14th International Workshop on OpenCL and SYCL

IWOCL 2026

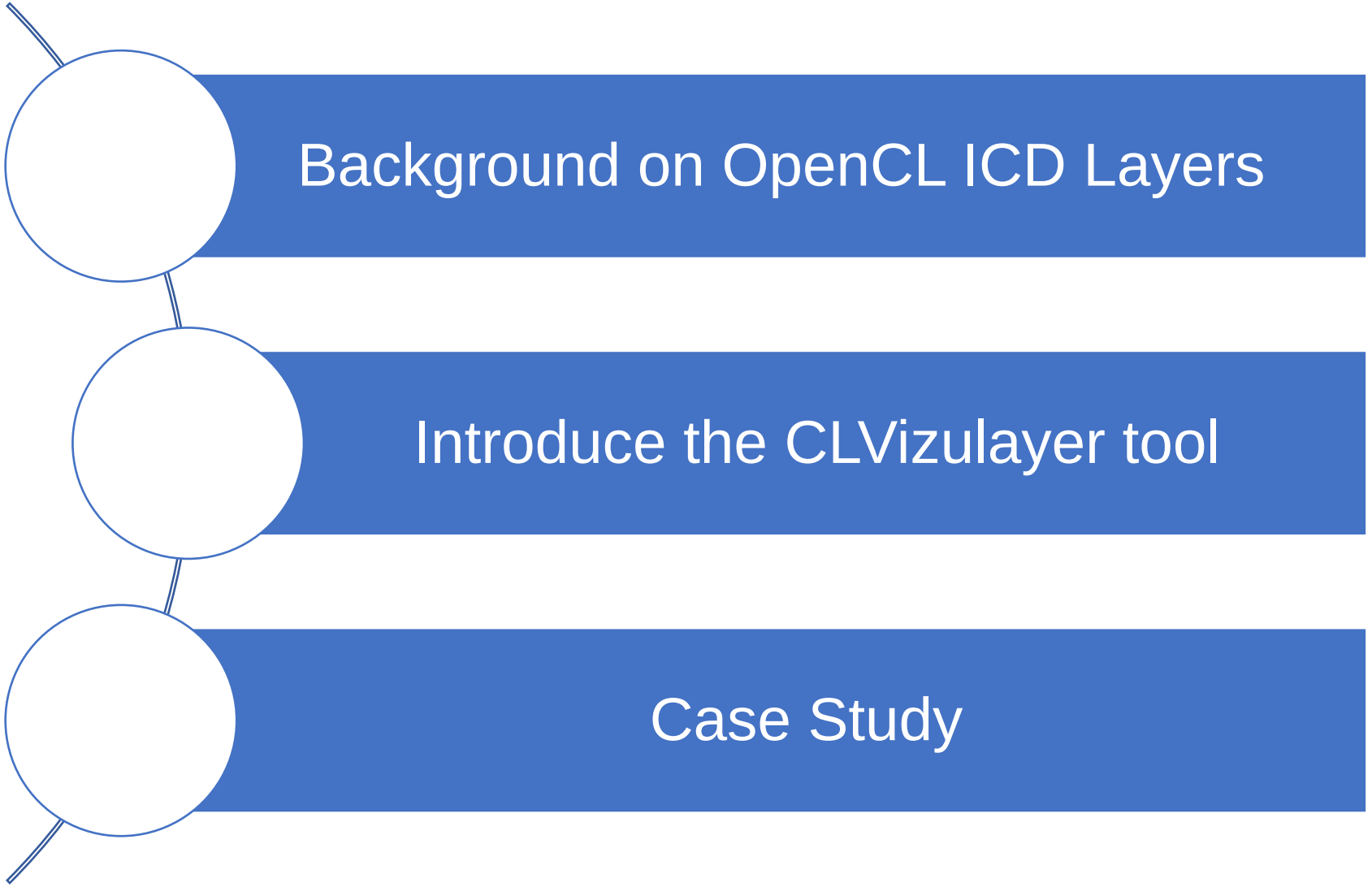
CLVizulayer: A Tool for Visualising the Directed Acyclic Graph of OpenCL Device Submissions

