Profiling OpenCL™ Kernels Using Wavefront Occupancy

Perhaad Mistry & Budi Purnomo
Advanced Micro Devices, Inc.

AGENDA

Existing profiling techniques and motivation for improvements to OpenCL tools

GCN and Wavefront Occupancy

Applying Wavefront occupancy in Radeon GPU Profiler

Future Work

Existing Profiling Techniques for OpenCL Developers

- Performance counters - Aggregate data – no indication of what happened in the kernel over time
  - Limitation – No Workgroup Scheduling across shader engines.
  - Limitation – Cannot see stalls in kernels waiting for memory or correlate to source

Performance Counters

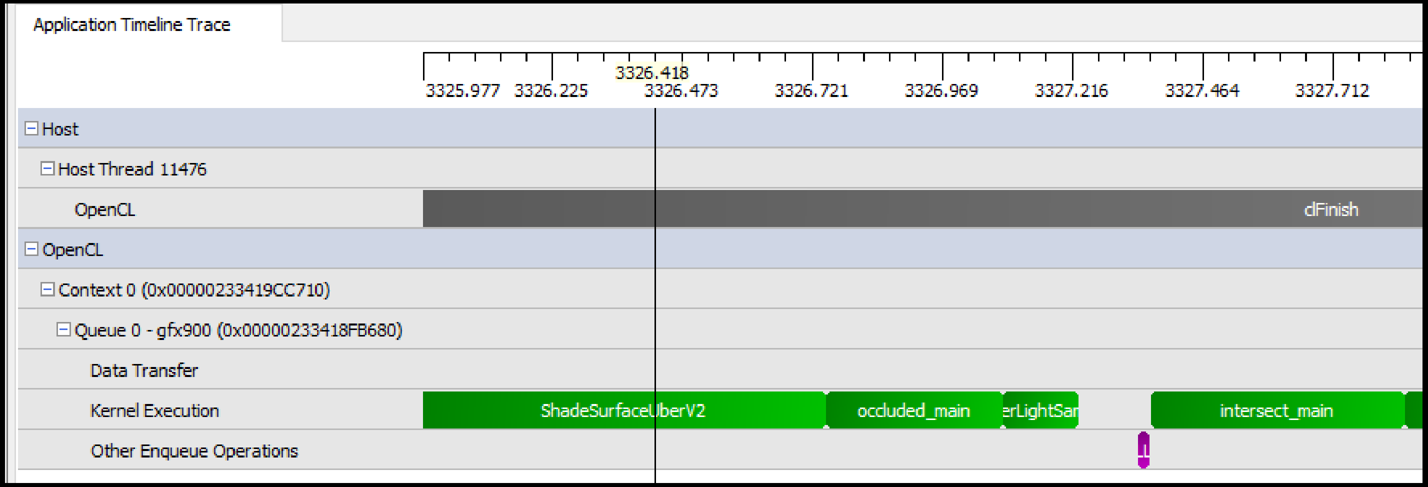
☑ Show Zero Columns

	Method	ExecutionOrder	ThreadID	CallIndex	GlobalWorkSize	WorkGroupSize	Time	LocalMemSize	VGPRs	SGPRs	KernelOccupancy	ScratchRegs	Wavefronts	VALUInsts	SALUIns
1	GenerateTileDomain k1 Baffin1	1	21084	1501	{ 1280 768 1}	{ 16 16 1}	0.06128	0	8	22	100	0	15360	19	10
2	PerspectiveCamera GeneratePaths k2 Baffin1	2	21084	1561	{ 983040 1 1}	{ 64 1 1}	0.66784	0	23	27	100	0	15360	247	43
3	InitPathData k3 Baffin1	3	21084	1723	{ 983040 1 1}	{ 64 1 1}	0.50656	0	7	19	100	0	15360	22	4
4	intersect main k4 Baffin1	4	21084	1794	{ 983040 1 1}	{ 64 1 1}	1.18992	4096	54	31	50	0	15360	836.15	271.56
5	FilterPathStream k5 Baffin1	5	21084	1830	{ 983040 1 1}	{ 64 1 1}	0.79840	0	11	19	100	0	15360	46.30	23
6	scan exclusive part int4 k6 Baffin1	6	21084	1863	{ 122880 1 1}	{ 64 1 1}	0.10224	256	28	23	100	0	1920	155	48
7	scan exclusive part int4 k6 Baffin1	7	21084	1872	{ 256 1 1}	{ 64 1 1}	0.00720	256	28	23	100	0	4	165	55
8	scan exclusive int4 k7 Baffin1	8	21084	1880	{ 64 1 1}	{ 64 1 1}	0.00672	256	27	22	100	0	1	182	68

Source: CodeXL

Existing Profiling Techniques for OpenCL Developers

- Timestamps with API interception don't provide visibility into work the driver did.
- Limitation - Barriers inserted to flush caches
  - Cause - Applications scheduling consecutive kernels with data dependencies
- Limitation - Dispatches batched by driver into command buffers
  - Cause - Excessive synchronization like clFinish() in applications



Source: CodeXL

Our Design Goals for Improved OpenCL Tools

- Enable new optimizations for OpenCL applications
- Make optimization agnostic across architecture generations

- Needs to be applicable to graphics and compute workloads
  - A single dispatch like compute
  - A game where multiple shaders are in flight at any point in time

Define actionable metrics that allow us to quantify performance **over any time interval** as it executes on a device. Example: Wavefront Occupancy

AGENDA

Existing profiling techniques and motivation for improvements to OpenCL tools

GCN and defining Wavefront Occupancy

Applying Wavefront occupancy in Radeon GPU Profiler

Future Work

GCN Architecture

X Shader Engines per Chip with Y Compute Units per Shader Engine

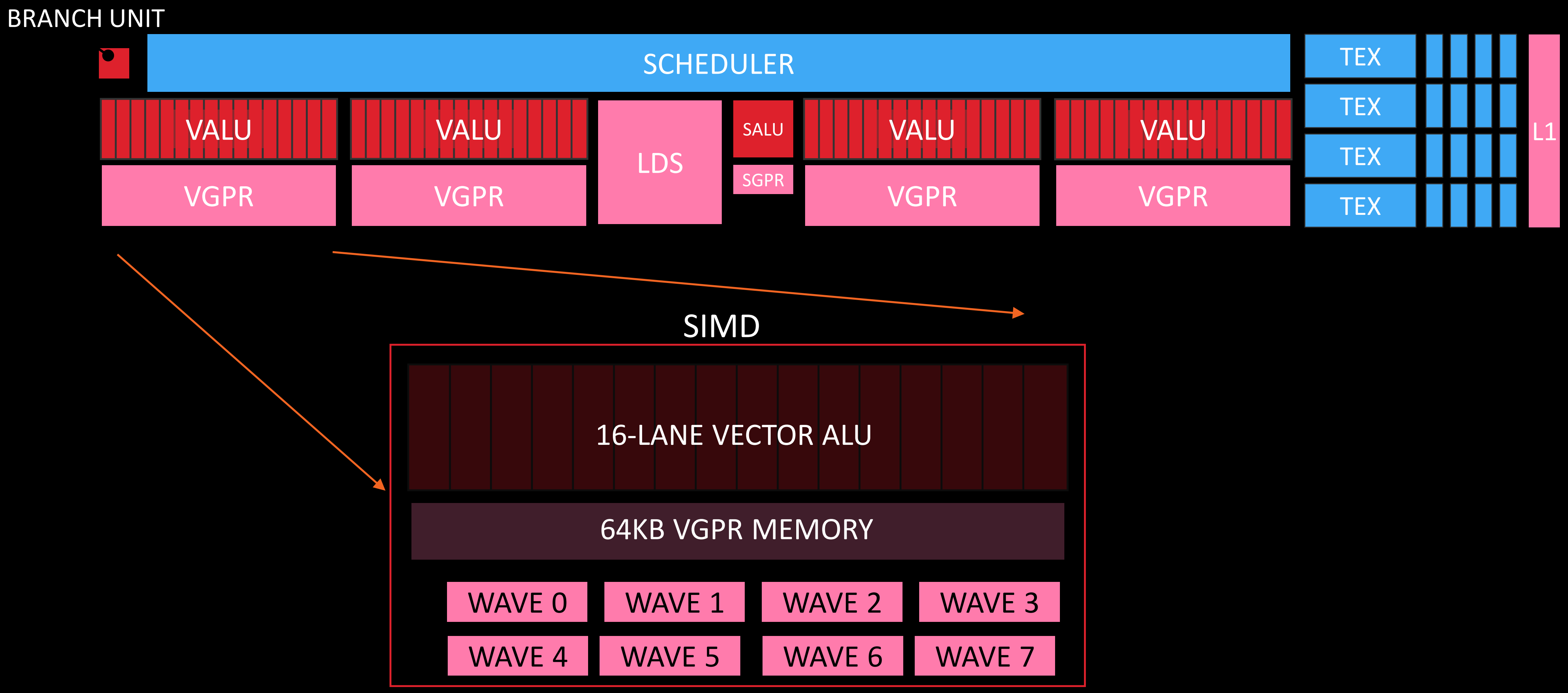
GCN Architecture

X Shader Engines per Chip with Y Compute Units per Shader Engine

For OpenCL developers

GCN Compute Unit

4 SIMD Units per CU



What is wavefront occupancy?

