

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

Perfor	mance Counters																		
Show Zero Columns																			
	Method	ExecutionOrder	ThreadID	CallIndex	Global	WorkSi	ze	Work	Group	Site	Time	LocalMemSize	VGPRs	SGPRs	KernelOccupancy	ScratchRegs	Wavefronts	VALUInsts	SALUIns ^
1	GenerateTileDomain k1 Baffin1	1	21084	1501	{ 1280	768	1}	{ 16	16	1}	0.06128		8	22	<u>100</u>	0	15360	19	10
2	PerspectiveCamera GeneratePaths k2 Baffin1	2	21084	1561	{ 983040	1	1}	{ 64	1 1	}	0.66784		23	27	<u>100</u>	0	15360	247	43
3	InitPathData k3 Baffin1	3	21084	1723	{ 983040	1	1}	{ 64	1 1	}	0.50656		7	19	100	0	15360	22	4
4	intersect main k4 Baffin1	4	21084	1794	{ 983040	1	1}	{ 64	1 1	}	1.18992	1096	54	31	50	0	15360	836.15	271.56
5	FilterPathStream k5 Baffin1	5	21084	1830	{ 983040	1	1}	{ 64	1 1	}	0.79840		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	56	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	<u>100</u>	0	1	182	68

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

Application Timeline Trace					
	3325.977 3326.22	3326.418 5 3326.473 3	326.721 3326.969	3327.216 3327.464	3327.712
⊡Host					
⊟ Host Thread 11476					
OpenCL					dFinish
OpenCL					
□ Context 0 (0x00000233419CC	.710)				
🗆 Queue 0 - gfx900 (0x00000	233418FB680)				
Data Transfer					
Kernel Execution	Shad	eSurfaceLlberV2	occluded_main erl	LightSar inters	ect_main
Other Enqueue Operations	S			1	

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

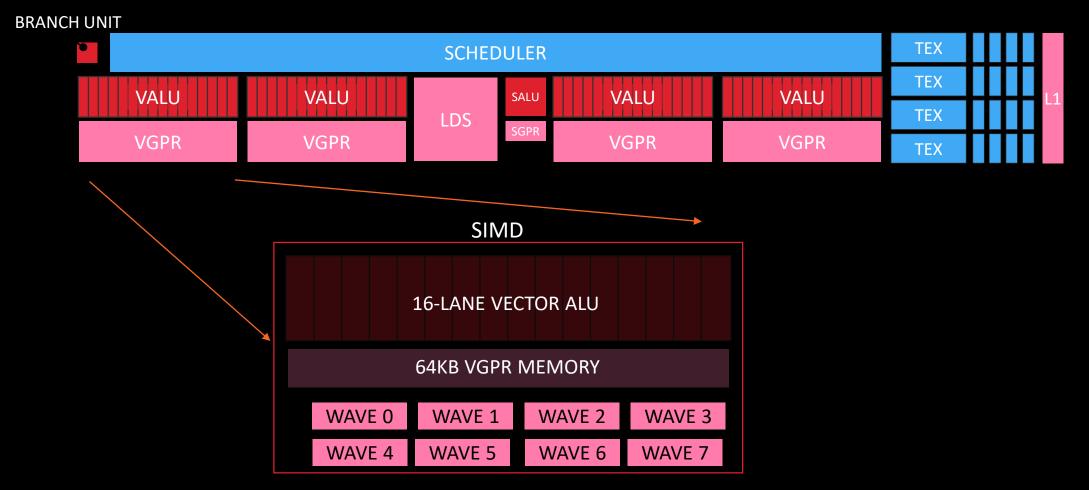
X Shader Engines per Chip with Y Compute Units per Shader Engine For OpenCL developers



IWOCL 2019

GCN Compute Unit

4 SIMD Units per CU



What is wavefront occupancy?

