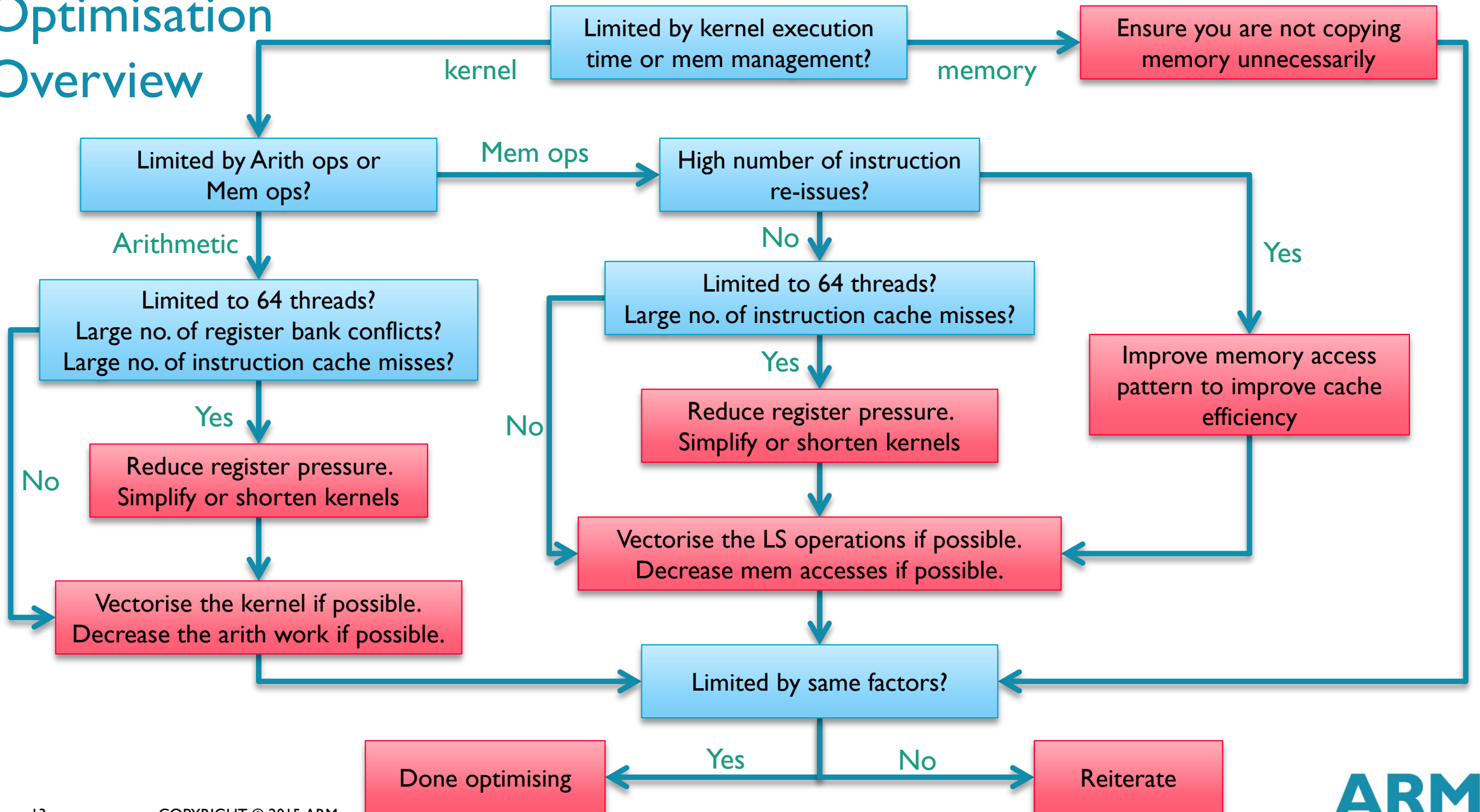
Streamline: OpenCL Timeline



Streamline: OpenCL Timeline



Optimisation Overview



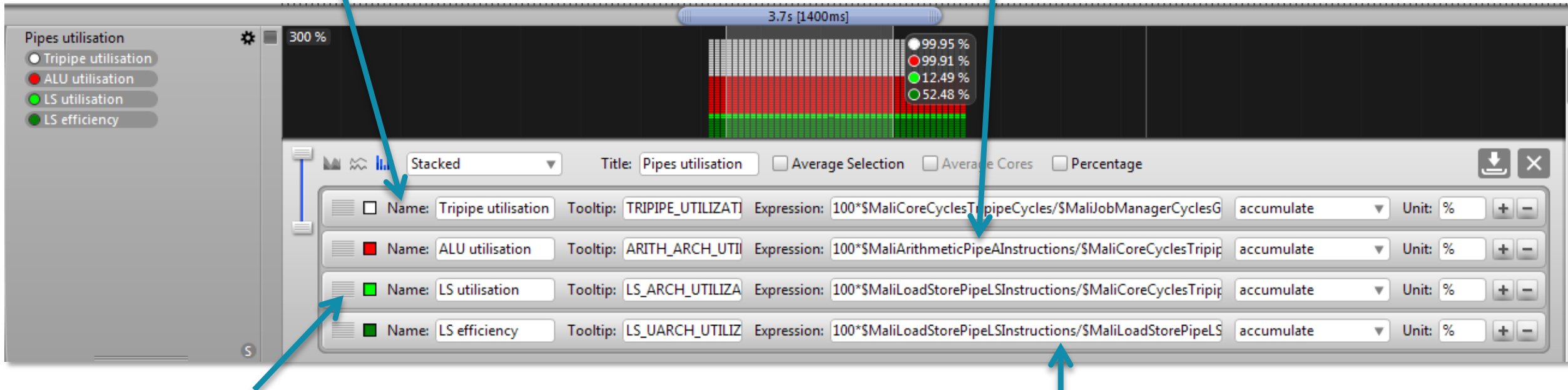
Deriving Meaning from Hardware Counters

- Counters on their own usually don't mean a huge amount
- Combining counters is more useful
 - Comparing values to determine limiting pipes
 - Calculating more meaningful values from multiple values
- New graph traces can be added from these counters