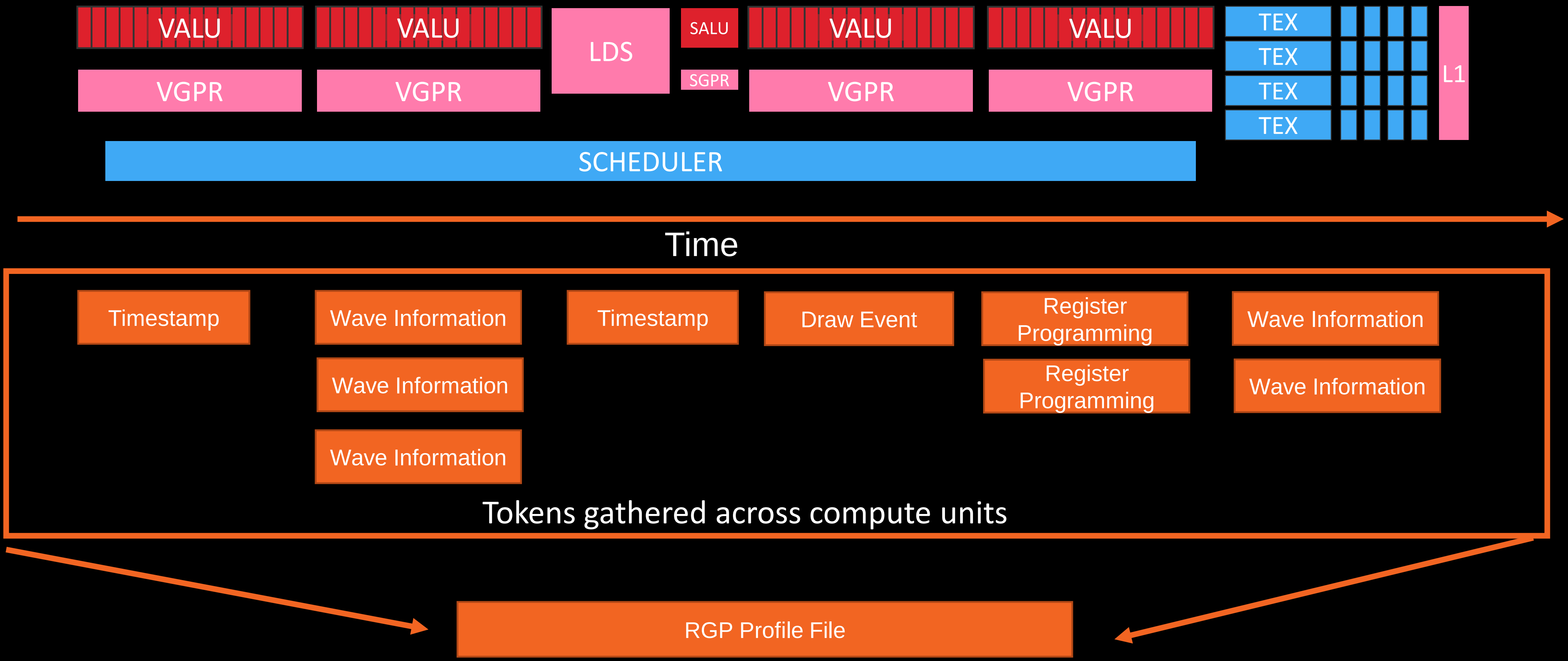
8 Wave Slots Per SIMD on RX480

8 Waves Per SIMD	SIMD	Wave	Wave	Wave	Wave	Wave	Wave	Wave	Wave	100%
4 Waves Per SIMD	SIMD	Wave	Wave	Wave	Wave	Empty	Empty	Empty	Empty	50%
2 Waves Per SIMD	SIMD	Wave	Wave	Empty	Empty	Empty	Empty	Empty	Empty	25%
1 Wave Per SIMD	SIMD	Wave	Empty	Empty	Empty	Empty	Empty	Empty	Empty	12.5%

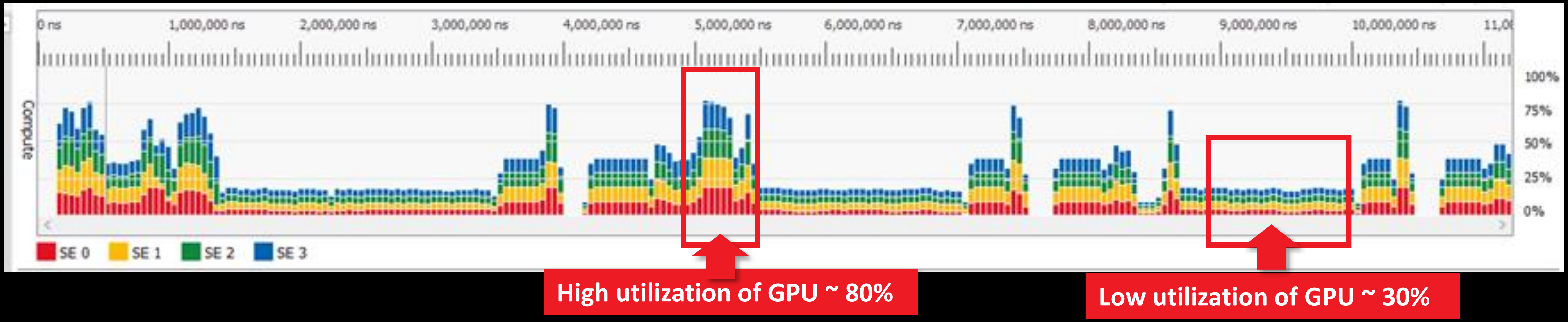
Measure of how close a SIMD is to its maximum wavefront capacity at a point in time

How do we calculate Wavefront Occupancy

Hardware support in AMD GCN compute units emits event tokens to GPU memory

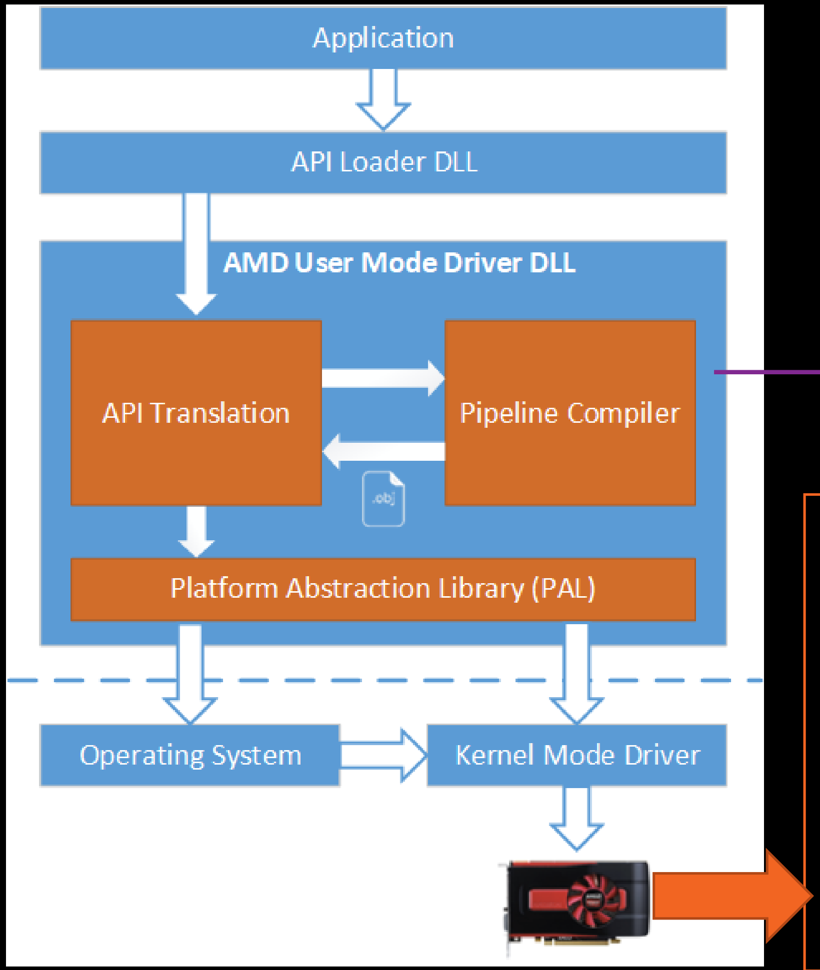


Wavefront Occupancy

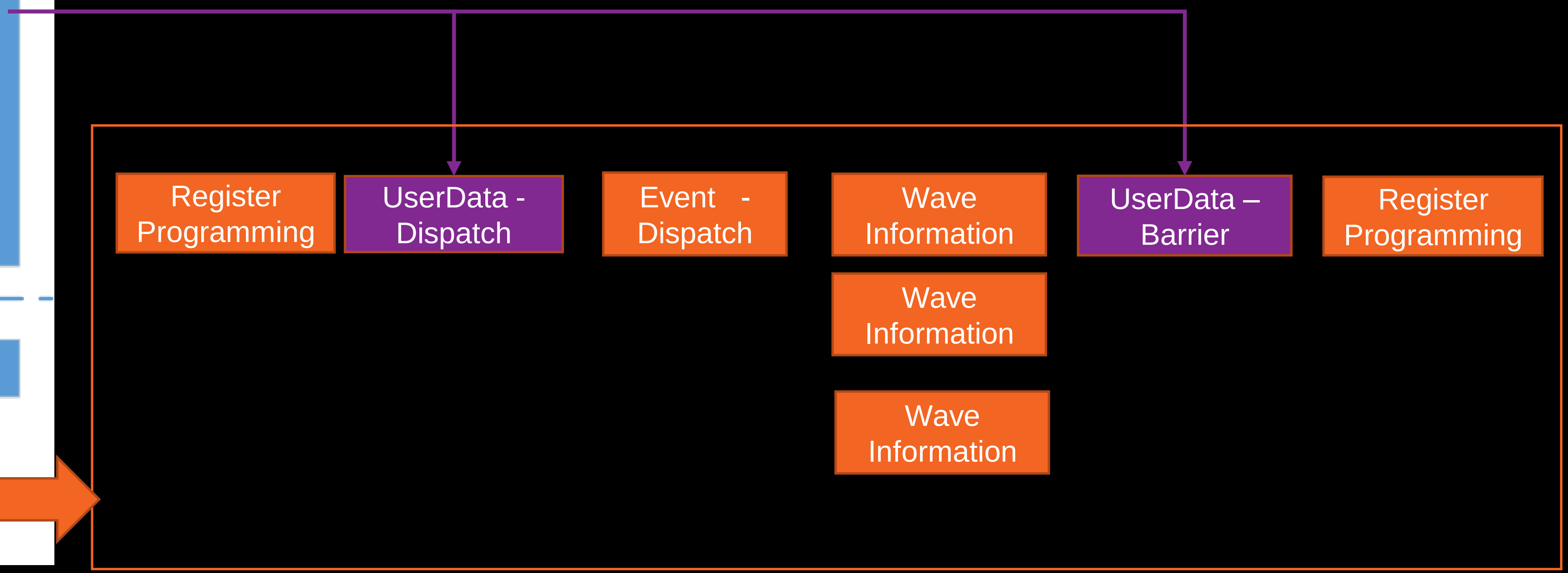


- Four shader engines doing OpenCL ray tracing
- Build a histogram showing how many waves were active in a time interval
  - Now get an idea of utilization of a GPU at any point in time
- However – How do we correlate this to a API call in OpenCL ?