8 Wave Slots Per SIMD on RX480

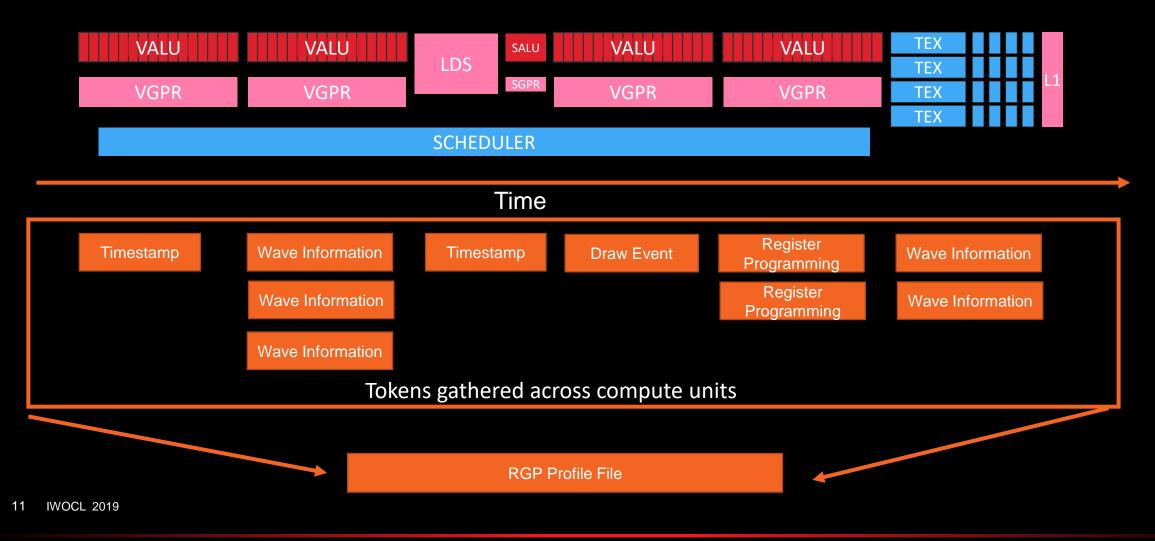


Measure of how close a SIMD is to its maximum wavefront capacity at a point in time