 - ...and become an integral part of the timeline

Custom Charts: Bringing Counters Together

$100 * \$MaliCoreCyclesTripipeCycles / \$MaliJobManagerCyclesGPUCycles$

$100 * \$MaliArithmeticPipeAInstructions / \$MaliCoreCyclesTripipeCycles$



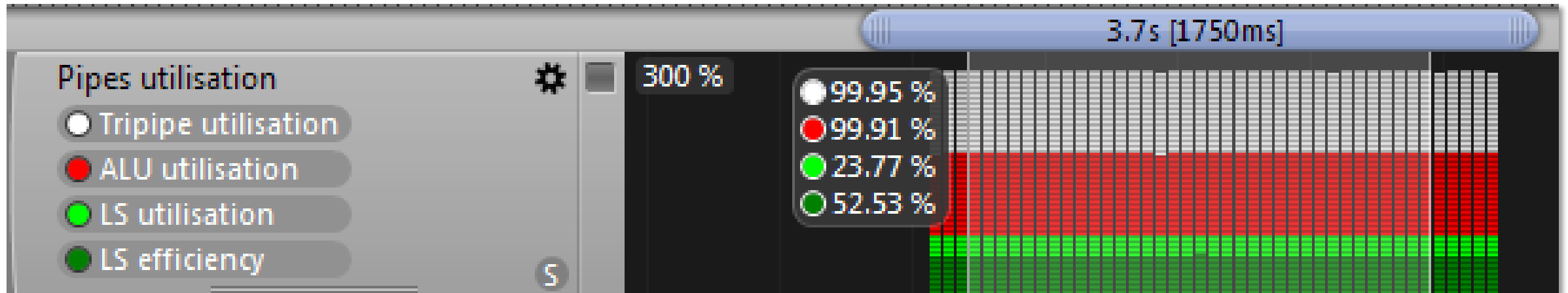
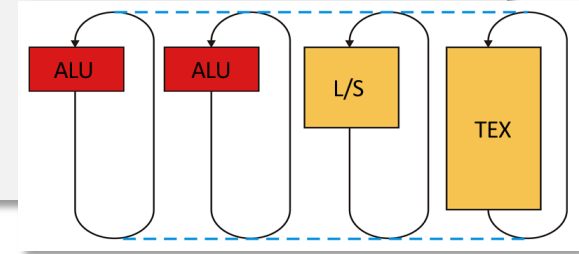
$100 * \$MaliLoadStorePipeLSInstructionIssues / \$MaliCoreCyclesTripipeCycles$

$100 * \$MaliLoadStorePipeLSInstructions / \$MaliLoadStorePipeLSInstructionIssues$