Ewan Crawford, Stream HPC

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

Agenda

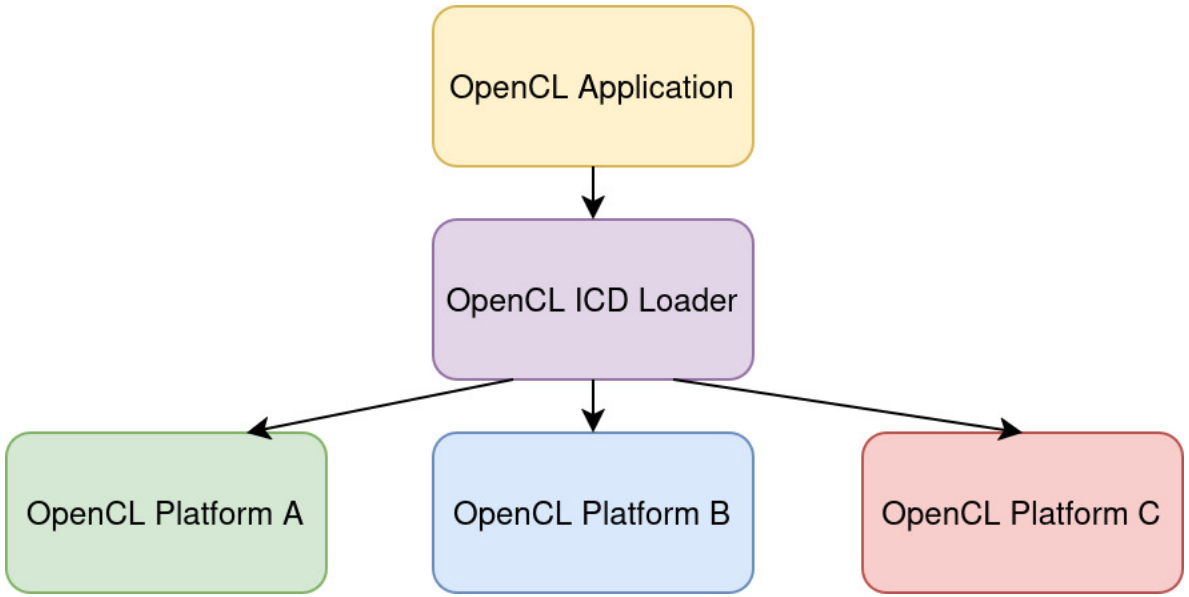
STREAM
High Performance Computing



Background

What is the OpenCL ICD Loader and how does it use layers?

