Emulating Command Buffer Extensions with OpenCL Layers

James Brodman, Intel Corporation

Ben Ashbaugh (Intel), Ewan Crawford (Codeplay)
OpenCL Command Buffers were provisionally released November 2021!

... but implementation support remains low

(Data from opencl.gpuinfo.org, March 2024)
Problem Statement

- Some OpenCL extensions take a long time to implement
- Some OpenCL devices may never support an OpenCL extension
- Lack of implementations hinders adoption:
  - Applications won’t support an extension without implementations
  - Other implementors won’t support an extension without applications
- We need a way to break this cycle!
  - Improve developer confidence that a feature will be available
  - Provide a competent fallback when an implementation is unavailable

We implemented support for command buffers in an OpenCL layer, demonstrating one way to break the cycle.
Prior Work
OpenCL Intercept Layer

- The OpenCL Intercept Layer can emulate some OpenCL extensions
- How does this work?
  - Augment existing APIs, e.g. `clGetDeviceInfo`
  - Implement new APIs by hooking `clGetExtensionFunctionAddress`
- Functional, but a heavyweight solution

https://github.com/intel/opencl-intercept-layer/blob/main/docs/controls.md#controls
Installable OpenCL Layers

- Installable OpenCL Layers can also intercept and augment OpenCL functions
- Lighter weight, easy to enable and disable individual layers
- Most prior work for tracing and validation
  - No (known) prior work to emulate extensions
- We decided to try this mechanism – and it worked!

How the Emulation Layer Works
Three Classes of Layer Functions

1. **Emulation Functions**: new functionality, implemented entirely within the layer

```c
cl_int CL_API_CALL clCommandBarrierWithWaitListKHR_EMU(
    cl_command_buffer_khr cmdbuf,
    cl_command_queue command_queue,
    cl_uint num_sync_points_in_wait_list,
    const cl_sync_point_khr* sync_point_wait_list,
    cl_sync_point_khr* sync_point,
    cl_mutable_command_khr* mutable_handle)
{
    if (!CommandBuffer::isValid(cmdbuf)) {
        return CL_INVALID_COMMAND_BUFFER_KHR;
    }
    if (cl_int errorCode = cmdbuf->checkRecordErrors(
            command_queue,
            num_sync_points_in_wait_list,
            sync_point_wait_list,
            mutable_handle)) {
        return errorCode;
    }
    cmdbuf->addCommand(
        BarrierWithWaitList::create(cmdbuf, command_queue),
        num_sync_points_in_wait_list,
        sync_point_wait_list,
        sync_point,
        mutable_handle);
    return CL_SUCCESS;
}
```
Three Classes of Layer Functions

1. **Emulation Functions**: new functionality, implemented entirely within the layer

2. **Override Functions**: add functionality in some cases, otherwise pass along

```c
static cl_int CL_API_CALL clGetDeviceInfo_layer(
    cl_device_id device,
    cl_device_info param_name,
    size_t param_value_size,
    void* param_value,
    size_t* param_value_size_ret)
{
    cl_int errorCode = CL_SUCCESS;
    if (clGetDeviceInfo_override(
            device,
            param_name,
            param_value_size,
            param_value,
            param_value_size_ret,
            &errorCode) == false)
    {
        return g_pNextDispatch->clGetDeviceInfo(
            device,
            param_name,
            param_value_size,
            param_value,
            param_value_size_ret);
    }
    return errorCode;
}
```
Three Classes of Layer Functions

1. **Emulation Functions**: new functionality, implemented entirely within the layer

2. **Override Functions**: add functionality in some cases, otherwise pass along

3. **Bookkeeping Functions**: record some info, then unconditionally pass along

```c
static cl_int CL_API_CALL clReleaseEvent_layer(cl_event event)
{
    cl_uint refCount = 0;
    g_pNextDispatch->clGetEventInfo(event,
        CL_EVENT_REFERENCE_COUNT,
        sizeof(refCount),
        &refCount, nullptr);
    if (refCount == 1) {
        auto& context = getLayerContext();
        auto it = context.EventMap.find(event);
        if (it != context.EventMap.end()) {
            g_pNextDispatch->clReleaseEvent(it->second);
            context.EventMap.erase(it);
        }
    }
    return g_pNextDispatch->clReleaseEvent(event);
}
```
Command Buffer Construction “Records” Commands