ALU Bound kernel

- One load
- One store
- “n” ALU operations

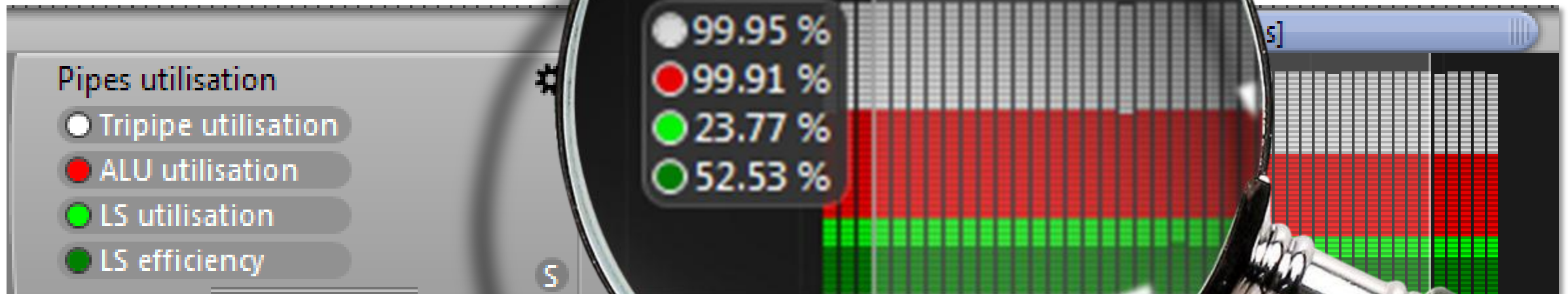
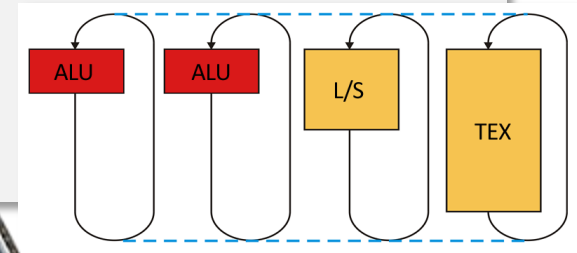
```
__kernel void kernel_alu_bound( global float* arr, uint n)
{
    float value = arr[get_global_id(0)];
    for(uint i = 0; i < n; i++)
    {
        value += sin(value);
    }
    arr[get_global_id(0)] = value;
}
```



ALU Bound kernel

- One load
- One store
- “n” ALU operations

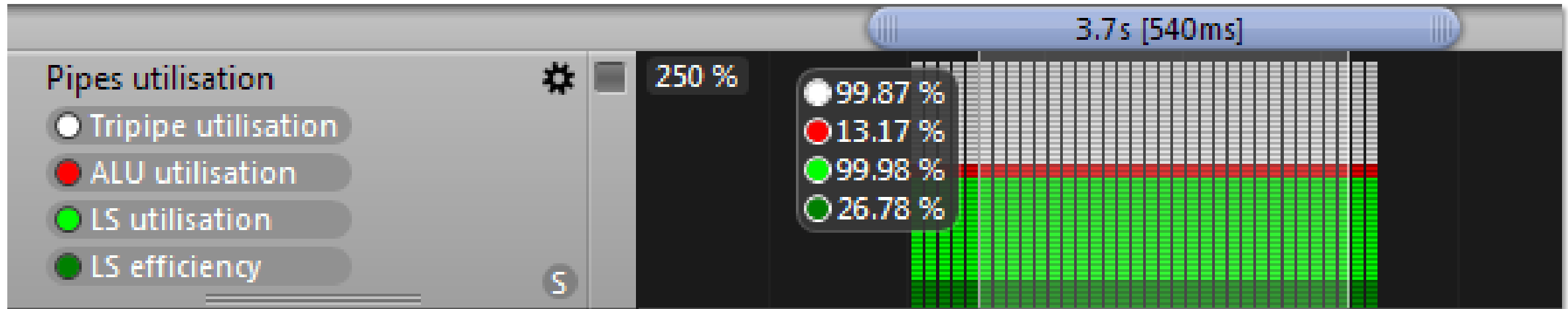
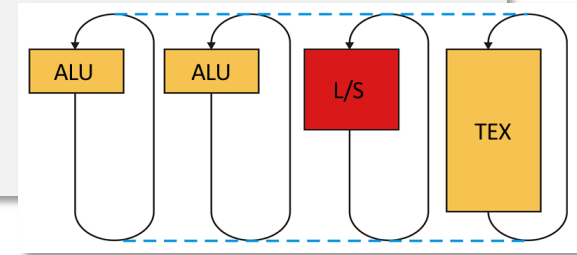
```
__kernel void kernel_alu_bound( global float* arr, uint n)
{
    float value = arr[get_global_id(0)];
    for(uint i = 0; i < n; i++)
    {
```