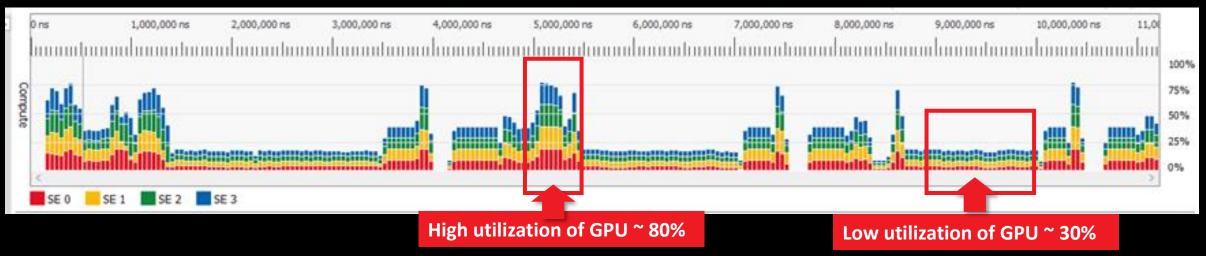
IWOCL 2019

How do we calculate Wavefront Occupancy

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

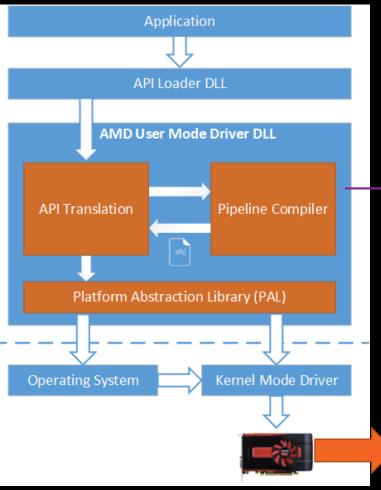


Wavefront Occupancy

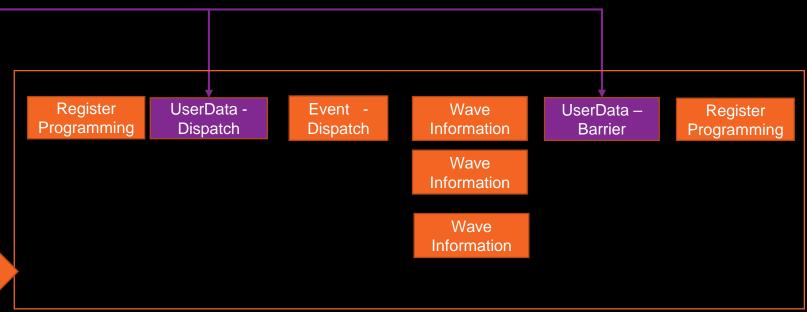


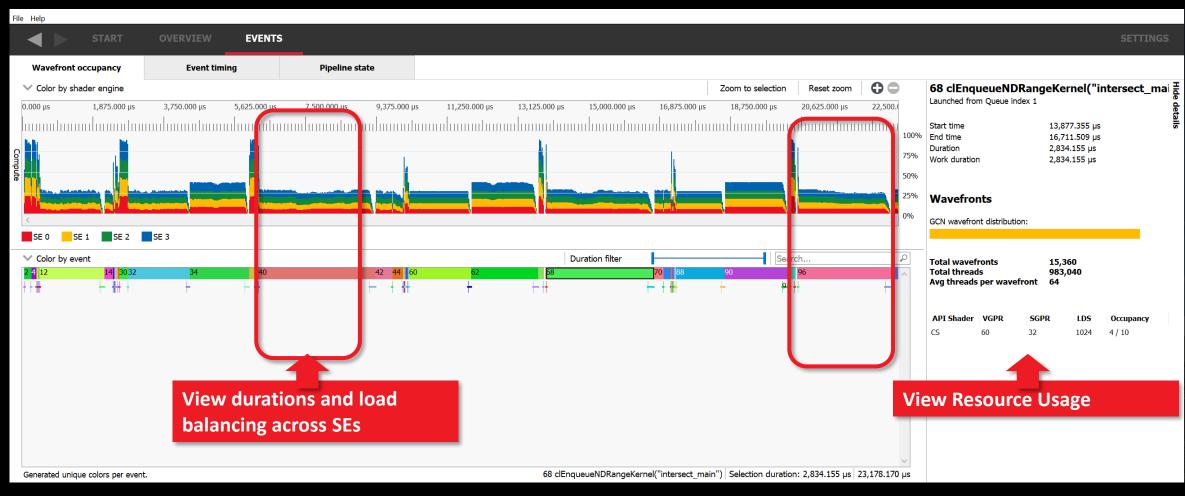
- Four shader engines doing OpenCL ray tracing
- Build a histogram showing how many waves were active in a time interval
 - Now get an idea of utilization of a GPU at any point in time
- However How do we correlate this to a API call in OpenCL?