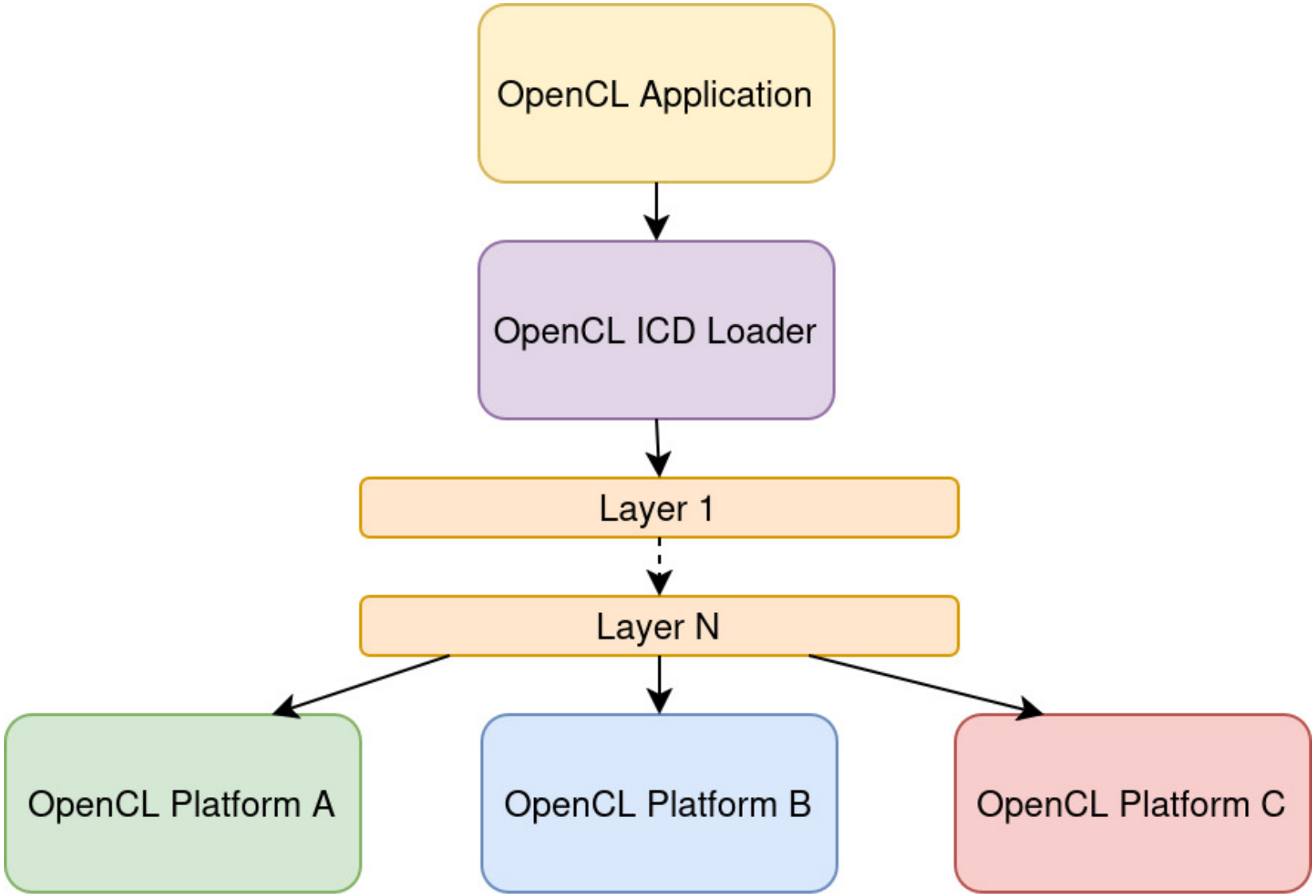
OpenCL ICD Loader



ICD stands for *Installable Client Driver*

- OpenCL applications are typically linked against an ICD Loader rather than OpenCL implementation directly.
- Benefits:
  - Ship applications that will run against OpenCL implementations the developer doesn't have access to.
  - Multiple OpenCL implementations can coexist on the same system

OpenCL ICD Loader Layers

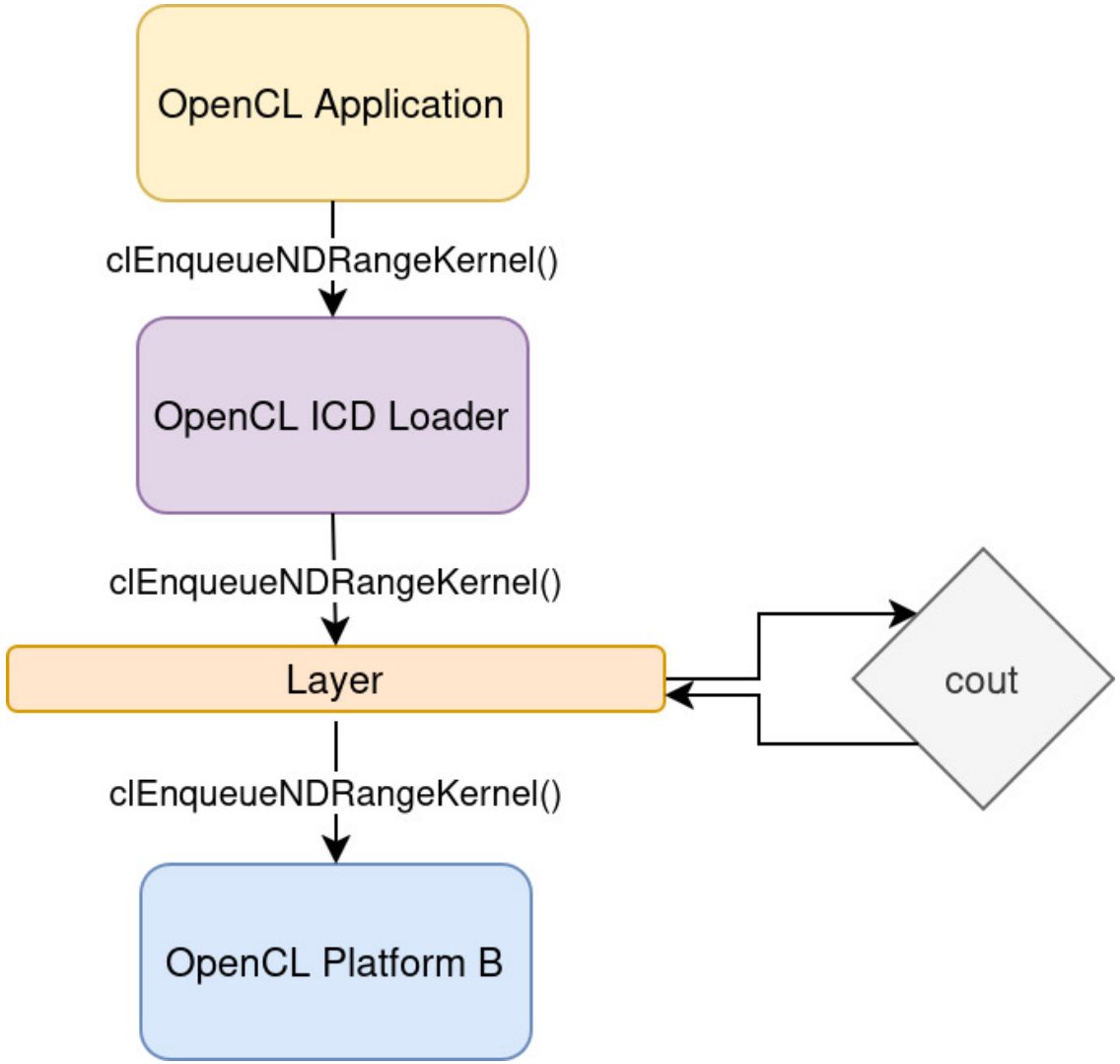


- Layers are shared libraries dynamically loaded at run time based on environment variables.
- With layers enabled the ICD Loader first redirects calls through active layers before the calling into the vendor driver.