Correlating wave fronts to API information

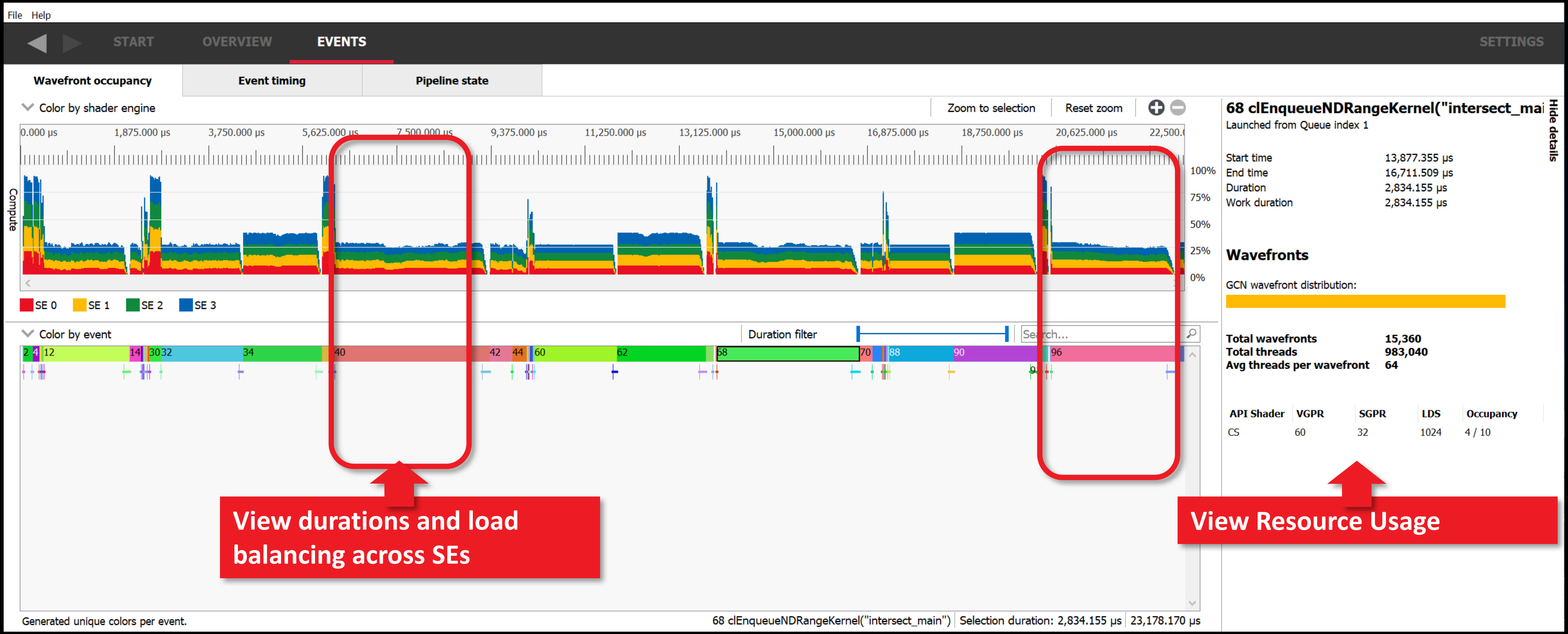


- User mode driver inserts data into hardware generated trace
- Allows adding API specific semantics
  - Example – OpenCL kernel names