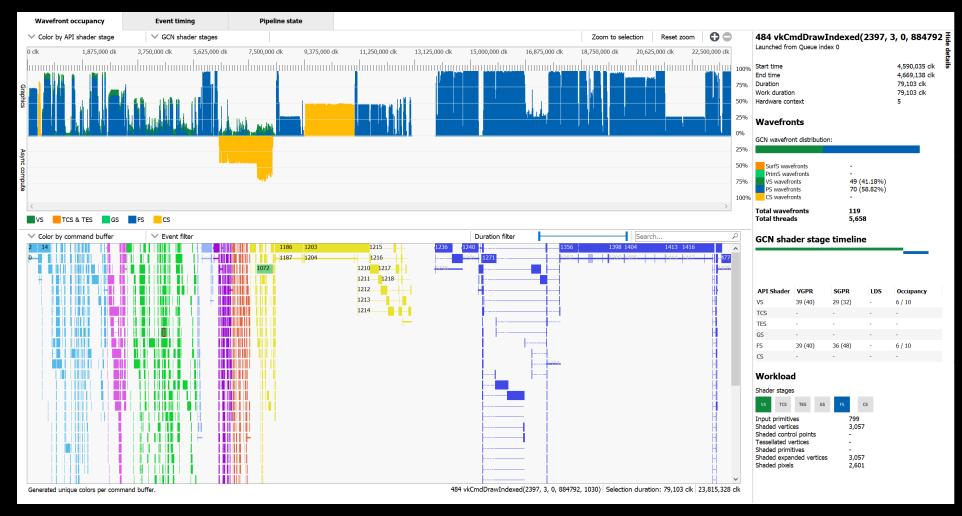
Correlating wave fronts to API information

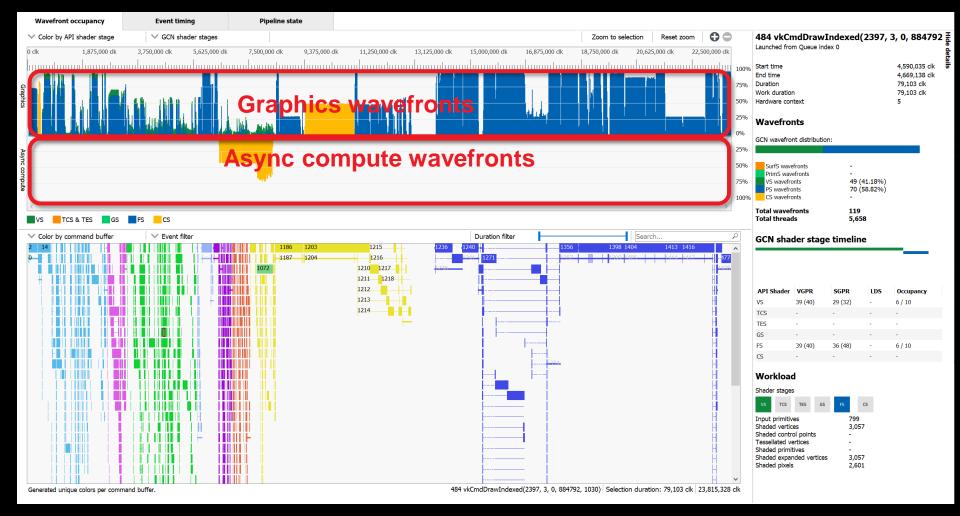


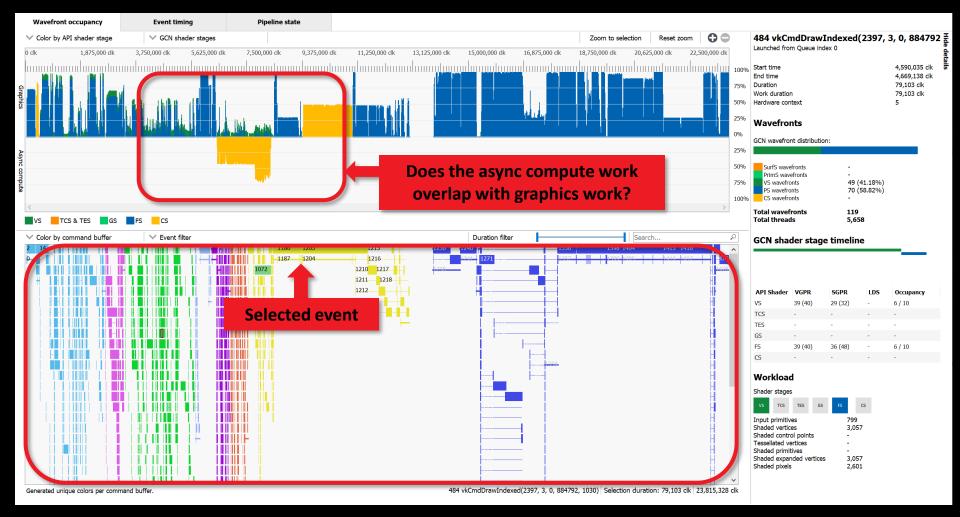
- User mode driver inserts data into hardware generated trace
- Allows adding API specific semantics
 - Example OpenCL kernel names