What can you do with layers?

- Print API calls
- Object leak detection
- Emulate functionality
- Intel Intercept Layer
  - Inject programs/buffers/images
  - Capture & replay kernels
  - Chrome & Intel VTune tracing

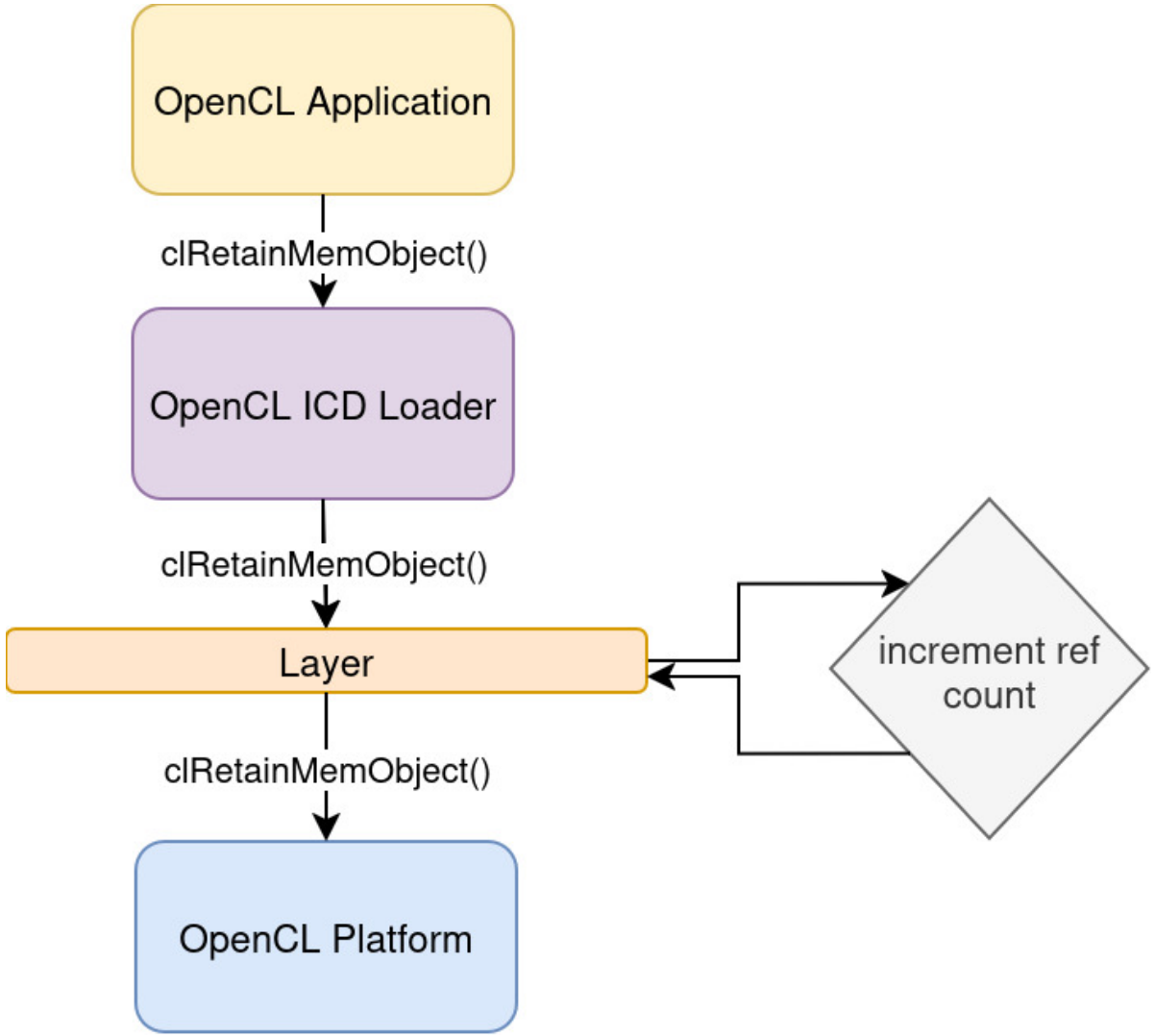
https://github.com/KhronosGroup/OpenCL-Layers



What can you do with layers?