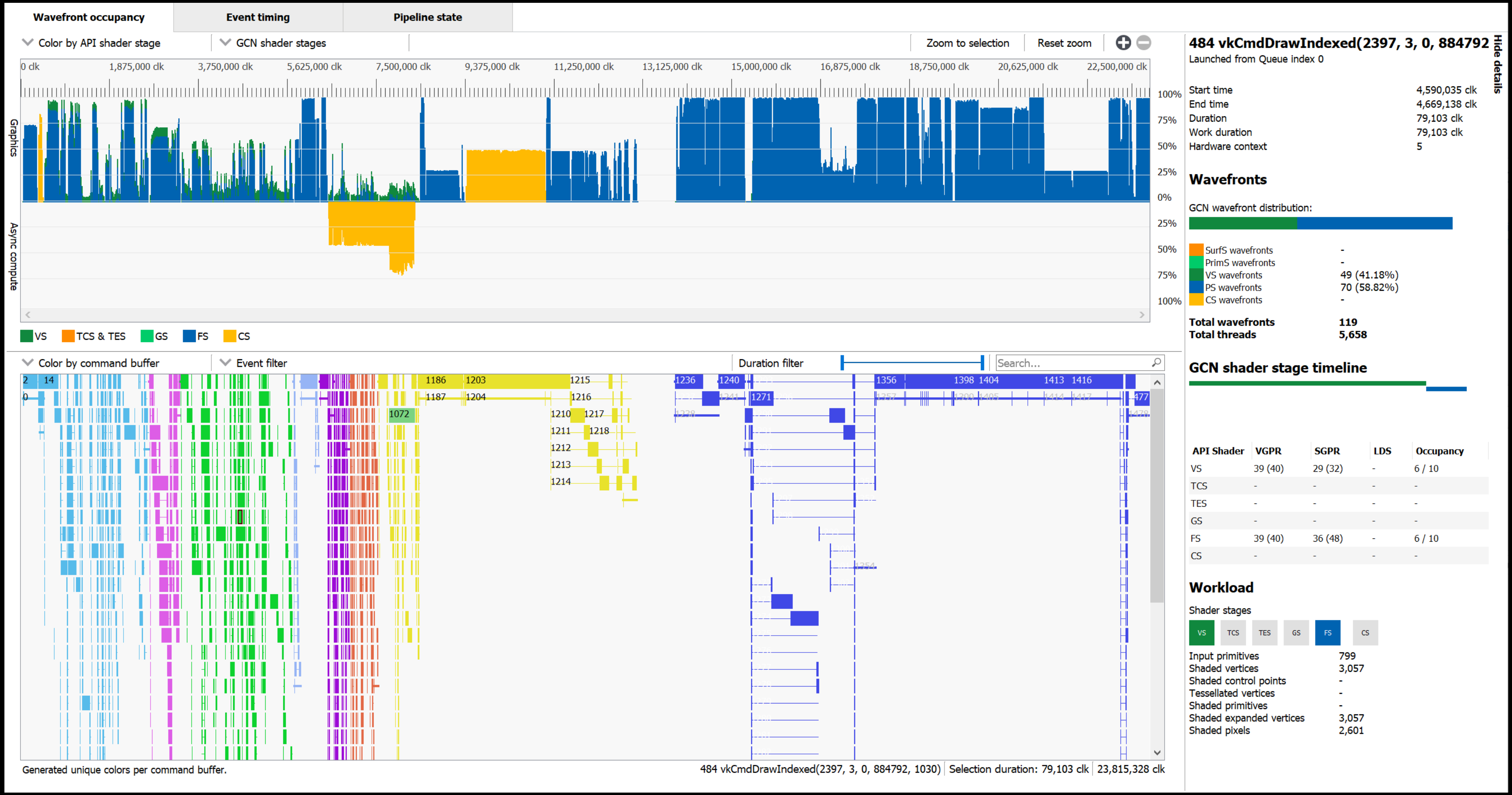
https://github.com/GPUOpen-Drivers/pal

Visualizing Wavefront Occupancy of OpenCL Applications



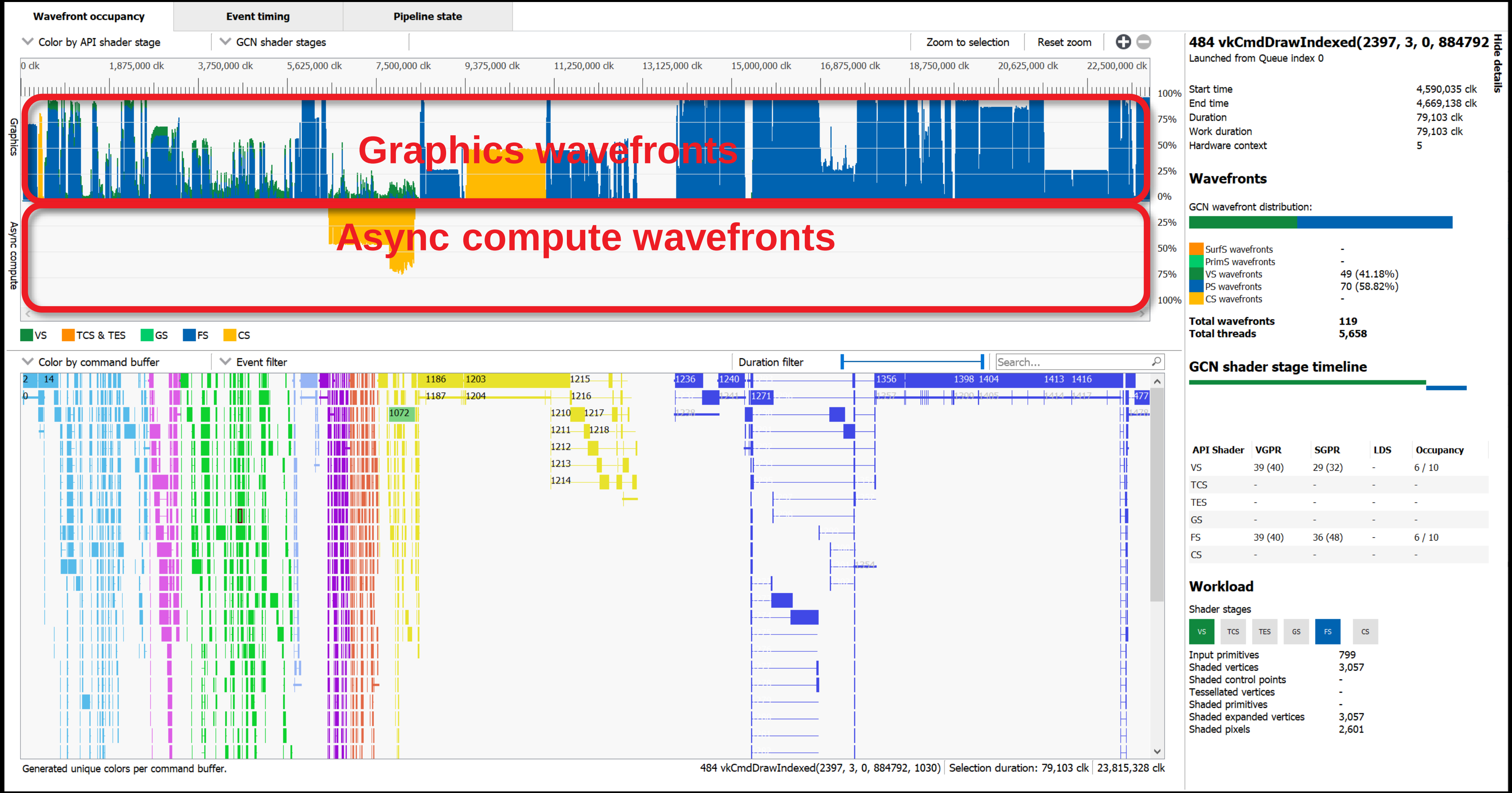
See how busy the GPU is

Visualizing Wavefront Occupancy of Vulkan Applications



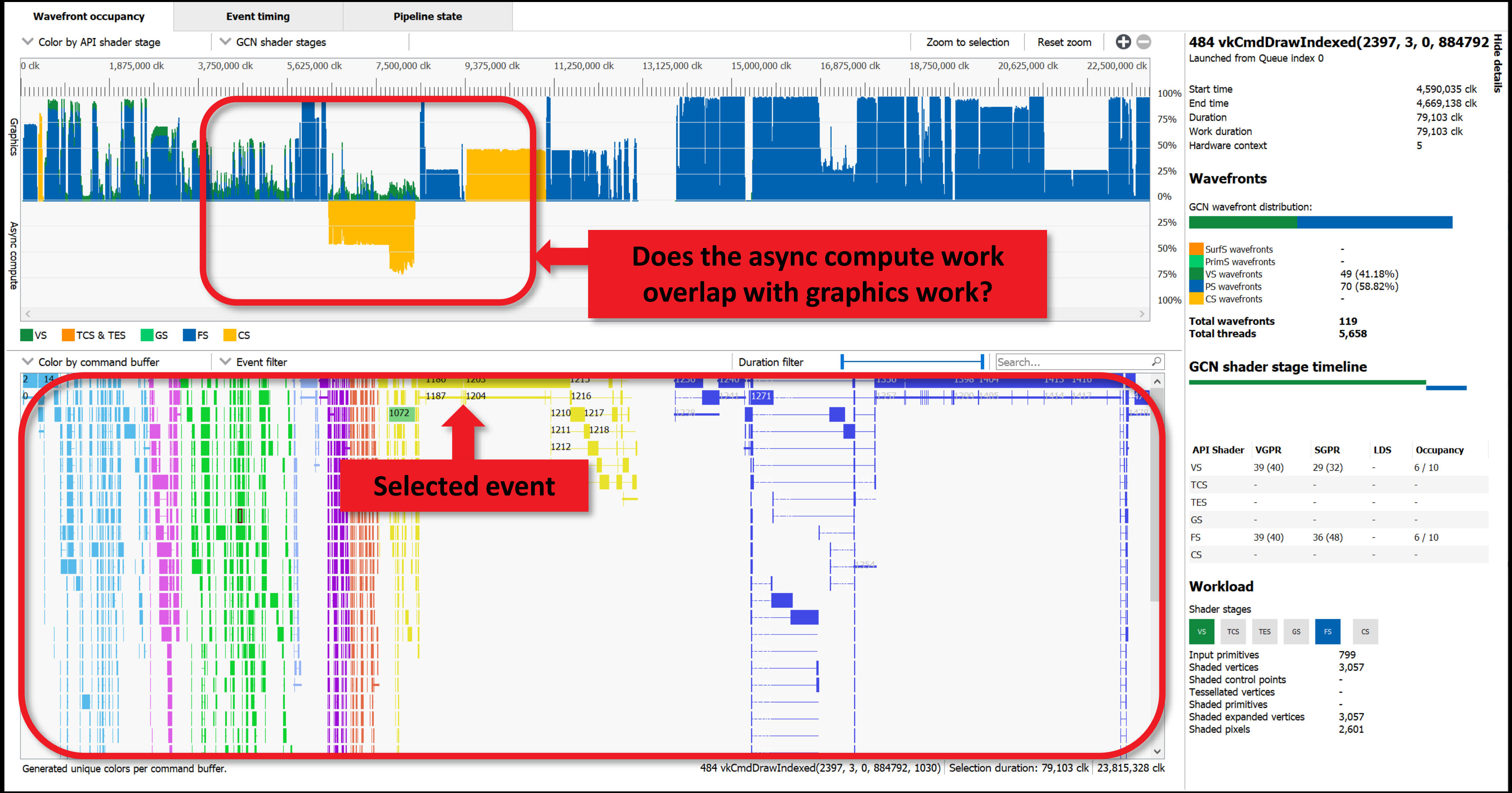
See how busy the GPU is

Visualizing Wavefront Occupancy of Vulkan Applications



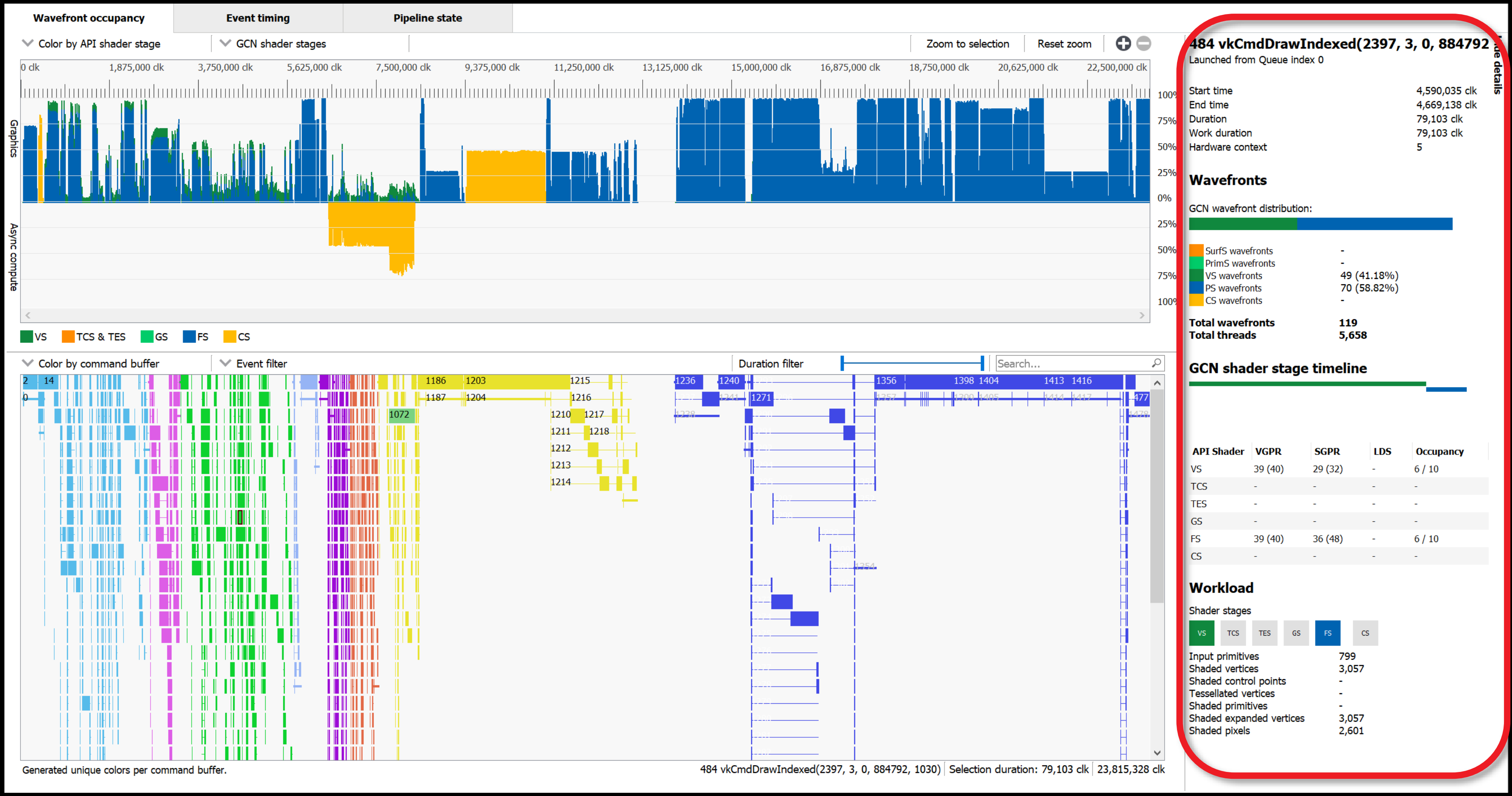
See how busy the GPU is

Visualizing Wavefront Occupancy of Vulkan Applications



See how busy the GPU is

Visualizing Wavefront Occupancy of Vulkan Applications



See how busy the GPU is

Understanding Application and Driver Interaction in OpenCL

- Application driver interaction visibility enabled by driver instrumentation and hardware support
- Driver adding Barriers when profiling enabled serializing dispatches

Shown as barriers in RGP

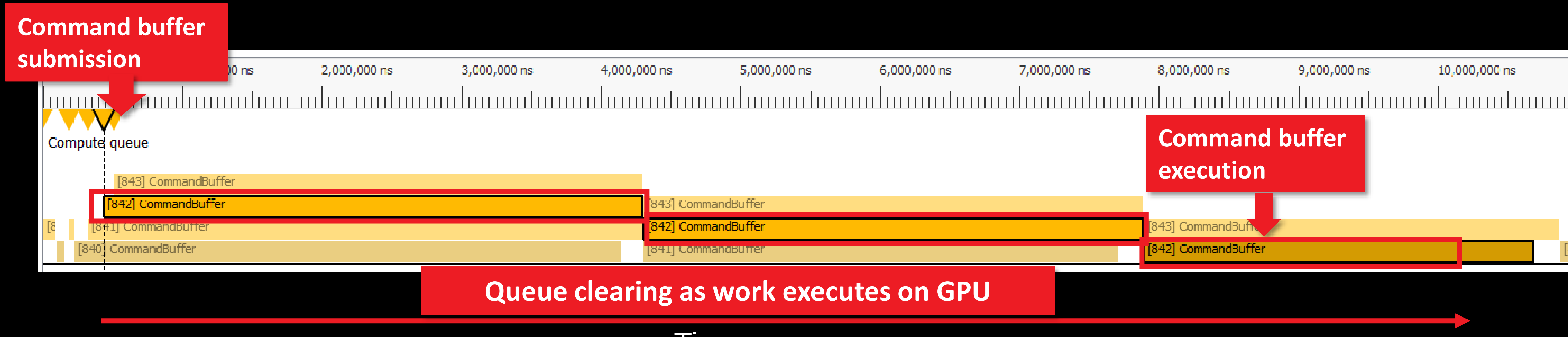
Caches

Barrier type and reason