- Record each command in the command buffer
  - Plus, any arguments
  - Plus, some bookkeeping info
- Notes:
  - Need to retain OpenCL objects!
  - Need to clone OpenCL kernels to preserve kernel args!

```cpp
struct CopyBuffer : Command
{
    static std::unique_ptr<CopyBuffer> create(
        cl_command_buffer_khr cmdbuf, cl_command_queue queue,
        cl_mem src_buffer, cl_mem dst_buffer,
        size_t src_offset, size_t dst_offset,
        size_t size)
    {
        auto ret = std::unique_ptr<CopyBuffer>(
            new CopyBuffer(cmdbuf, queue));

        ret->src_buffer = src_buffer;
        ret->dst_buffer = dst_buffer;
        ret->src_offset = src_offset;
        ret->dst_offset = dst_offset;
        ret->size = size;

        g_pNextDispatch->clRetainMemObject(ret->src_buffer);
        g_pNextDispatch->clRetainMemObject(ret->dst_buffer);

        return ret;
    }

private:
    CopyBuffer(
        cl_command_buffer_khr cmdbuf,
        cl_command_queue queue) : Command(cmdbuf, queue, CL_COMMAND_COPY_BUFFER) {};
};
```
Command Buffer Enqueue “Plays Back” Commands

- Enqueues each recorded command into the provided command queue

- Notes:
  - Need to map sync points to events
  - May need to insert command queue barriers in some cases (not shown)

```cpp
struct CopyBuffer : Command
{
  // <snip>
  int playback(
      cl_command_queue queue,
      std::vector<cl_event>& deps) const override
  {
    auto wait_list = getEventWaitList(deps);
    auto signal = getEventSignalPtr(deps);
    return g_pNextDispatch->clEnqueueCopyBuffer(
      queue,
      src_buffer,
      dst_buffer,
      src_offset,
      dst_offset,
      size,
      static_cast<cl_uint>(wait_list.size()),
      wait_list.data(),
      signal);
  }
  // <snip>
};
```
Brief Retrospective
Most things went well!

- OpenCL installable layer mechanism is solid!
- Many OpenCL features make layering easy:
  - Built-in Reference Counting and Object Queries
  - `clCloneKernel` to Clone Kernels and their Arguments
Some things were tricky...

- How can we do event profiling for command buffers?
  - Need to profile a group of commands
- Solution: add barriers with event profiling

![Command Buffer Diagram]

Use this event for:
- CL_PROFILING_COMMAND_QUEUED,
- CL_PROFILING_COMMAND_SUBMIT,
- CL_PROFILING_COMMAND_START

Use this event for:
- CL_PROFILING_COMMAND_END
Verdict: Success!

(Data collected with the OpenCL Intercept Layer, IWOCL 2018)
Some things were tricky…

- How can we do error checking when commands are recorded?

```plaintext
clCommandNDRangeKernelKHR does not return CL_INVALID_WORK_GROUP_SIZE when invalid work size are passed #95
```

mfrancepillois opened this issue on Dec 20, 2023 - 5 comments

mfrancepillois commented on Dec 20, 2023

While testing the Command Buffer Emulation layer, I noticed that `clCommandNDRangeKernelKHR` does not return `CL_INVALID_WORK_GROUP_SIZE` when invalid work size is passed whereas `clEnqueueNDRangeKernel` returns it.

When using the Command Buffer Emulation layer this error code is actually returned when calling `clEnqueueCommandBufferKHR`.