L/S Bound kernel

- One load
- One store
- No ALU operation

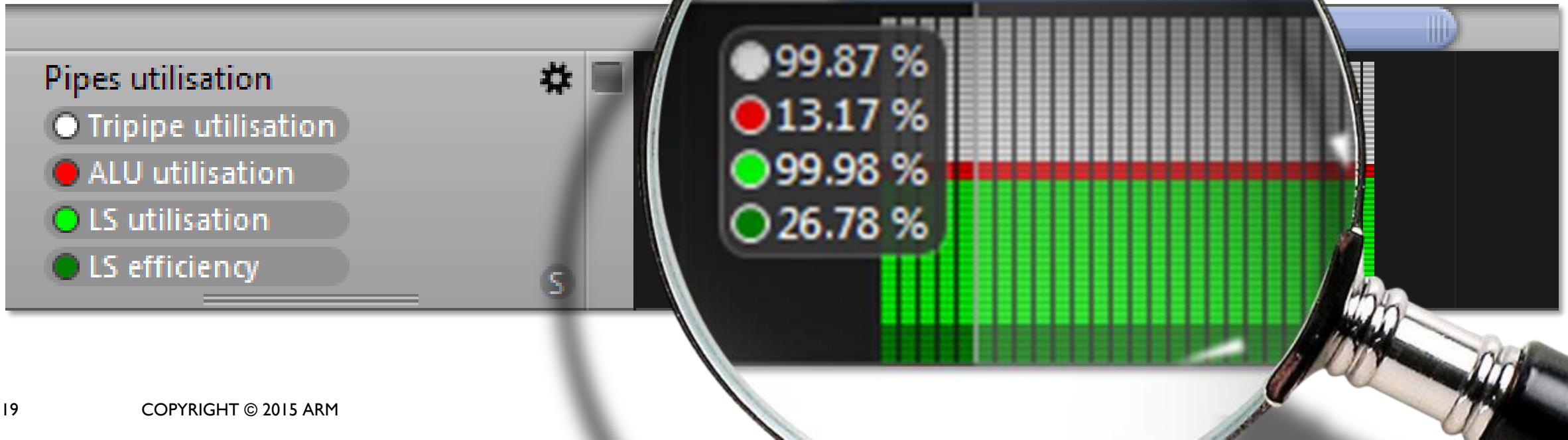
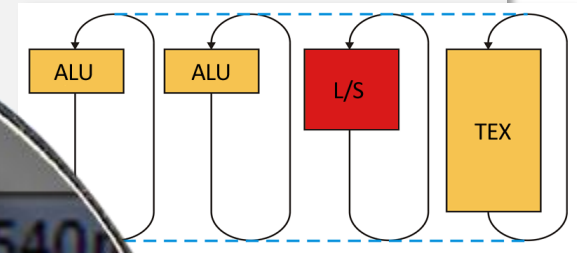
```
__kernel void kernel_memcpy( global float *a, global float *b)
{
    float4 v = vload4(0, a);
    vstore4(v, get_global_id(0), b);
}
```



L/S Bound kernel

- One load
- One store
- No ALU operation

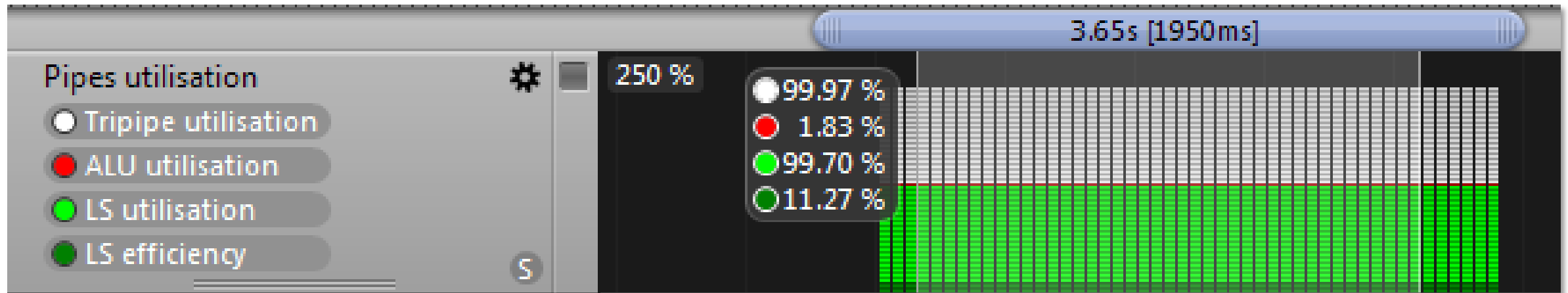
```
__kernel void kernel_memcpy( global float *a, global float *b)
{
    float4 v = vload4(a, ...);
    vstore4(v, b, ...);
}
```