Event Numbers	Duration	Drain Time	Stalls	Invalidated	Flushed	Barrier type	Reason for barrier
1	1,124 ns	753 ns	CS	K L1		DRIVER	Profiling control.
3	17,949 ns	745 ns	CS	K L1		DRIVER	Profiling control.
5	12,680 ns	726 ns	CS	K L1		DRIVER	Profiling control.
7	10,324 ns	780 ns	CS	K L1		DRIVER	Profiling control.
9	10,334 ns	801 ns	CS	K L1		DRIVER	Profiling control.
11	1,266 ns	739 ns	CS	K L1		DRIVER	Profiling control.

Understanding Application and Driver Interaction in OpenCL

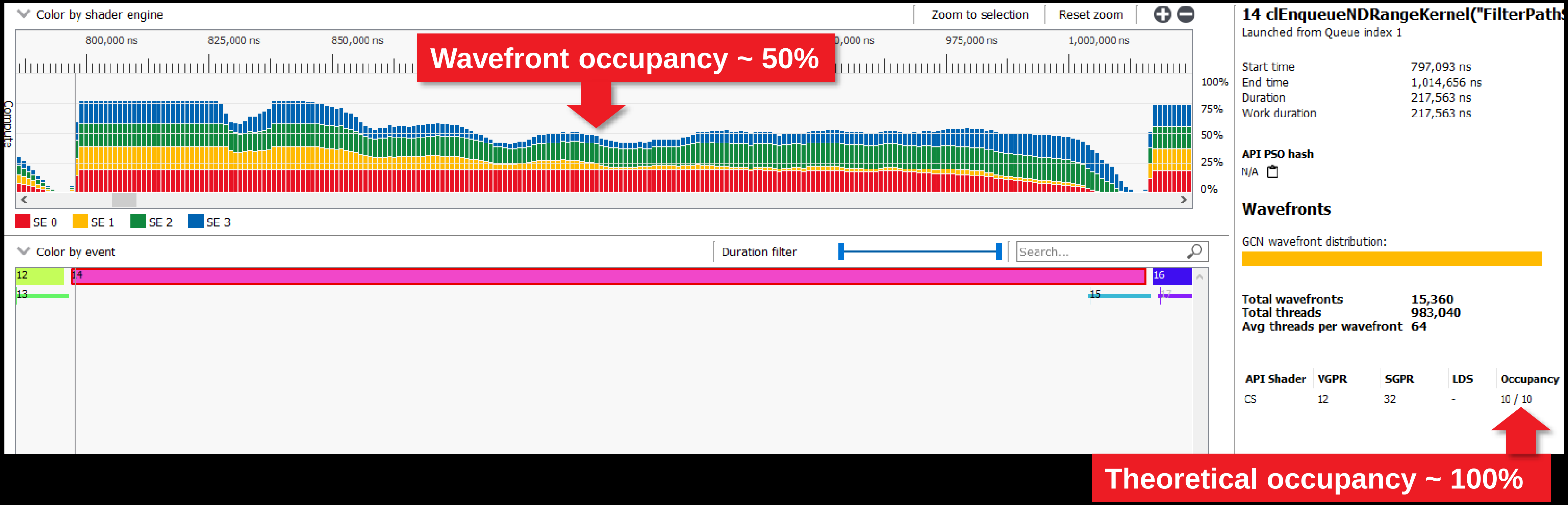
- Enabled by driver instrumentation and hardware support
- A command buffer is a batch of dispatches
- View grouping of OpenCL dispatches into command buffers
  - Example: Find the cost of clFlush()
  - View how long DMA takes and overlap



Time

Understanding Application and Driver Interaction in OpenCL

- Enabled by driver instrumentation and hardware support
- Difference between “Real” and “Theoretical Occupancy”