Test case

I set up a simple test based on 04Julia sample code to show this problem:
**Tentative Solution:**

**Setup:**
1. Create a “Test Queue” when command buffer is created
2. Also, create a “Blocking Event” when command buffer is created
3. Enqueue a Barrier dependent on the “Blocking Event”

**Recording:**
4. Enqueue commands to “Test Queue” before recording
   - Command does not execute due to barrier dependency
   - But error checking is performed!

**Finalization:**
5. Set “Blocking Event” to error state when command buffer is finalized
   - All dependent command discarded!
Verdict: Partial Success?

- Relies on tricky behavior / dusty corners of the spec
- Still in a branch, probably will not be enabled by default
Some things were tricky…

- How can we track command buffer states?
  - RECORDING is straightforward…
  - EXECUTABLE is straightforward, too…
  - PENDING is complicated!

- No current solution

- Possibility:
  - Track event for the last enqueue, test if it is COMPLETE?
  - Might work, but adds complexity and overhead
Some things were tricky...

- The **PENDING** state is the only layer CTS failure!
- Nice to fix, but probably doesn’t affect much code in practice...

```bash
$ ./test_conformance/extensions/cl_khr_command_buffer/test_cl_khr_command_buffer info_state
Initializing random seed to 0.
Requesting Default device based on command line for platform index 3 and device index 0
Compute Device Name = Intel(R) UHD Graphics 770, Compute Device Vendor = Intel(R) Corporation, Compute Device Version = OpenCL 3.0 NEO , CL C Version = OpenCL C 1.2
Device latest conformance version passed: v2023-05-16-00
Supports single precision denormals: YES
sizeof( void*) = 8 (host)
sizeof( void*) = 8 (device)
```

```
info_state
ERROR: Unexpected result of CL_COMMAND_BUFFER_STATE_KHR query!! (!(state == expected) from /home/bashbaug/git/OpenCL-CTS/test_conformance/extensions/cl_khr_command_buffer/command_buffer_get_command_buffer_info.cpp:222)
ERROR: verify_state failed! ((unknown) from /home/bashbaug/git/OpenCL-CTS/test_conformance/extensions/cl_khr_command_buffer/command_buffer_get_command_buffer_info.cpp:260)
ERROR: RunStateInfoTest failed! ((unknown) from /home/bashbaug/git/OpenCL-CTS/test_conformance/extensions/cl_khr_command_buffer/command_buffer_get_command_buffer_info.cpp:69)
ERROR: Test Failed! ((unknown) from /home/bashbaug/git/OpenCL-CTS/test_conformance/extensions/cl_khr_command_buffer/basic_command_buffer.h:105)
info_state FAILED
PASSED sub-test.
FAILED test.
```
Current Usage Examples
Conformance Test Suite Development

Test CL_COMMAND_BUFFER_CONTEXT_KHR #1697

Develop and debug the CTS on any device!

Bonus: CTS found a few bugs in the layer, too…
Layered Extension Development

- `cl_khr_command_buffer` is a base specification, designed to support additional functionality via layered extensions
  - Examples:
    - `cl_khr_command_buffer_multi_device`
    - `cl_khr_command_buffer_mutable_dispatch`
    - `cl_khr_command_buffer_mutable_memory_commands`

Emulation layer provides a convenient mechanism to quickly prototype layered extensions!
High-Level Language Feature Development

- SYCL Graph is an experimental oneAPI extension to build and execute entire graphs of commands:

- For OpenCL backends, graphs are recorded into command buffers.

Emulation layer provides a convenient mechanism to develop, debug, and test the SYCL Graph extension!

(Diagram from “Towards Deferred Execution of a SYCL Command Graph”, IWOCL 2023)
A Brief Look at Performance
Key Performance Questions

- Is the layer expensive?
  - How does layer performance compare to non-command buffer performance?

- Test Parameters:
  - Submission time or completion time?
  - How many kernels?

- Is the layer effective?
  - How does layer performance compare to native command buffer performance?

- Test Parameters:
  - Submission time or completion time?
  - How many kernels?
  - In-order or out-of-order?

Developed microbenchmarks to answer these questions!