Cache misses

- One byte read every 64 bytes
- One byte written every 64 bytes
- Really bad cache utilisation!

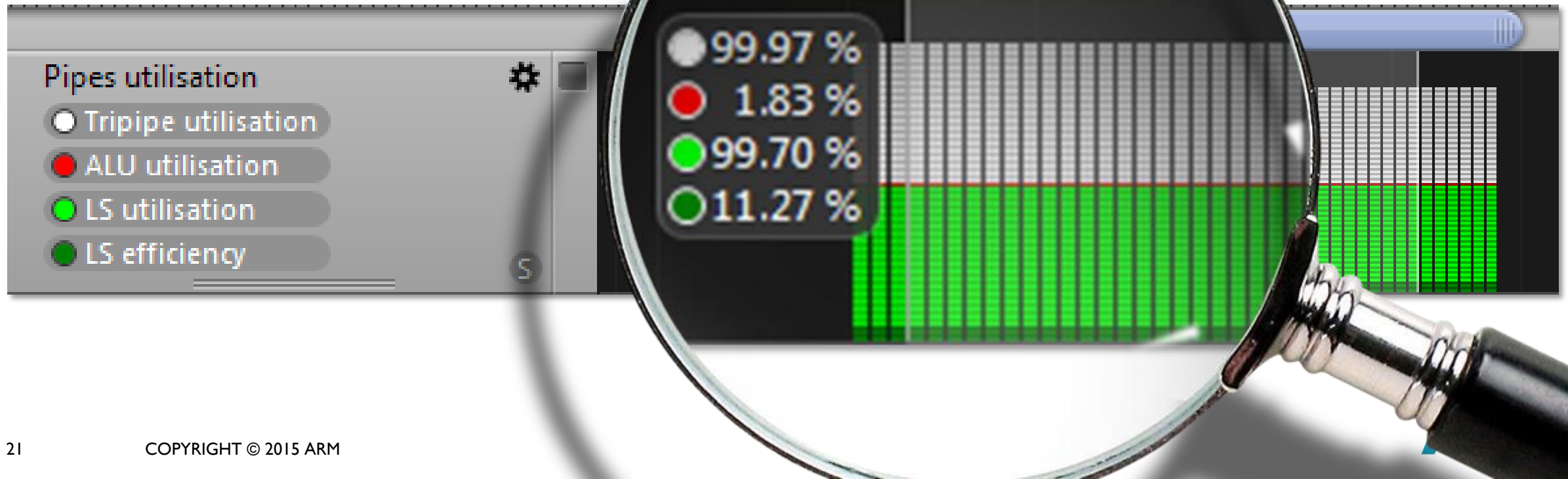
```
__kernel void kernel_cache_misses( global uchar *a,  
                                   global uchar *b)  
{  
    b[64 * get_global_id(0)] = a[64 * get_global_id(0)];  
}
```