STREAM
High Performance Computing

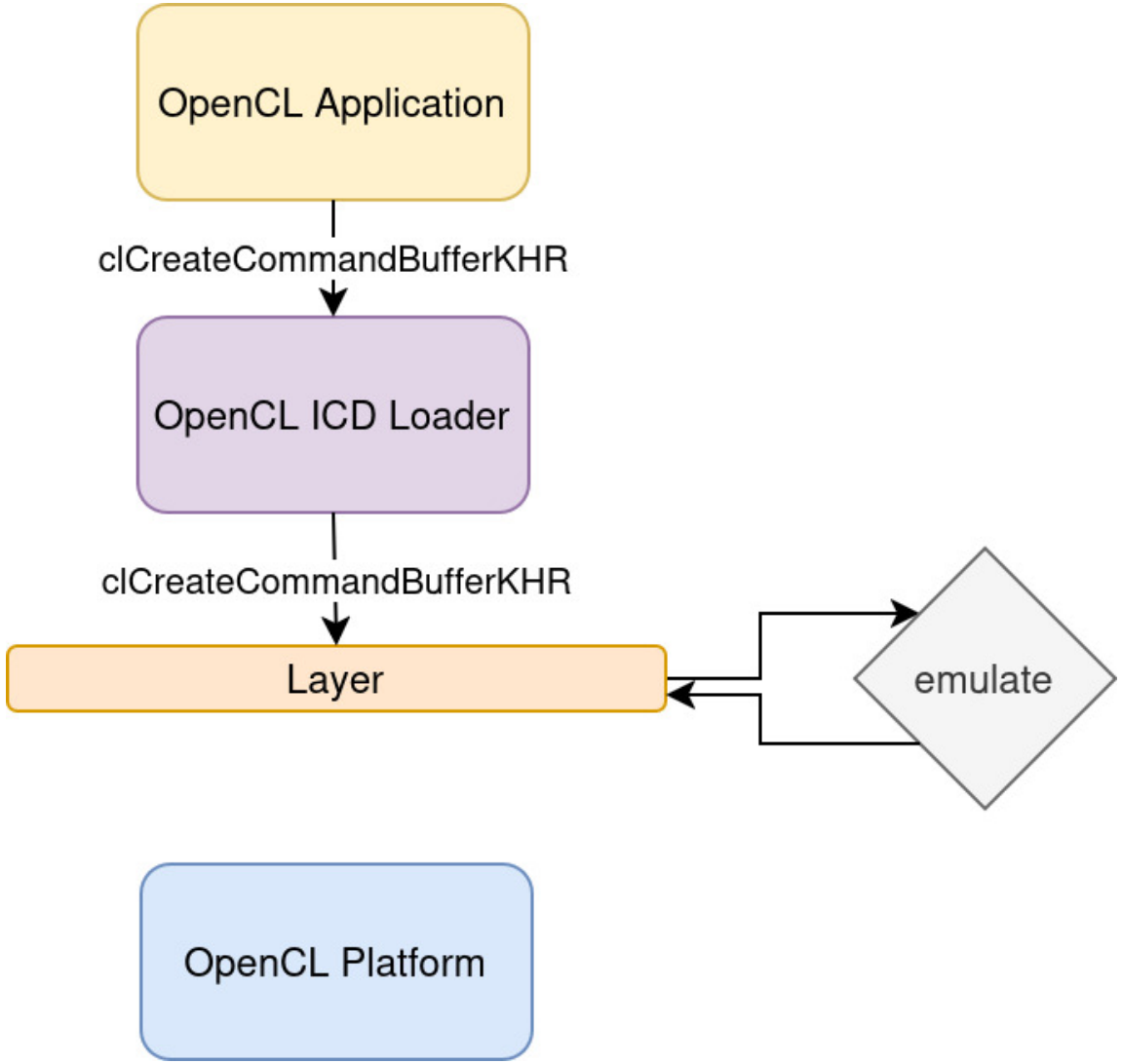
- Print API calls
- Object leak detection
- Emulate functionality
- Intel Intercept Layer
  - Inject programs/buffers/images
  - Capture & replay kernels
  - Chrome & Intel VTune tracing



https://github.com/KhronosGroup/OpenCL-Layers

What can you do with layers?