(“How to Optimize Compute Drivers? Let’s Start with Writing Good Benchmarks!”, IWOCL 2022)
Microbenchmark #1: ExecuteCommandBuffer

- Enqueue N kernels directly?

- Or enqueue N kernels in a Command Buffer?

- Measure submission time or completion time

ExecuteCommandBuffer Results

Command Buffer Execution Time With Layer
(Normalized to Non-Command Buffer Time, Lower Is Better)

Generally, no layer penalty!

Submission Time: ExecuteCommandBuffer(api=ocl UseCommandBuffers=1 NumKernels=10 KernelExecutionTime=1 MeasureCompletionTime=0)
Completion Time: ExecuteCommandBuffer(api=ocl UseCommandBuffers=1 NumKernels=10 KernelExecutionTime=1 MeasureCompletionTime=1)
ExecuteCommandBuffer Results

Command Buffer Execution Time Without Layer
(Normalized to Layer Time, Lower Is Better)

Bug reported!
Native implementation faster:

Layer performance is acceptable.

Submission Time: `ExecuteCommandBuffer(api=ocl UseCommandBuffers=1 NumKernels=10 KernelExecutionTime=1 MeasureCompletionTime=0)`
Completion Time: `ExecuteCommandBuffer(api=ocl UseCommandBuffers=1 NumKernels=10 KernelExecutionTime=1 MeasureCompletionTime=1)`
Summary and Conclusion
Summary and Conclusion

- Successfully emulated command buffers with an OpenCL layer!
  - Almost all features are implemented, layer is *almost* conformant

- Command buffer emulation layer is useful!
  - Accelerates layered extension design and development
  - Accelerates CTS development
  - Accelerates SYCL Graph development
  - Handy alternative for debugging and performance analysis

- OpenCL layer mechanism is robust, performant, and capable
  - Consider emulation for future extensions to improve adoption?

- Thank you!
Disclaimers

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Related Links and References

- Command Buffer Emulation Layer
  - [https://github.com/bashbaug/SimpleOpenCLSamples/tree/main/layers/10_cmdbufemu](https://github.com/bashbaug/SimpleOpenCLSamples/tree/main/layers/10_cmdbufemu)

- Command Buffer Microbenchmarks

- Referenced IWOCL Presentations
  - Layers for OpenCL (IWOCL 2021) ([slides](#))
  - Debugging and Analyzing Programs Using the Intercept Layer for OpenCL Applications (IWOCL 2018) ([slides](#))
  - Towards Deferred Execution of a SYCL Command Graph (IWOCL 2023) ([slides](#))
  - How to Optimize Compute Drivers? Let’s Start with Writing Good Benchmarks! (IWOCL 2022) ([slides](#))
## System Configuration

<table>
<thead>
<tr>
<th><strong>Host:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OS:</strong></td>
<td>Linux bashbaug-newpc 6.5.0-26-generic #26~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Mar 12 10:22:43 UTC 2 x86_64 x86_64 x86_64 GNU/Linux</td>
</tr>
<tr>
<td><strong>CPU:</strong></td>
<td>12th Gen Intel(R) Core(TM) i9-12900K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Drivers:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA GeForce RTX 3060</td>
<td>535.86.10</td>
</tr>
<tr>
<td>Intel(R) UHD Graphics 770</td>
<td>24.09.28717.12</td>
</tr>
<tr>
<td>POCL</td>
<td>PoCL 5.0 Linux, RelWithDebInfo, RELOC, SPIR, SPIR-V, LLVM14.0.0, SLEEF, POCL_DEBUG (built from tag v5.0, commit 0bffce0)</td>
</tr>
<tr>
<td>oneAPI Construction Kit</td>
<td>ComputeAorta 4.0.0 Linux x86_64 (RelWithDebInfo, 85dfbf7e) (built from commit 85dfbf7e, with LLVM19.0.0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Software:</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulation Layer</td>
<td>(built from commit 80222e5)</td>
</tr>
<tr>
<td>Compute-Benchmarks</td>
<td>(built from commit 17b58e0)</td>
</tr>
</tbody>
</table>