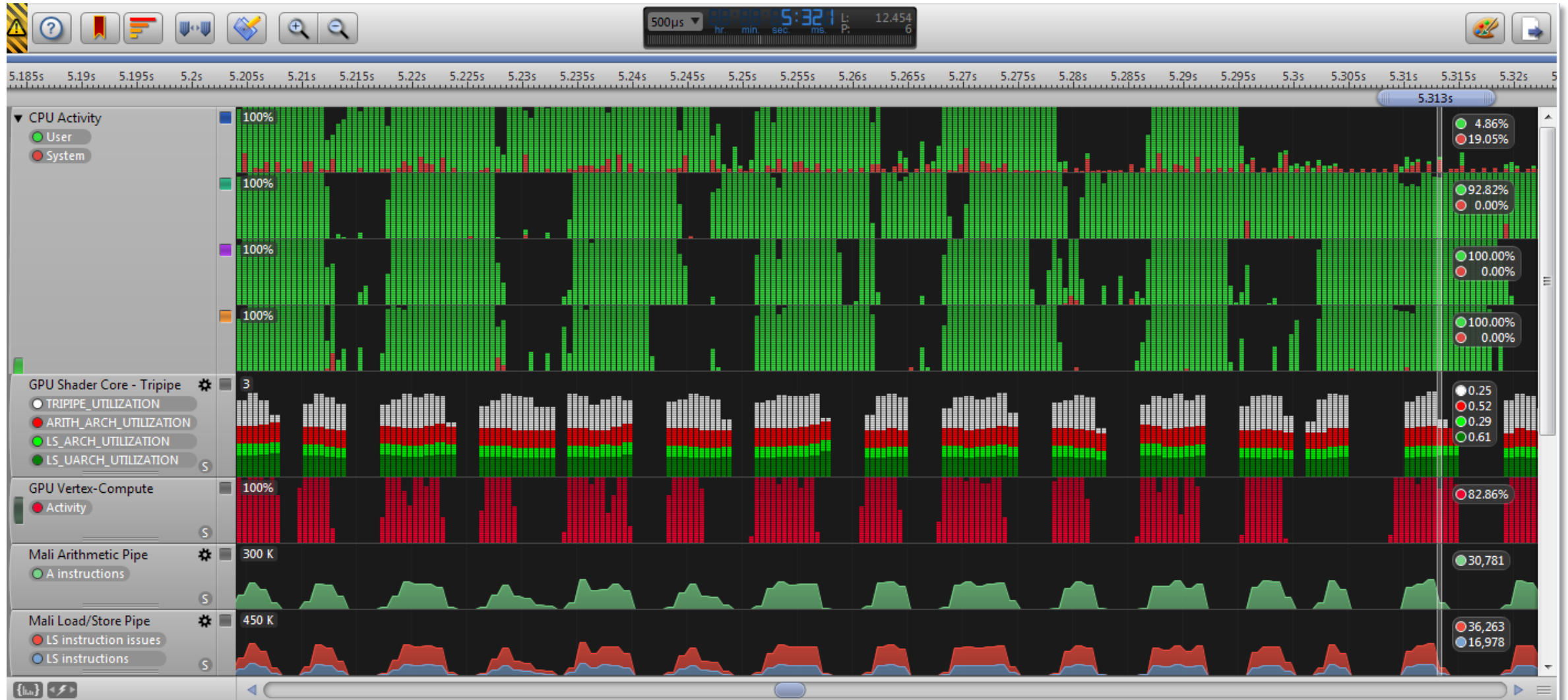
Cache misses

- One byte read every 64 bytes
- One byte written every 64 bytes
- Really bad cache utilisation!

```
__kernel void kernel_cache_misses( global uchar *a,  
                                   global uchar *b)  
{  
    b[64 * get_global_id(0)] = a[64 * get_global_id(0)];  
}
```



What does good whole-system optimisation look like?



Conclusions

- Computer vision applications need careful optimisation
 - Understanding your system as a whole is a vital first step
 - Understanding each individual processor core type is the next
- Use tools to measure hardware counters across the entire platform
 - Whole-system views of the relative performance of heterogeneous architectures are invaluable
 - Allows you to decide where there is capacity to move workloads
 - And how to target optimisations by exposing the limiting component within individual cores
 - Ideally, use these tool throughout the development process, not just at the end
- The Mali Ecosystem is making GPU Compute a reality today
 - ARM enables developers with platforms, drivers, tools and support
 - Industry leaders take advantage of ARM Mali GPU capabilities to innovate and deliver
 - Be one of them!

Tomorrow at the EVA Summit, 4pm:

“Understanding the Role of Integrated GPUs in Vision Applications”, Roberto Mijat

Ecosystem Resources

- www.malideveloper.com
 - Download guides, papers, tools (including DS-5 Streamline), etc.
- <http://community.arm.com/welcome>
 - Community forums, blogs and more
- malidevelopers@arm.com
 - Graphics and GPU Compute developer support
- <http://malideveloper.arm.com/develop-for-mali/opengl-renderscript-tutorials/>
 - A range of video and written tutorials for GPU Compute
- <http://malideveloper.arm.com/develop-for-mali/features/mali-t6xx-gpu-user-space-drivers/>
 - ARM Mali-T600 series GPU user-space binary drivers available for download
- Linaro BSP now available with Mali-T600 series GPU support

Measuring the Whole System

Holistic Profiling of CPU and GPU for Optimal Vision Applications on ARM Platforms

Tim Hartley