Debugging and Analyzing Programs using the Intercept Layer for OpenCL™ Applications

Ben Ashbaugh

IWOCL 2018

https://github.com/intel/opencl-intercept-layer
Why am I here?

Intercept Layer for OpenCL Applications
- Debug and Analyze OpenCL Applications
- Open Source, Permissive License
- Works with Any* OpenCL Implementation
- Requires No Application Modifications
- Thin, Fast, Easy to Install / Uninstall
- Community Contributions are Welcome and Encouraged!

* Covers all major OpenCL implementations.
Agenda

History
How it Works
What it Can Do
Implementation Details
Possible Next Steps
Wrap Up
History

(2009-Present)
Initial Requests:

I’m debugging an application. Can you modify the driver to print the OpenCL APIs that are called?

Yeah, no problem.

One week later...

Can you print the API arguments too?

Sure, I think we can add that.
I’m debugging the GPU compiler. Can you modify the driver to dump OpenCL kernels to a file?

Yeah, that’s not too hard.

Great.

Can you also make it work for the CPU OpenCL implementation?

I’m not sure – I think so?

Fantastic. Can you make it work for [third party competitor]?
Meanwhile:

Our Driver Team was also adding instrumentation:

- Flush or Finish After Enqueue
- Assert on OpenCL Errors
- Timing API Calls
- More ...

But:

- Required driver modifications!

Is there a better way to add these capabilities?
Prior Work from Graphics APIs:

Can we build something similar for OpenCL?
How It Works
Intercept Layer for OpenCL Applications

Architecture: How it Works*:
- Inserts between Application and OpenCL ICD Loader
- Constructs Dispatch Table During Initialization
- Passes Through API Calls... or not!

Philosophies:
- Focus on Features that Solve Problems
  - For OpenCL Implementers
  - For OpenCL Developers
- Support Any OpenCL Device on Any Platform
- Be Invisible By Default

* Typical usage on Windows and Linux, OSX is a little different.
What It Can Do - Examples
Call and Error Logging

```plaintext
>>> clGetPlatformIDs
<<<< clGetPlatformIDs
>>>> clGetPlatformIDs
<<<< clGetPlatformIDs
>>>> clGetDeviceIDs: platform = [ NVIDIA CUDA ], device_type = CL_DEVICE_TYPE_ALL (FFFFFFFF)
<<<< clGetDeviceIDs
>>>> clGetDeviceIDs: platform = [ Intel(R) OpenCL ], device_type = CL_DEVICE_TYPE_ALL (FFFFFFFF)
<<<< clGetDeviceIDs
>>>> clCreateContextFromType: properties = [ CL_CONTEXT_PLATFORM = Intel(R) OpenCL ], ...
ERROR! clCreateContextFromType returned CL_DEVICE_NOT_FOUND (-1)
<<<< clCreateContextFromType: returned 00000000
>>>> clCreateContextFromType: properties = [ CL_CONTEXT_PLATFORM = Intel(R) OpenCL ], ...
<<<< clCreateContextFromType: returned 00E97068
>>>> clGetContextInfo: param_name = CL_CONTEXT_DEVICES (00001081)
<<<< clGetContextInfo
>>>> clGetContextInfo: param_name = CL_CONTEXT_DEVICES (00001081)
<<<< clGetContextInfo
>>>> clCreateCommandQueue: device = [ Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz (CL_DEVICE_TYPE_CPU) ]
<<<< clCreateCommandQueue: returned 05B038F8
>>>> clGetContextInfo: param_name = CL_CONTEXT_DEVICES (00001081)
<<<< clGetContextInfo
>>>> clGetContextInfo: param_name = CL_CONTEXT_DEVICES (00001081)
<<<< clGetContextInfo
>>>> clCreateProgramWithSource: context = 00E97068, count = 1
<<<< clCreateProgramWithSource: returned 04572EA8, program number = 0000
```
Dumping Program Source (and Binaries!)

Can also Modify and/or Inject Modified Program Source or Binaries!
## Host API Performance Timing

### Host Performance Timing Results:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Average (ns)</th>
<th>Min (ns)</th>
<th>Max (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clBuildProgram</td>
<td>3</td>
<td>711065926</td>
<td>22172160</td>
<td>1634864192</td>
</tr>
<tr>
<td>clCreateBuffer</td>
<td>23</td>
<td>2234125</td>
<td>2113</td>
<td>36218573</td>
</tr>
<tr>
<td>clCreateCommandQueue</td>
<td>1</td>
<td>25054</td>
<td>25054</td>
<td>25054</td>
</tr>
<tr>
<td>clCreateContext</td>
<td>1</td>
<td>123618277</td>
<td>123618277</td>
<td>123618277</td>
</tr>
<tr>
<td>clCreateImage2D</td>
<td>2</td>
<td>8600269</td>
<td>4682137</td>
<td>12518402</td>
</tr>
<tr>
<td>clCreateKernel</td>
<td>6</td>
<td>7898</td>
<td>2113</td>
<td>14489</td>
</tr>
<tr>
<td>clCreateProgramWithSource</td>
<td>3</td>
<td>24551</td>
<td>4829</td>
<td>51617</td>
</tr>
<tr>
<td>clEnqueueNDRangeKernel</td>
<td>1</td>
<td>7529273</td>
<td>7529273</td>
<td>7529273</td>
</tr>
<tr>
<td>clEnqueueNDRangeKernel</td>
<td>1</td>
<td>1095145</td>
<td>1095145</td>
<td>1095145</td>
</tr>
<tr>
<td>clEnqueueNDRangeKernel</td>
<td>1</td>
<td>25952</td>
<td>15998</td>
<td>24253177</td>
</tr>
<tr>
<td>clEnqueueNDRangeKernel</td>
<td>1</td>
<td>29856</td>
<td>15696</td>
<td>218847</td>
</tr>
<tr>
<td>clEnqueueReadBuffer</td>
<td>2288</td>
<td>3758695</td>
<td>123158</td>
<td>10236648</td>
</tr>
<tr>
<td>clFinish</td>
<td>2</td>
<td>4723341</td>
<td>717519</td>
<td>8729163</td>
</tr>
<tr>
<td>clFlush</td>
<td>18036</td>
<td>31018</td>
<td>21432</td>
<td>374003</td>
</tr>
<tr>
<td>clGetDeviceIDs</td>
<td>4</td>
<td>1811</td>
<td>301</td>
<td>5735</td>
</tr>
</tbody>
</table>

