OpenCL ICD Loader
Status and Perspectives

Brice Videau
Argonne National Laboratory

Khronos F2F Meeting, Montréal, 2023-10-27

OpenCL ICD Loader Status

OpenCL ICD Loader Features

- Multiplexing of vendor drivers
  - Through vendor provided dispatch tables
- Provides a layering system
  - Global layers, enabled for each call
  - Introspection utility (`cllayerinfo`)
- Configuration through environment variables (or registry on windows)

OpenCL ICD Loader Issues

- Not a well behaved library
  - Leaks memory
  - Cannot be unloaded gracefully
    - Doesn’t unload vendor drivers
    - Doesn’t de-initialize and unload layers
- Dispatching issues
  - The loader doesn’t know the size of the vendor provided dispatch table, leading to segfaults when calling outside the dispatch table range

OpenCL ICD Loader Missing Features

- No layer can be added while inside the application
  - Instance layers (Vulkan)

Potential Solutions

Library Behavior

- Expected behavior
  - libOpenCL.xx should not leak memory
  - libOpenCL.xx should be able to be opened and closed
- Solution: library destructor (and constructor?)
  - Leverages host system synchronization to avoid race conditions
- Caveats:
  - Vendor libraries need to support being loaded/unloaded (optional in vulkan, env variable) could be an extension
  - Layers need an explicit de-initialization function (new version of API)

New Dispatch Strategy - 1

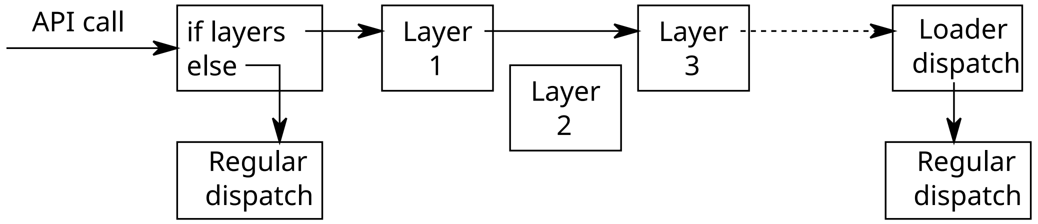
- Expected behavior
  - libOpenCL.xx should not rely on a vendor provided dispatch table
- Solution: loader managed dispatch (Vulkan)
  - API to query all entry points: clGetFunctionAddressForPlatformKHR (queried through clGetExtensionFunctionAddress)
  - Change OpenCL objects layouts (more on the next slide)
  - Maintain backward compatibility?

New Dispatch Strategy - 2

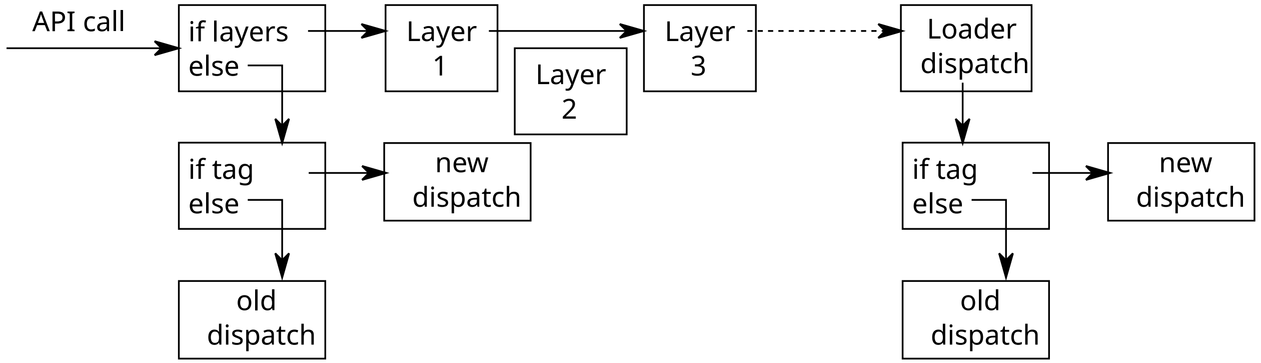
- OpenCL objects layouts
  - cl_khr_icd2
  - Currently: `struct _cl_platform_id { cl_icd_dispatch *dispatch; };`
  - Proposed: `struct _cl_platform_id { cl_icd_dispatch *dispatch; void *disp_data; };`
  - Use `clGetPlatformIDs` entry (first entry of the dispatch table, unused by the loader) to look a for a tag.
  - A well chosen tag should allow distinguishing objects from old OpenCL implementations
    - Low bits set, not a valid pointer to a function on most architectures
      - 64 bit: `0x4F50454E434C3331` (ASCII "OPENCL31")
      - 32 bit: `0x434C3331` (ASCII "CL31")
  - `disp_data` can be set by the loader (but could also be set by compliant implementation, more on that later)
  - `disp_data` would contain necessary dispatch information from the loader

New Dispatch Strategy - 3

- As of today ICD Loader dispatch:



- Backward compatible Strategy:



New Dispatch Strategy - 4

- Use `disp_data` to store dispatch information about object
  - Loader managed dispatch tables (akin to Vulkan)
  - Proof of concept here: https://github.com/Kerilk/OpenCL-ICD-Loader/commits/managed-dispatch
  - Populated through new clGetFunctionAddressForPlatformKHR
  - When objects are created the `disp_data` is set to that of their parent
    - Can be done by the loader, but not in extension functions like:

      extern CL_API_ENTRY cl_int CL_API_CALL clEnqueueSVMUnmapARM(cl_command_queue command_queue, void* svm_ptr, cl_uint num_events_in_wait_list, const cl_event* event_wait_list, **cl_event* event**);
    - Maybe better to let the vendor driver copy the `disp_data` pointer from the `command_queue` to the `event`