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

					Cac	hes	Barrier	type and reason
	Event Number	Duration s	Drain Time	Stalk	Invalidated	Flushed	Barrier type	Reason for barrier
Shown as barriers	1	1,124 ns	753 ns	CS	K L1		DRIVER	Profiling control.
in RGP	3	17,949 ns	745 ns	C	K L1		DRIVER	Profiling control.
	5	12,680 ns	726 ns	C	K L1		DRIVER	Profiling control.
	7	10,324 ns	780 ns	C	K L1		DRIVER	Profiling control.
	9	10,334 ns	801 ns	CS	K L1		DRIVER	Profiling control.
	11	1,266 ns	739 ns	C	K L1		DRIVER	Profiling control.

Understanding Application and Driver Interaction in OpenCL

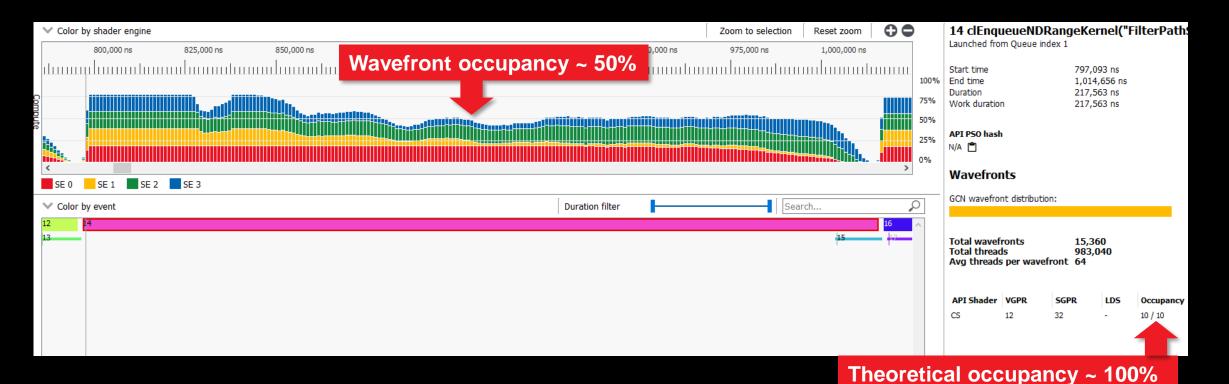
- Enabled by driver instrumentation and hardware support
- A command buffer is a batch of dispatches
- View grouping of OpenCL dispatches into command buffers
 - Example: Find the cost of clFlush()
 - View how long DMA takes and overlap

ubmis	ssion	0 ns 2,000,000 ns	3,000,000 ns	4,000,000 ns	5,000,000 ns	6,000,000 ns	7,000,000 ns	8,000,000 ns	9,000,000 ns	10,000,000 ns	Ī
Compu	te queue							Command			1
[8	[843] CommandBuffer [842] CommandBuffer [841] CommandBuffer 40] CommandBuffer			[843] Comn [842] Comn [842] Comn [841] Comn	nandBuffer			[843] CommandBuffer		_	1
10			Queue			utes on GPU		[0 12] Commandadurer			
20 _{IW}	OCI 2019			Tir	ne						