AGENDA

Existing profiling techniques and motivation for improvements to OpenCL tools

GCN and defining Wavefront Occupancy

Applying Wavefront occupancy in Radeon GPU Profiler

Future Work

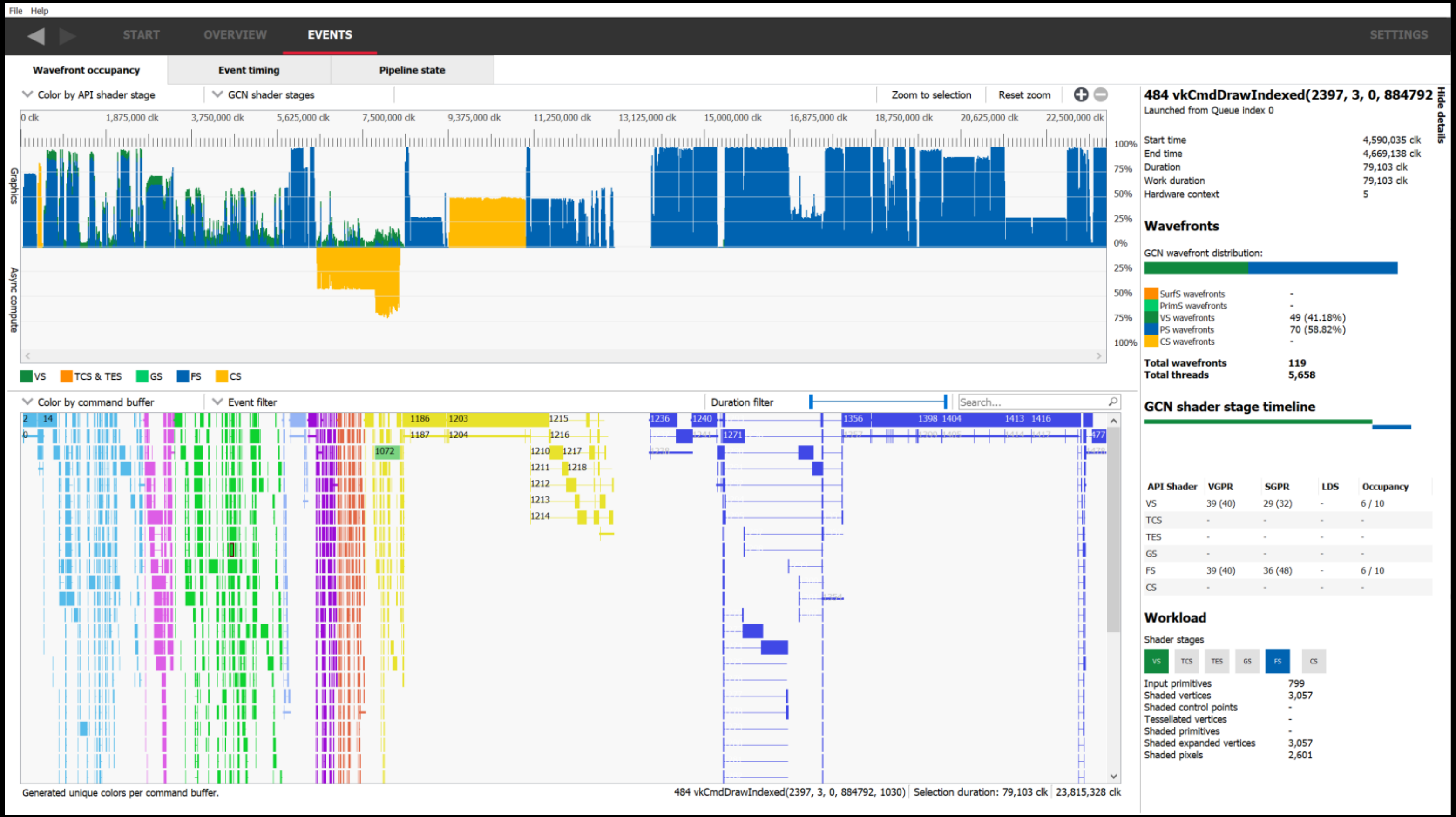
Radeon GPU Profiler

GPU performance analysis tool

- Visualizes GPU workloads to identify performance bottlenecks
- Bridge the gap between explicit APIs and GCN
- Built-in, hardware thread-tracing, allowing deep inspection of GPU workloads.

Designed to support

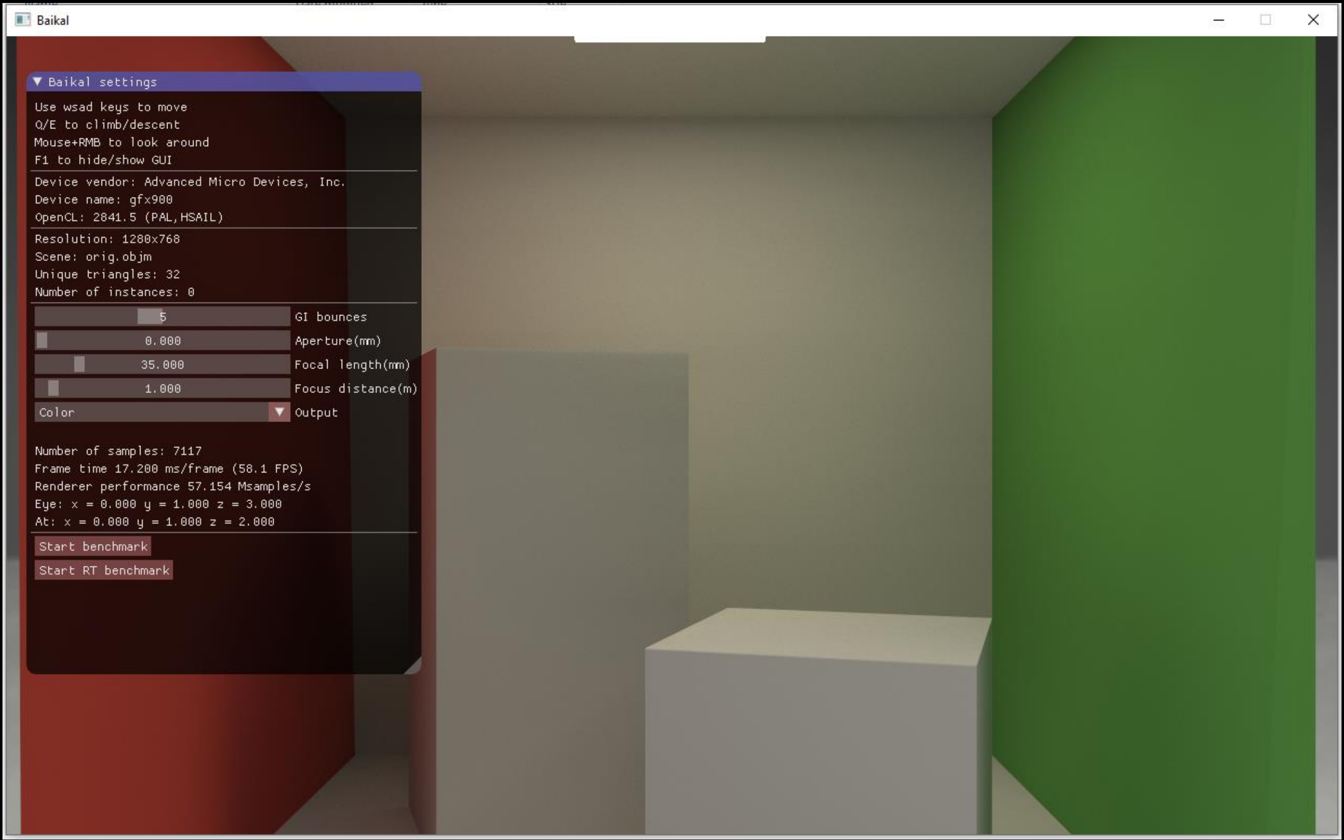
- Linux® and Windows®
- Vulkan®, DirectX® 12 and OpenCL



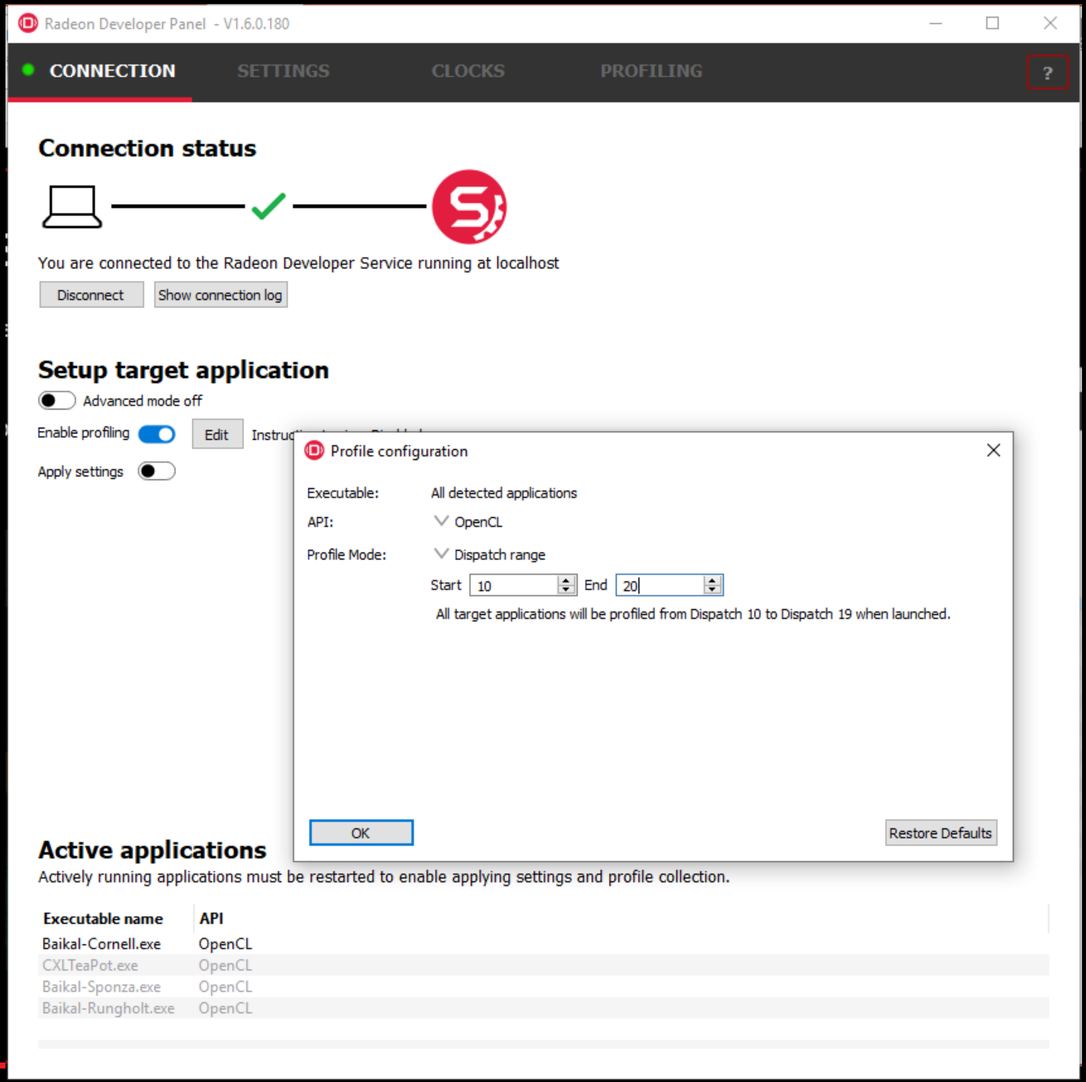
How does it work?