...
Device Performance Timing Results:

Total Time (ns): 123904875200

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Time (ns)</th>
<th>Time (%)</th>
<th>Average (ns)</th>
<th>Min (ns)</th>
<th>Max (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdvancePaths</td>
<td>18036</td>
<td>28203368032</td>
<td>22.76%</td>
<td>1563726</td>
<td>1388096</td>
<td>1761472</td>
</tr>
<tr>
<td>Init</td>
<td>1</td>
<td>8600000</td>
<td>0.01%</td>
<td>8600000</td>
<td>8600000</td>
<td>8600000</td>
</tr>
<tr>
<td>InitFrameBuffer</td>
<td>1</td>
<td>155712</td>
<td>0.00%</td>
<td>155712</td>
<td>155712</td>
<td>155712</td>
</tr>
<tr>
<td>Intersect</td>
<td>18036</td>
<td>79765237056</td>
<td>64.38%</td>
<td>4422556</td>
<td>3248832</td>
<td>5297600</td>
</tr>
<tr>
<td>Sampler</td>
<td>18036</td>
<td>14307721664</td>
<td>11.55%</td>
<td>793286</td>
<td>75712</td>
<td>1182400</td>
</tr>
<tr>
<td>clEnqueueReadBuffer</td>
<td>2288</td>
<td>1619792736</td>
<td>1.31%</td>
<td>707951</td>
<td>39904</td>
<td>4220992</td>
</tr>
</tbody>
</table>
Performance Timing on VTune and Chrome*

This app created one In-Order GPU queue.

This app made API calls from one thread.

These are commands in the GPU queue:

You can click on a command to get more detail, in some cases.
Explore how applications respond to different query responses!
Implementation Details: OpenCL API Learnings and Insights
OpenCL APIs from a Layering Perspective

Most things went really well!

Features that made life easy:
- Built-in Reference Counting and Object Queries
- Standard Event Profiling, Standard Program Binaries
- Online Compilation

Features that made things complicated:
- NULL Local Work Size: Need “what happened” queries!
- Out-of-Order Queues – especially with Device Timing
- Device-side Only Controls (kernel attributes)
  - Easier to permute Host-side Controls (build options)
Intel-Specific Enhancements
# Intel Specific Enhancement: Driver Diagnostics

**cl_intel_driver_diagnostics**: Intel Extension

- Extends Context Callback to Include **GOOD / BAD / INFORMATIONAL** Diagnostics
- Use the Intercept Layer for OpenCL Applications to Enable and Log Diagnostics

```plaintext
>>>>
clCreateBuffer: flags = CL_MEM_WRITE_ONLY | CL_MEM_ALLOC_HOST_PTR (12), ...
======>
Context Callback (private_info = 00AFF728, cb = 4):
Performance hint: clCreateBuffer needs to allocate memory for buffer. For subsequent
operations the buffer will share the same physical memory with CPU.

<====== End of Context Callback

<<<<
clCreateBuffer: returned 0573E000
...
```
Future Work
Future Work – Short Term

Continue to stay use-case driven: How to find bugs and fix them faster?

- Example: OpenCL Object Leak Detection

Improve Usability and Accessibility
Future Work – Longer Term

Android Support 😊

Intercept Layer as an ICD: Work with applications that statically link to the ICD loader.
Future Work – Longer Term

Log and Analyze Graphs of OpenCL Commands

- Especially Important with Out-of-Order Queues
- Can we plot graphs of commands?
- Can we time device execution of subgraphs?
Future Work – Longer Term

Automatic Reproducer Generation
- Very Limited Capture-Playback
- One Kernel + Inputs + Params
- See Fossilize for Vulkan

Speaking of Vulkan...
- Lots of Layer Prior Art
- Steal with Pride?
Wrap Up
Summary and Call to Action

Try the Intercept Layer for OpenCL Applications!
- Debug and Analyze OpenCL programs faster!
- Send Issues and Pull Requests!

Grow the OpenCL Ecosystem with Layers
- Layers are an important part of the OpenCL ecosystem

To the Khronos OpenCL Working Group: Design the API with layers in mind!
To OpenCL Users: Use layers, evangelize layers, build layers!

Thank you!
- ben.ashbaugh@intel.com
Acknowledgements

Thanks to Michal Mrozek, Michael Carroll, Mike Kinsner, and Adam Herr for reviewing these slides.

Thanks to everyone from Intel who has used or contributed to the Intercept Layer for OpenCL Applications!
Useful Links:

Intercept Layer for OpenCL Applications:
https://github.com/intel/opencl-intercept-layer

Vulkan Loader and Layers:
https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers
Legal Notice and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2018 Intel Corporation.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

* Other names and brands may be claimed as the property of others.
Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804