Understanding Application and Driver Interaction in OpenCL

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

IWOCL 2019



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

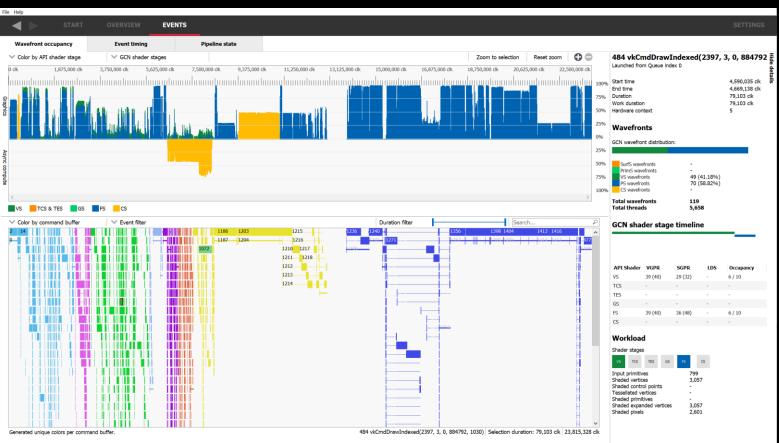
Radeon GPU Profiler

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

Radeon Developer P	anel - V1.6.0.180			- 0	
CONNECTION	SETTINGS	CLOCKS	PROFILING		
Connection	status				
\Box —	/	-5			
	to the Radeon Developer Se ww.connection.log	ervice running at localho	st		
Setup targe	t application				
Enable profiling 🔵	Edit Instrug	configuration		×	1
Apply settings	Executable	: All detected applica	ations		
	API:	✓ OpenCL			
	Profile Mod	e: V Dispatch range			
		Start 10			
			ons will be profiled from Dispatch 10 to Dis	spatch 19 when launched.	
Active appli	Cations	to enable applying setti	ngs and profile collection	Restore Defaults	
Executable name	API	to enable applying sett	ngs and prome conection.		
Baikal-Cornell.exe	OpenCL				
CXLTeaPot.exe	OpenCL				

Baikal-Sponza.exe

Baikal-Rungholt.exe OpenCl

OpenCL

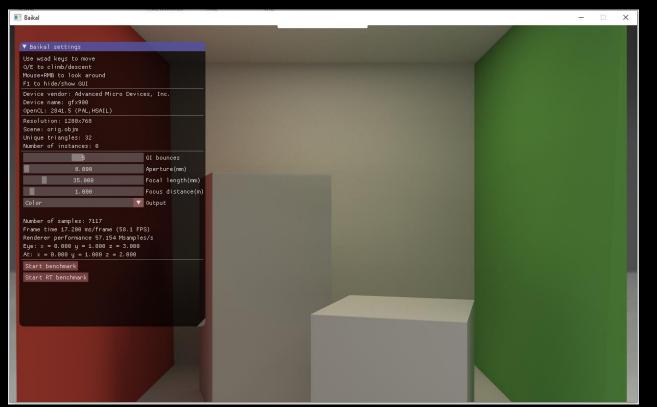
How does it work?

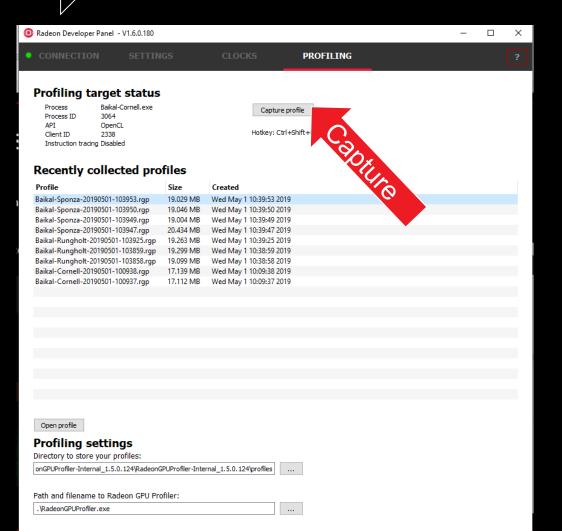
Launch the target application

Generate RGP Profile

Capture a trace Double click to open in RGP

(RGP support is built directly into the production driver)