Launch the target application

(RGP support is built directly into the production driver)



Launch Developer Panel and Choose your dispatch range



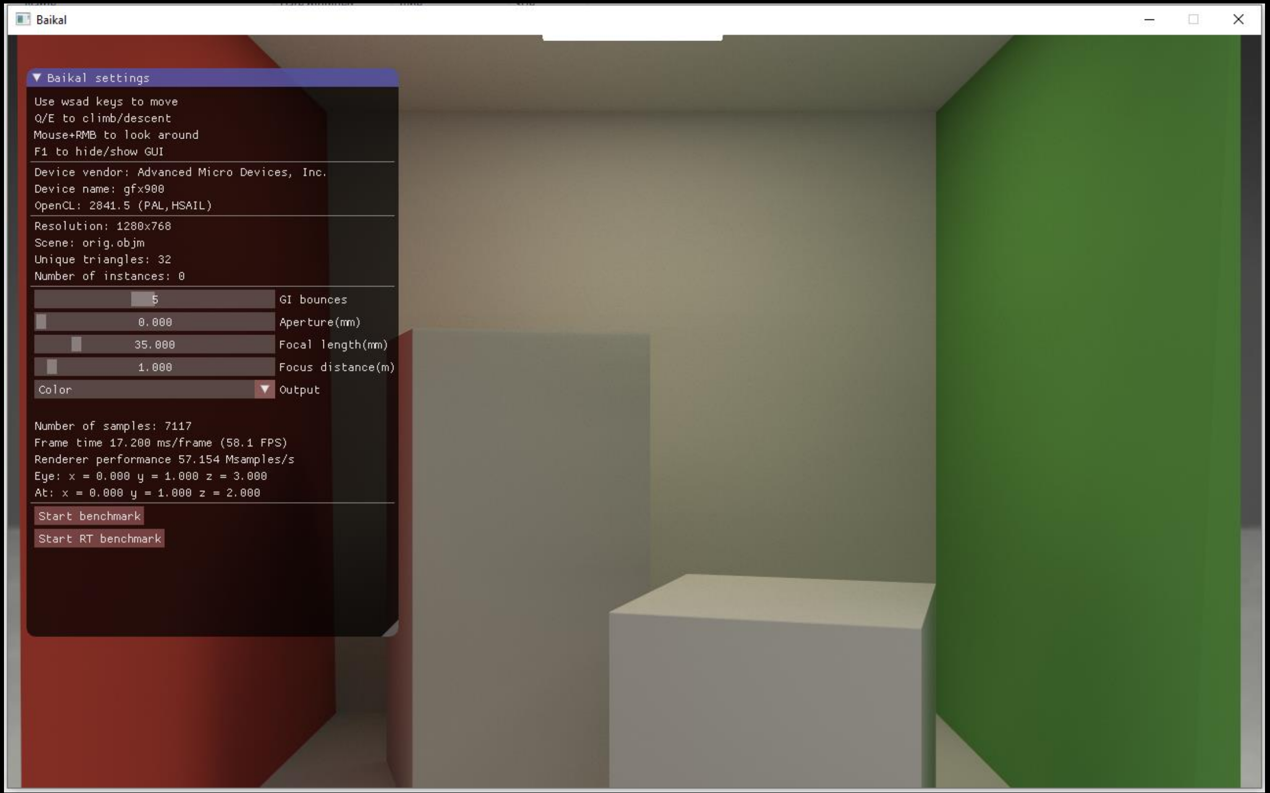
How does it work?

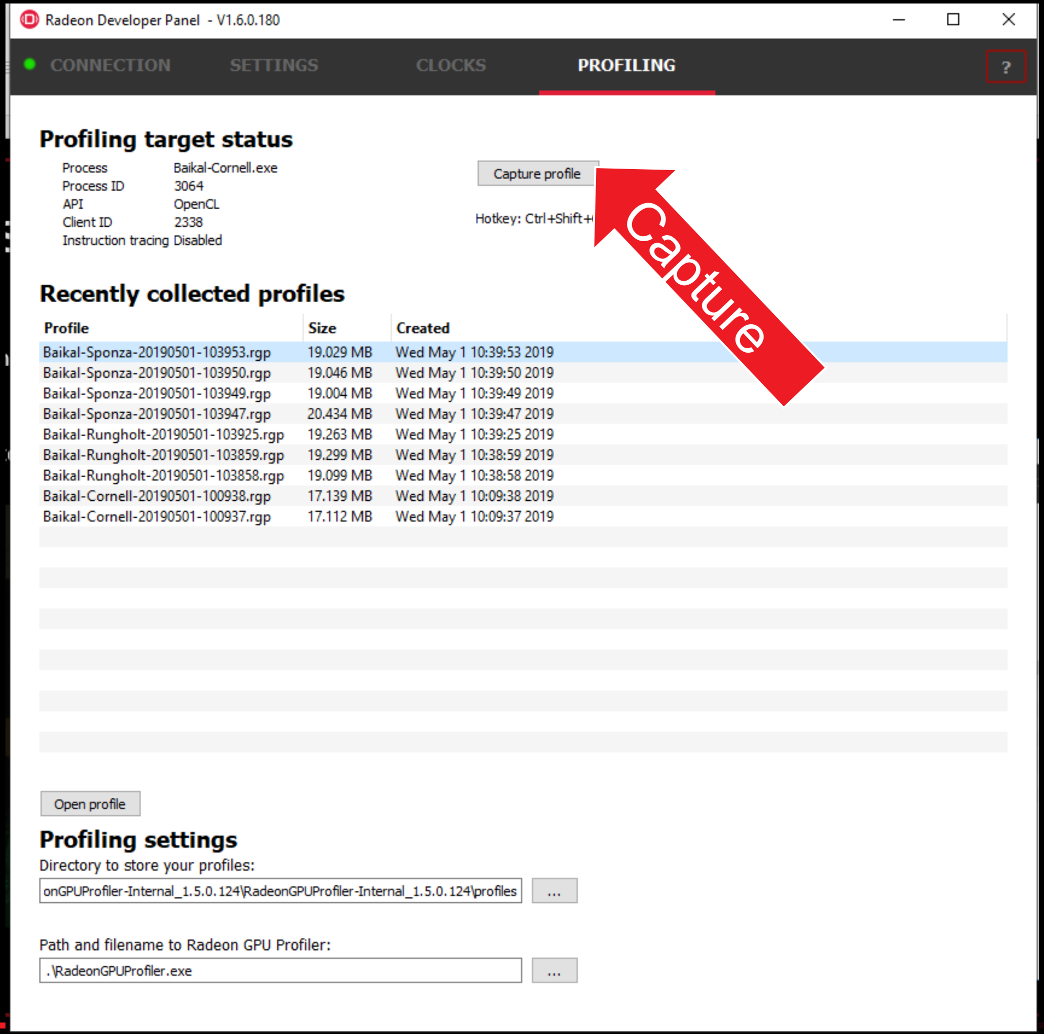
Launch the target application

Generate RGP Profile

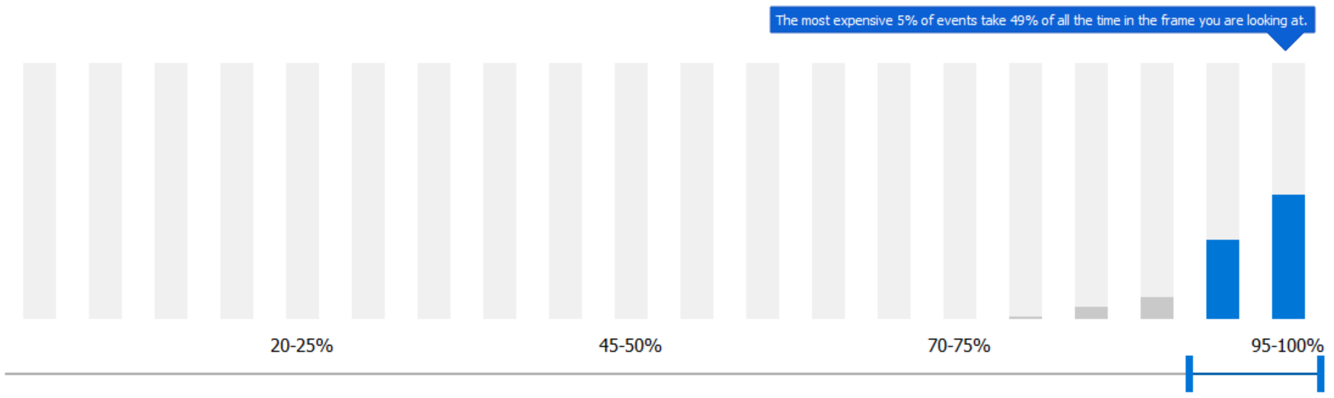
Capture a trace
Double click to open in RGP

(RGP support is built directly into the production driver)





Workflow with RGP - Most expensive events



The region you selected is approximately 80% of the profile's GPU time.