Instances and Layers

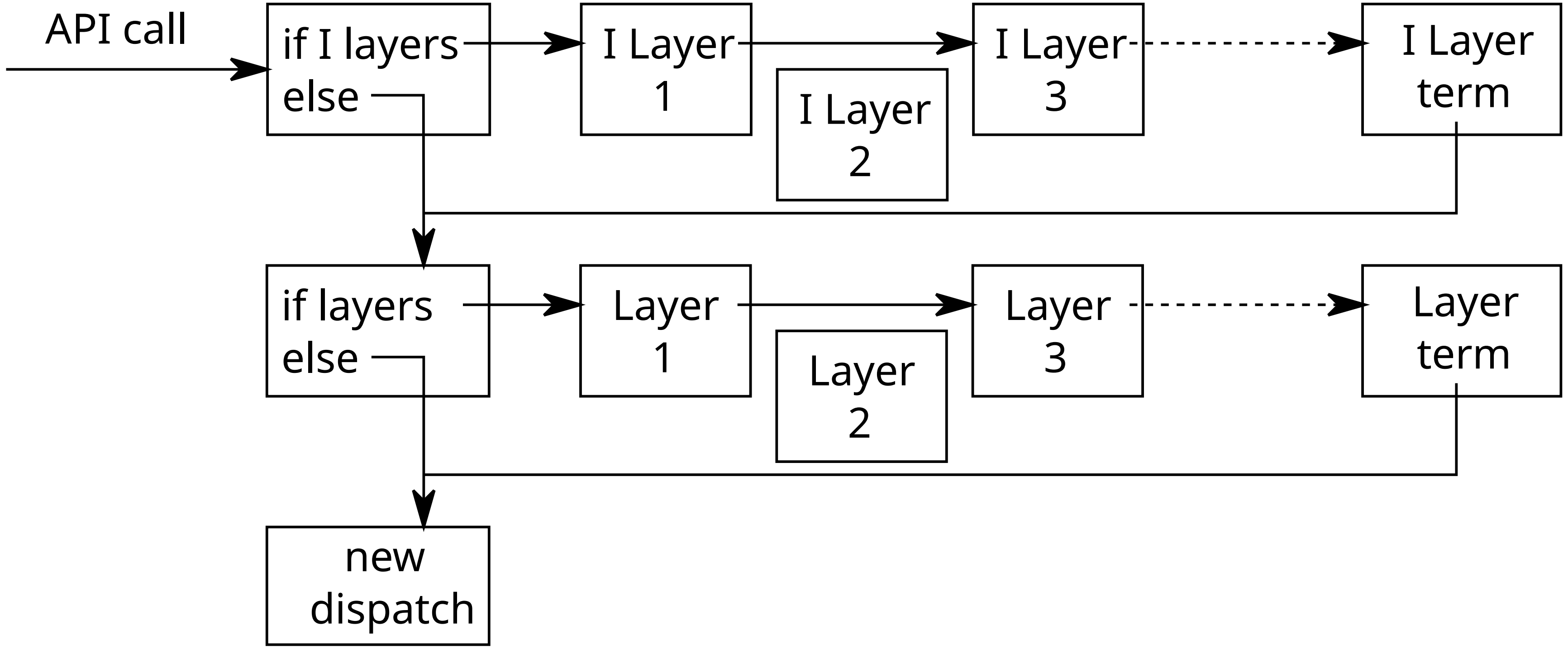
Can We Extend Further?

- Proposed solutions would fix current shortcomings
  - Library behavior
  - Dispatching issues
- Can we extend things further?
  - Application controlled
    - Instances
    - layers
  - Similar to what Vulkan proposes
  - Sketched by Ben Ashbaugh (Intel) in 2018

Yes We Can (or at least we should be able to...)

- Vulkan proved loader managed dispatch is enough for application controlled layers
- Application controlled layer without instances:
  - Would be limited to per platform, since dispatch data is inherited from platform to devices
- Idea, similar to Vulkan: create an instance to encapsulate an application controlled layer setup (and other state potentially, loader implemented)
  - clCreateInstance(...layers configuration...) / clDestroyInstance / clGetPlatformIDsForInstance
- Add concept of instance platforms and devices, instance specific handles for platform and devices (has to be implemented by vendors)
  - `clGetInstancePlatformsIDsKHR(cl_instance instance, cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)`
  - `clReleaseInstancePlatformIDsKHR()`
  - `clGetDeviceIDs` called with an instance platform would return instance devices.
  - Instance handles can be used as regular handles in the rest of the API (like sub-devices)

Instance Layers Dispatching



Conclusion

cl_khr_icd2

- As outlined cl_khr_icd2 would allow
  - Fixing library behavior
  - Solving dispatching issues
  - Enabling new features such as instance layers
  - Maintain backward compatibility
    - Instances would only be supported for ck_khr_icd2 implementations
    - Legacy drivers would not be unloaded
    - New implementations would continue working on old loaders

Impact on Vendors

- Extension specifying if vendor driver can be unloaded? (or part of cl_khr_icd2)
- cl_khr_icd2 would be
  - 3 new entry points
    - clGetFunctionAddressForPlatformKHR
    - clGetInstancePlatformsIDsKHR
    - clReleaseInstancePlatformIDsKHR
  - 2 new object types: instance platform and devices (but not new C types)
  - A pointer copy per object creation
  - Most probably a few info queries
- Could reduce this overhead if we abandon instances

Acknowledgment

This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.