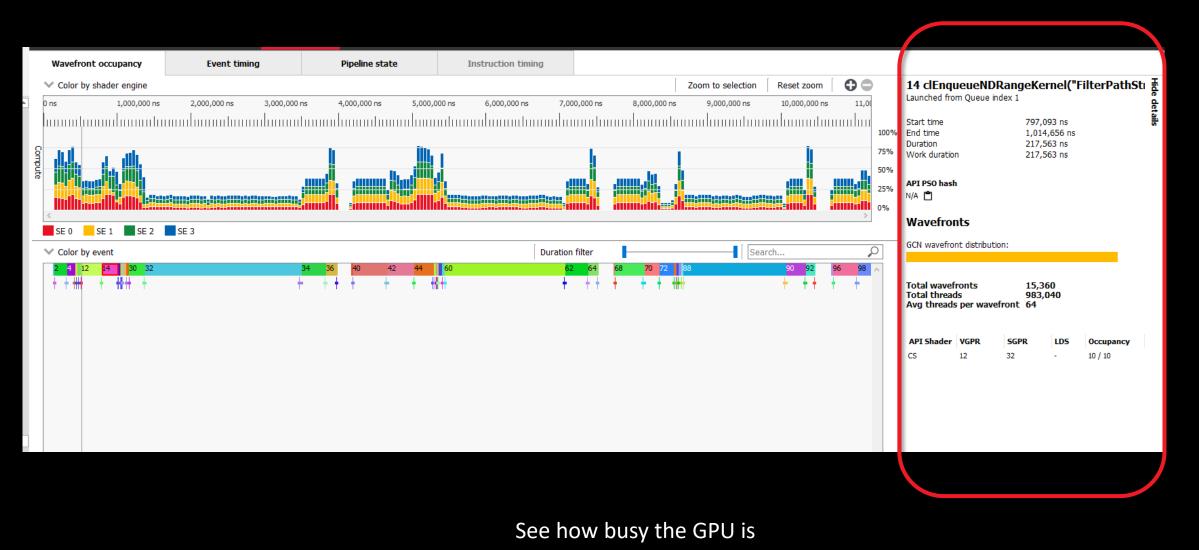


Workflow with RGP - Most expensive events

				The most expensive 5% of events take 49% of	all the time in the frame you are looking at.
		20-25%	45-50%	70-75%	95-100%
					— —
The region	you selec	ted is approximately 80% of the pro	ofile's GPU time.		
The region	you selec	ted is approximately 80% of the pro	ofile's GPU time.		
Queue Queue index 1	Event ID	Event	Duration		
Queue Queue index 1 Queue index 1	Event ID	Event clEnqueueNDRangeKernel("intersect_main")	Duration 4,166.205 μs		
Queue Queue index 1 Queue index 1 Queue index 1	Event ID 32 24	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	Duration 4,166.205 μs 3,198.228 μs		
Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1	Event ID 32 24 26	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("occluded_main")	Duration 4,166.205 μs 3,198.228 μs 3,076.989 μs		
Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1	Event ID 32 24 26 52	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("cocluded_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	Duration 4,166.205 μs 3,198.228 μs 3,076.989 μs 2,946.078 μs		
Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1	Event ID 32 24 26 52 60	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("occluded_main") clEnqueueNDRangeKernel("intersect_main")	Duration 4,166.205 μs 3,198.228 μs 3,076.989 μs 2,946.078 μs 2,869.898 μs		
Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1 Queue index 1	Event ID 32 24 26 52 60 4	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("occluded_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main")	Duration 4,166.205 μs 3,198.228 μs 3,076.989 μs 2,946.078 μs 2,869.898 μs 2,402.357 μs		
Queue index 1 Queue index 1	Event ID 32 24 26 52 60 4 80	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("occluded_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2")	Duration 4,166.205 µs 3,198.228 µs 3,076.989 µs 2,946.078 µs 2,869.898 µs 2,402.357 µs 2,280.933 µs		
Queue index 1 Queue index 1	Event ID 32 24 26 52 60 4 80 88	Event clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("ShadeSurfaceUberV2") clEnqueueNDRangeKernel("occluded_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main") clEnqueueNDRangeKernel("intersect_main")	Duration 4,166.205 µs 3,198.228 µs 3,076.989 µs 2,946.078 µs 2,869.898 µs 2,402.357 µs 2,280.933 µs 2,100.567 µs		