Queue	Event ID	Event	Duration
Queue index 1	32	clEnqueueNDRangeKernel("intersect_main")	4,166.205 μs
Queue index 1	24	clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	3,198.228 μs
Queue index 1	26	clEnqueueNDRangeKernel("occluded_main")	3,076.989 μs
Queue index 1	52	clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	2,946.078 μs
Queue index 1	60	clEnqueueNDRangeKernel("intersect_main")	2,869.898 μs
Queue index 1	4	clEnqueueNDRangeKernel("intersect_main")	2,402.357 μs
Queue index 1	80	clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	2,280.933 μs
Queue index 1	88	clEnqueueNDRangeKernel("intersect_main")	2,100.567 μs
Queue index 1	54	clEnqueueNDRangeKernel("occluded_main")	2,051.601 μs
Queue index 1	82	clEnqueueNDRangeKernel("occluded_main")	1,516.010 μs

Pinpoint optimization candidates

Workflow with RGP - Understand the Wavefront Occupancy

See how busy the GPU is

AGENDA

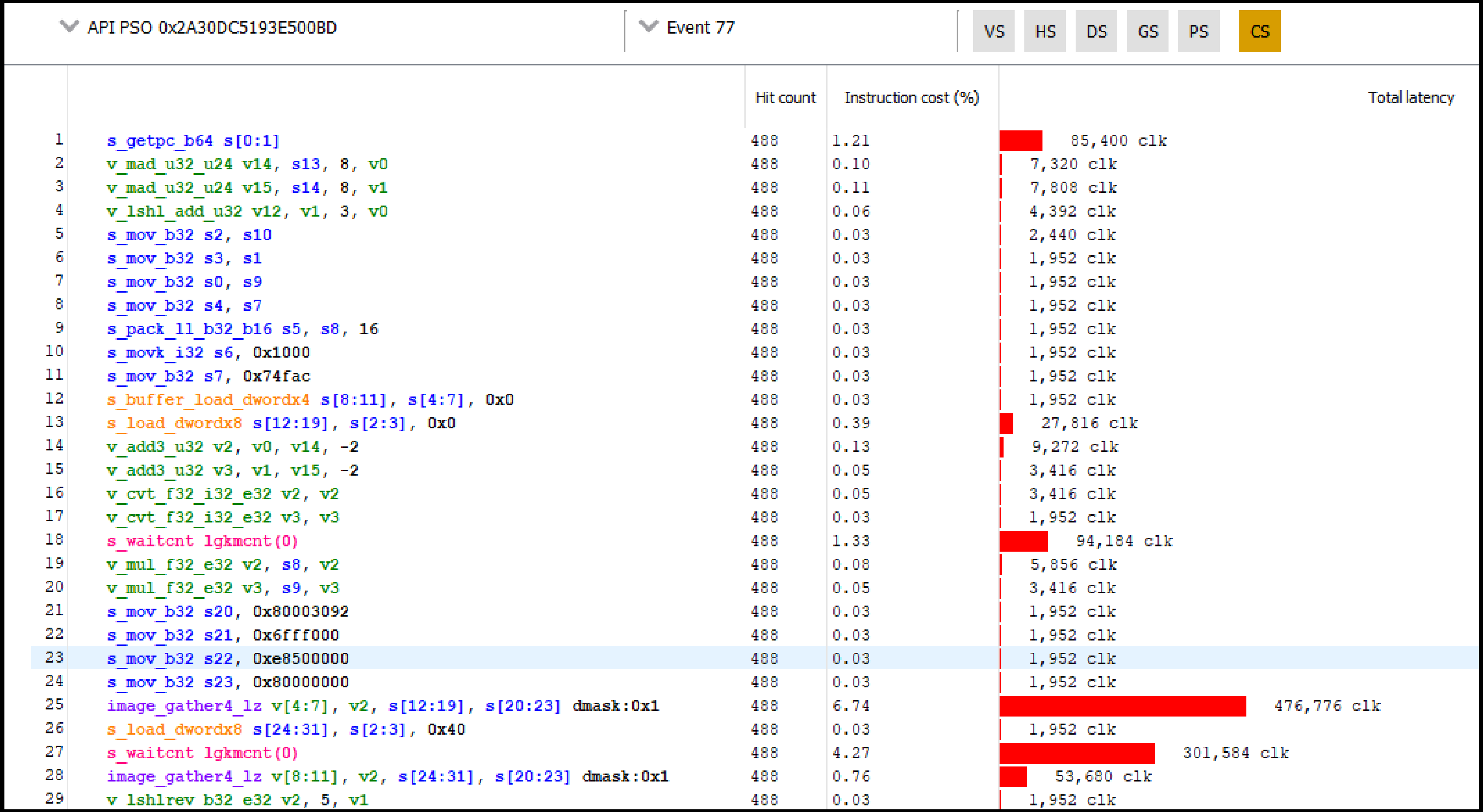
Existing profiling techniques and motivation for improvements to OpenCL tools

GCN and defining Wavefront Occupancy

Applying Wavefront occupancy in Radeon GPU Profiler

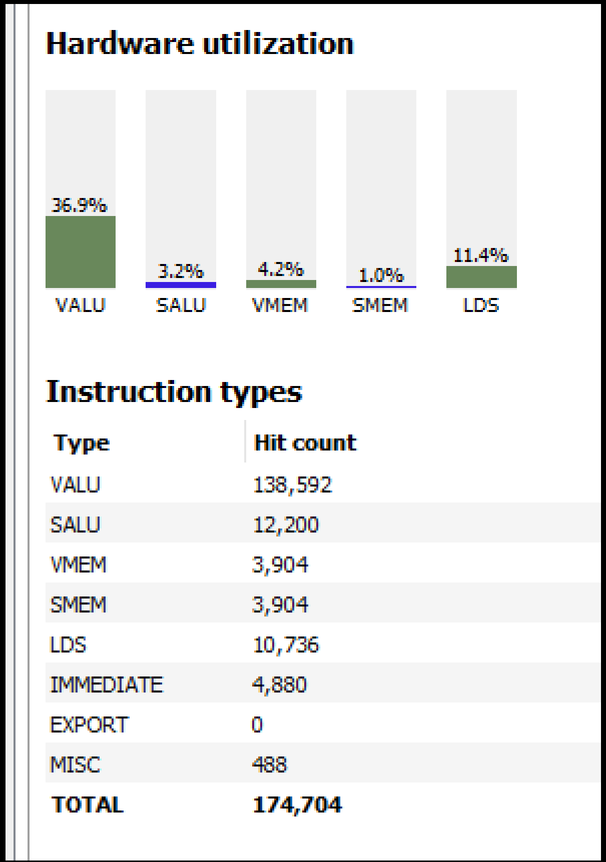
Future Work

Future work - RGP and Instruction Tracing



API PSO 0x2A30DC5193E500BD | Event 77 | VS HS DS GS PS CS

Instruction	Hit count	Instruction cost (%)	Total latency
s_getpc_b64 s[0:1]	488	1.21	85,400 clk
v_mad_u32_u24 v14, s13, 8, v0	488	0.10	7,320 clk
v_mad_u32_u24 v15, s14, 8, v1	488	0.11	7,808 clk
v_lshl_add_u32 v12, v1, 3, v0	488	0.06	4,392 clk
s_mov_b32 s2, s10	488	0.03	2,440 clk
s_mov_b32 s3, s1	488	0.03	1,952 clk
s_mov_b32 s0, s9	488	0.03	1,952 clk
s_mov_b32 s4, s7	488	0.03	1,952 clk
s_pack_ll_b32_b16 s5, s8, 16	488	0.03	1,952 clk
s_movk_i32 s6, 0x1000	488	0.03	1,952 clk
s_mov_b32 s7, 0x74fac	488	0.03	1,952 clk
s_buffer_load_dwordx4 s[8:11], s[4:7], 0x0	488	0.03	1,952 clk
s_load_dwordx8 s[12:19], s[2:3], 0x0	488	0.39	27,816 clk
v_add3_u32 v2, v0, v14, -2	488	0.13	9,272 clk
v_add3_u32 v3, v1, v15, -2	488	0.05	3,416 clk
v_cvt_f32_i32_e32 v2, v2	488	0.05	3,416 clk
v_cvt_f32_i32_e32 v3, v3	488	0.03	1,952 clk
s_waitcnt lgkmcnt(0)	488	1.33	94,184 clk
v_mul_f32_e32 v2, s8, v2	488	0.08	5,856 clk
v_mul_f32_e32 v3, s9, v3	488	0.05	3,416 clk
s_mov_b32 s20, 0x80003092	488	0.03	1,952 clk
s_mov_b32 s21, 0x6fff000	488	0.03	1,952 clk
s_mov_b32 s22, 0xe8500000	488	0.03	1,952 clk
s_mov_b32 s23, 0x80000000	488	0.03	1,952 clk
image_gather4_lz v[4:7], v2, s[12:19], s[20:23] dmask:0x1	488	6.74	476,776 clk
s_load_dwordx8 s[24:31], s[2:3], 0x40	488	0.03	1,952 clk
s_waitcnt lgkmcnt(0)	488	4.27	301,584 clk
image_gather4_lz v[8:11], v2, s[24:31], s[20:23] dmask:0x1	488	0.76	53,680 clk
v_lshlrev_b32_e32 v2, 5, v1	488	0.03	1,952 clk



Hardware utilization

Instruction types

Type	Hit count
VALU	138,592
SALU	12,200
VMEM	3,904
SMEM	3,904
LDS	10,736
IMMEDIATE	4,880
EXPORT	0
MISC	488
TOTAL	**174,704**

- Top-down program execution
- Find which part of your program is hot
- Available for all shader stages
- See instruction durations
- Functional unit utilization (VALU, SALU, LDS)
- Does not require kernel modification

Conclusion

- Wavefront Occupancy allows us to quantify performance at any point in time of a shader as it executes on a device

- HW support and driver instrumentation allows Radeon GPU Profiler to view wavefront occupancy and answer questions such as:
  - How OpenCL, DirectX 12 & Vulkan work on the GPU
  - Maps APIs directly to GPU concepts and activity
  - Uses custom GPU hardware for accurate low-level event and timing data

- Someone once said something like: “A picture is worth a thousand DWORDS”
  - RGP visualizes profile data using a simple UI

Thank you!

Questions?

Information

GPUOpen: https://gpuopen.com/

RGP: https://gpuopen.com/gaming-product/radeon-gpu-profiler-rgp/

RGA: https://gpuopen.com/gaming-product/radeon-gpu-analyzer-rga/

Downloads

RGP: https://github.com/GPUOpen-Tools/RGP/releases

RGA: https://github.com/GPUOpen-Tools/RGA/releases

Acknowledgements

The AMD Developer Tools Team

Gregory Mitrano for RGP content

Contact

Perhaad.Mistry@amd.com

Disclaimer & Attribution

ATTRIBUTION