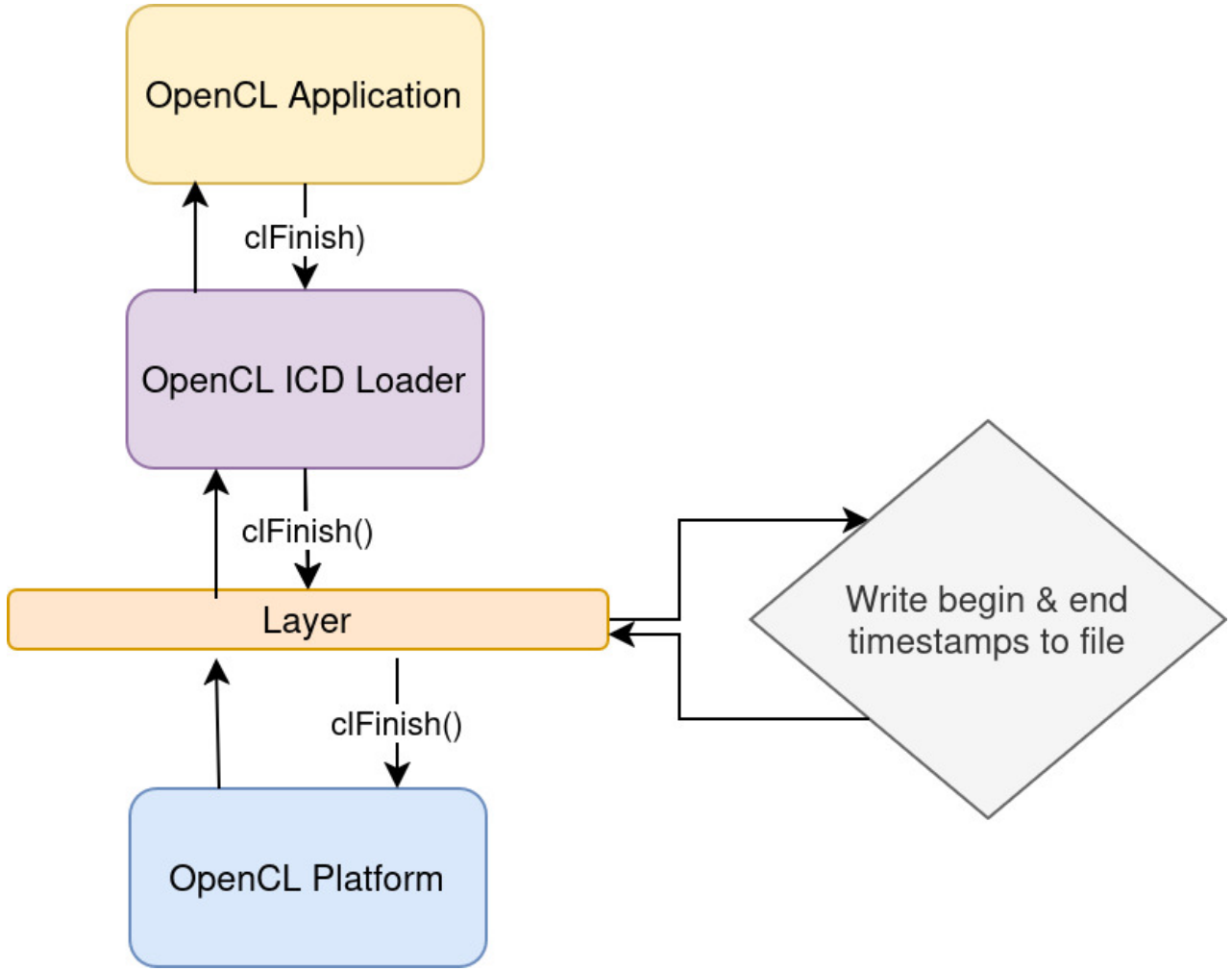
- Print API calls
- Object leak detection
- Emulate functionality
- Intel Intercept Layer
  - Inject programs/buffers/images
  - Capture & replay kernels
  - Chrome & Intel VTune tracing



https://github.com/bashbaug/SimpleOpenCLSamples

What can you do with layers?

- Print API calls
- Object leak detection
- Emulate functionality
- Intel Intercept Layer
  - Inject programs/buffers/images
  - Capture & replay kernels
  - Perfetto & Intel VTune tracing



https://github.com/intel/opencl-intercept-layer

Closer look at timeline tracing


```
$ cliloader -cdt  ./ooo_loop
```

https://github.com/EwanC/CLVizulayer/blob/main/test/ooo_loop.cpp

Using the Intel OpenCL Intercept Layer we can get a JSON timeline trace that is viewable in Perfetto.

Shows kernels named ‘no_op’ executing linearly on an out-of-order queue, but doesn’t show if that was enforced by cl_event dependencies or is implementation specific behavior.

There were no cl_event dependencies in this application.