Pinpoint optimization candidates

Workflow with RGP - Understand the Wavefront Occupancy



27 IWOCL 2019

AGENDA

Existing profiling techniques and motivation for improvements to OpenCL tools

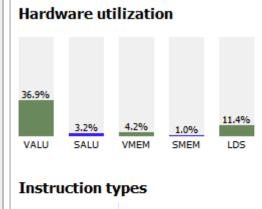
GCN and defining Wavefront Occupancy

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
		Hit count	Instruction cost (%)		Total latency
1	s getpc b64 s[0:1]	488	1.21	85,400 clk	
2	v mad u32 u24 v14, s13, 8, v0	488	0.10	7,320 clk	
3	v mad u32 u24 v15, s14, 8, v1	488	0.11	7,808 clk	
4	v lshl add u32 v12, v1, 3, v0	488	0.06	4,392 clk	
5	s mov b32 s2, s10	488	0.03	2,440 clk	
6	s mov b32 s3, s1	488	0.03	1,952 clk	
7	s mov b32 s0, s9	488	0.03	1,952 clk	
8	s mov b32 s4, s7	488	0.03	1,952 clk	
9	s pack 11 b32 b16 s5, s8, 16	488	0.03	1,952 clk	
10	s movk i32 s6, 0x1000	488	0.03	1,952 clk	
11	s mov b32 s7, 0x74fac	488	0.03	1,952 clk	
12	<pre>s buffer load dwordx4 s[8:11], s[4:7], 0x0</pre>	488	0.03	1,952 clk	
13	s load dwordx8 s[12:19], s[2:3], 0x0	488	0.39	27,816 clk	
14	v add3 u32 v2, v0, v14, -2	488	0.13	9,272 clk	
15	v add3 u32 v3, v1, v15, -2	488	0.05	3,416 clk	
16	v cvt f32 i32 e32 v2, v2	488	0.05	3,416 clk	
17	v cvt f32 i32 e32 v3, v3	488	0.03	1,952 clk	
18	s waitcnt lgkmcnt(0)	488	1.33	94,184 clk	
19	v mul f32 e32 v2, s8, v2	488	0.08	5,856 clk	
20	v mul f32 e32 v3, s9, v3	488	0.05	3,416 clk	
21	s mov b32 s20, 0x80003092	488	0.03	1,952 clk	
22	s mov b32 s21, 0x6fff000	488	0.03	1,952 clk	
23	s mov b32 s22, 0xe8500000	488	0.03	1,952 clk	
24	s mov b32 s23, 0x80000000	488	0.03	1,952 clk	
25	image gather4 lz v[4:7], v2, s[12:19], s[20:23] dmask:0x	1 488	6.74		476,776 clk
26	s load dwordx8 s[24:31], s[2:3], 0x40	488	0.03	1,952 clk	
27	s waitcnt lgkmcnt(0)	488	4.27	301,584	clk
28	image gather4 lz v[8:11], v2, s[24:31], s[20:23] dmask:0	x1 488	0.76	53,680 clk	
29	v lshlrev b32 e32 v2, 5, v1	488	0.03	1,952 clk	



Туре	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: <u>https://gpuopen.com/</u>

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

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

Downloads

RGP: <u>https://github.com/GPUOpen-Tools/RGP/releases</u> RGA: <u>https://github.com/GPUOpen-Tools/RGA/releases</u>

Acknowledgements

The AMD Developer Tools Team Gregory Mitrano for RGP content

Contact

Perhaad.Mistry@amd.com

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2019 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.