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

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
What is wavefront occupancy?

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

Tokens gathered across compute units

RGP Profile File
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

View Resource Usage

View durations and load balancing across SEs
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

Does the async compute work overlap with graphics work?

Selected event
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

<table>
<thead>
<tr>
<th>Event Numbers</th>
<th>Duration</th>
<th>Drain Time</th>
<th>Stall</th>
<th>Invalidated</th>
<th>Flushed</th>
<th>Barrier Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,124 ns</td>
<td>753 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
<tr>
<td>2</td>
<td>17,949 ns</td>
<td>745 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
<tr>
<td>3</td>
<td>12,680 ns</td>
<td>726 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
<tr>
<td>7</td>
<td>10,324 ns</td>
<td>780 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
<tr>
<td>9</td>
<td>10,354 ns</td>
<td>801 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
<tr>
<td>11</td>
<td>1,206 ns</td>
<td>739 ns</td>
<td>CS</td>
<td>K</td>
<td>L1</td>
<td>DRIVER</td>
</tr>
</tbody>
</table>
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
Understanding Application and Driver Interaction in OpenCL

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

![Image of OpenCL occupancy graphs]

- Wavefront occupancy ~ 50%
- Theoretical occupancy ~ 100%
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

Active applications

<table>
<thead>
<tr>
<th>Executable name</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babylon-Load</td>
<td>OpenCL</td>
</tr>
<tr>
<td>CudaTest.axd</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Babylon-Space</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Babylon-Rumble</td>
<td>OpenCL</td>
</tr>
</tbody>
</table>

IWOCL 2019
How does it work?

1. Launch the target application
2. Generate RGP Profile
3. 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.

<table>
<thead>
<tr>
<th>Queue</th>
<th>Event ID</th>
<th>Event</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue index 1</td>
<td>32</td>
<td>cElr: voueln(RangeKernel\text{intersect_main})</td>
<td>4,106,205 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>24</td>
<td>cElr: voueln(RangeKernel\text{Shade_Surface_vertex2})</td>
<td>3,198,228 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>26</td>
<td>cElr: voueln(RangeKernel\text{occluded_main})</td>
<td>3,176,699 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>52</td>
<td>cElr: voueln(RangeKernel\text{Shade_Surface_vertex2})</td>
<td>2,946,070 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>60</td>
<td>cElr: voueln(RangeKernel\text{intersect_main})</td>
<td>2,669,090 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>4</td>
<td>cElr: voueln(RangeKernel\text{intersect_main})</td>
<td>2,402,357 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>80</td>
<td>cElr: voueln(RangeKernel\text{Shade_Surface_vertex2})</td>
<td>2,380,033 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>88</td>
<td>cElr: voueln(RangeKernel\text{intersect_main})</td>
<td>2,106,567 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>54</td>
<td>cElr: voueln(RangeKernel\text{occluded_main})</td>
<td>2,151,001 µs</td>
</tr>
<tr>
<td>Queue index 1</td>
<td>82</td>
<td>cElr: voueln(RangeKernel\text{occluded_main})</td>
<td>1,516,010 µs</td>
</tr>
</tbody>
</table>

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

- 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

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 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.