CLVizulayer

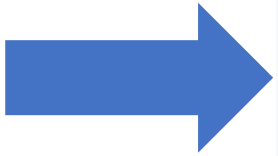
https://github.com/EwanC/CLVizulayer

CLVizulayer

STREAM
High Performance Computing

OpenCL application

```
cl_event Events[3];
clEnqueueNDRangeKernel(OutOfOrderQueue, Kernel, 1, nullptr, GlobalSize,
                       0, nullptr, &Events[0]);

clEnqueueNDRangeKernel(OutOfOrderQueue, Kernel, 1, nullptr, GlobalSize,
                       1, &Events[0], &Events[1]);

clEnqueueNDRangeKernel(OutOfOrderQueue, Kernel, 1, nullptr, GlobalSize,
                       1, &Events[0], &Events[2]);

cl_event LeafDeps[2] = {Events[1], Events[2]};
clEnqueueNDRangeKernel(OutOfOrderQueue, Kernel, 1, nullptr, GlobalSize,
                       2, LeafDeps, nullptr);

clFinish(OutOfOrderQueue);
```



DOT File

```
digraph CLVizulayer {
  node [style=bold]
  subgraph cluster_0 {
    label = "clFinish()";
    node_0[label="clEnqueueNDRangeKernel"];
    node_1[label="clEnqueueNDRangeKernel"];
    node_2[label="clEnqueueNDRangeKernel"];
    node_3[label="clEnqueueNDRangeKernel"];
  }
  node_0 -> node_1
  node_0 -> node_2
  node_1 -> node_3
  node_2 -> node_3
}
```

```
$ OPENCL_LAYERS=libCLVizuLayer.so VIZ_DOT_FILE=diamond.dot ./diamond_deps
```

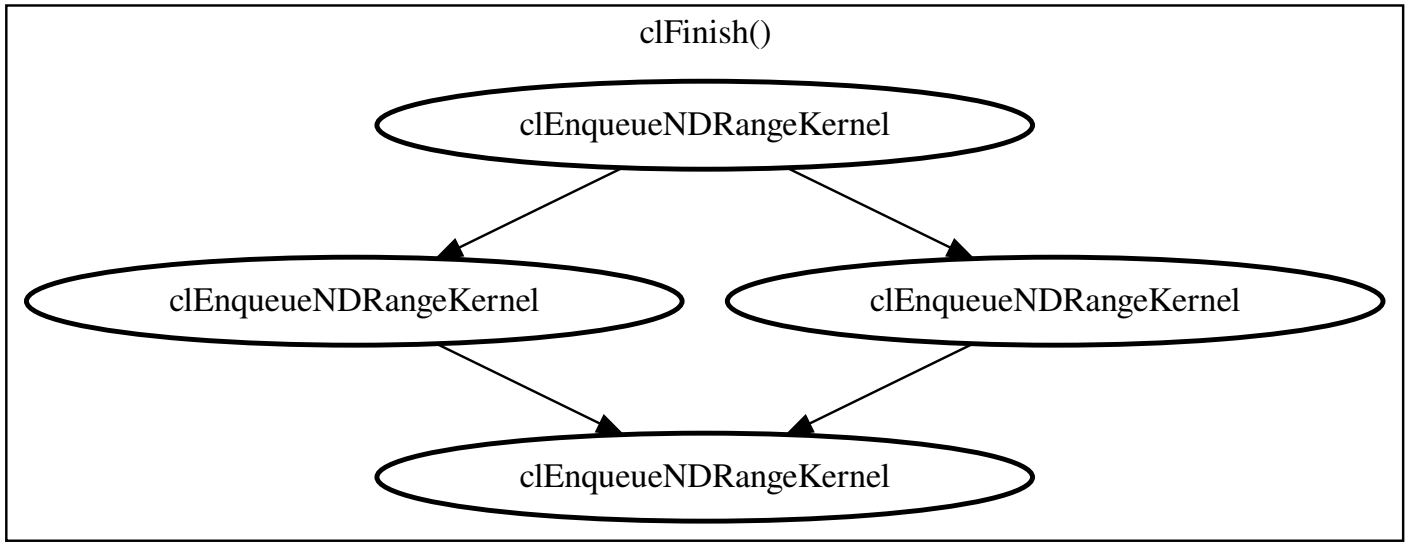
Graphviz DOT format

STREAM
High Performance Computing

DOT File

```
digraph CLVizulayer {
  node [style=bold]
  subgraph cluster_0 {
    label = "clFinish()";
    node_0[label="clEnqueueNDRangeKernel"];
    node_1[label="clEnqueueNDRangeKernel"];
    node_2[label="clEnqueueNDRangeKernel"];
    node_3[label="clEnqueueNDRangeKernel"];
  }
  node_0 -> node_1
  node_0 -> node_2
  node_1 -> node_3
  node_2 -> node_3
}
```

diamond.svg

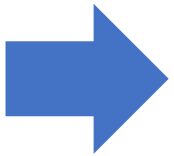


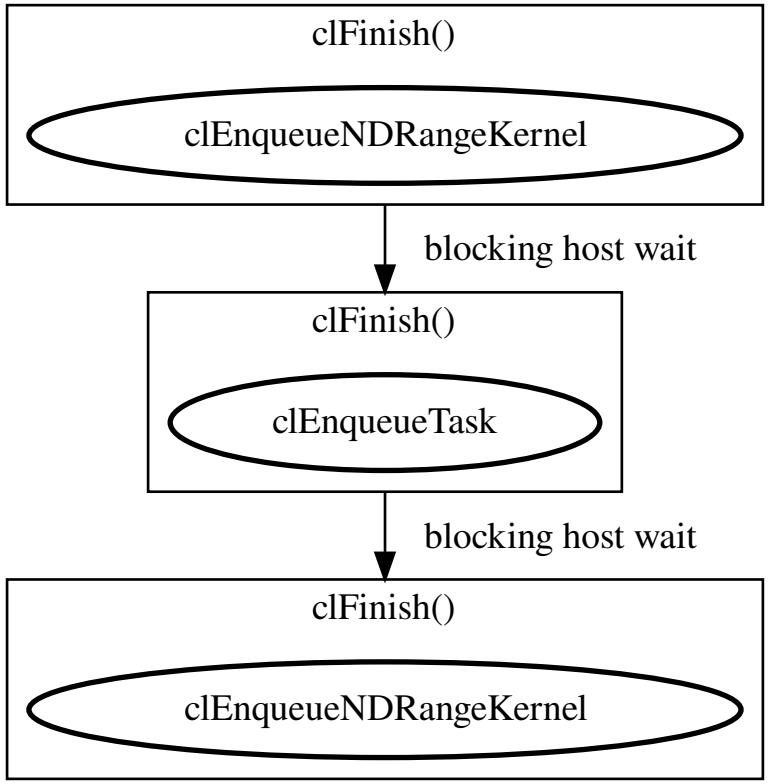

```
$ dot -Tsvg diamond.dot -o diamond.svg
```

Blocking host commands

STREAM
High Performance Computing

```
int main() {
  CLState State;

  cl_int Ret =
      clEnqueueNDRangeKernel(State.OutOfOrderQueue, State.Kernel, 1, nullptr,
                             &State.GlobalSize, nullptr, 0, nullptr, nullptr);
  CHECK(Ret);

  Ret = clFinish(State.OutOfOrderQueue);
  CHECK(Ret);

  Ret = clEnqueueTask(State.OutOfOrderQueue, State.Kernel, 0, nullptr, nullptr);
  CHECK(Ret);

  Ret = clFinish(State.OutOfOrderQueue);
  CHECK(Ret);

  Ret = clEnqueueNDRangeKernel(State.OutOfOrderQueue, State.Kernel, 1, nullptr,
                               &State.GlobalSize, nullptr, 0, nullptr, nullptr);
  CHECK(Ret);

  Ret = clFinish(State.OutOfOrderQueue);
  CHECK(Ret);

  return 0;
}
```





CLVizulayer: SYCL

STREAM
High Performance Computing

```cpp
#include <iostream>
#include <sycl/sycl.hpp>

int main() {
  sycl::float4 a = {1.0, 2.0, 3.0, 4.0};
  sycl::float4 b = {4.0, 3.0, 2.0, 1.0};
  sycl::float4 c = {0.0, 0.0, 0.0, 0.0};

  sycl::queue queue;
  {
    sycl::buffer<sycl::float4, 1> a_sycl(&a, sycl::range<1>(1))
    sycl::buffer<sycl::float4, 1> b_sycl(&b, sycl::range<1>(1))
    sycl::buffer<sycl::float4, 1> c_sycl(&c, sycl::range<1>(1))

    queue.submit([&](sycl::handler &cgh) {
      auto a_acc = a_sycl.get_access<sycl::access::mode::read>(cgh);
      auto b_acc = b_sycl.get_access<sycl::access::mode::read>(cgh);
      auto c_acc = c_sycl.get_access<sycl::access::mode::write>(cgh);

      cgh.single_task([=]() { c_acc[0] = a_acc[0] + b_acc[0]; });
    });
  }
  std::cout << "  A { " << a.x() << ", " << a.y() << ", " << a.z() << ", "
            << a.w() << " }\n"
            << "+ B { " << b.x() << ", " << b.y() << ", " << b.z() << ", "
            << b.w() << " }\n"
            << "------------------\n"
            << "= C { " << c.x() << ", " << c.y() << ", " << c.z() << ", "
            << c.w() << " }" << std::endl;

  return 0;
}
```

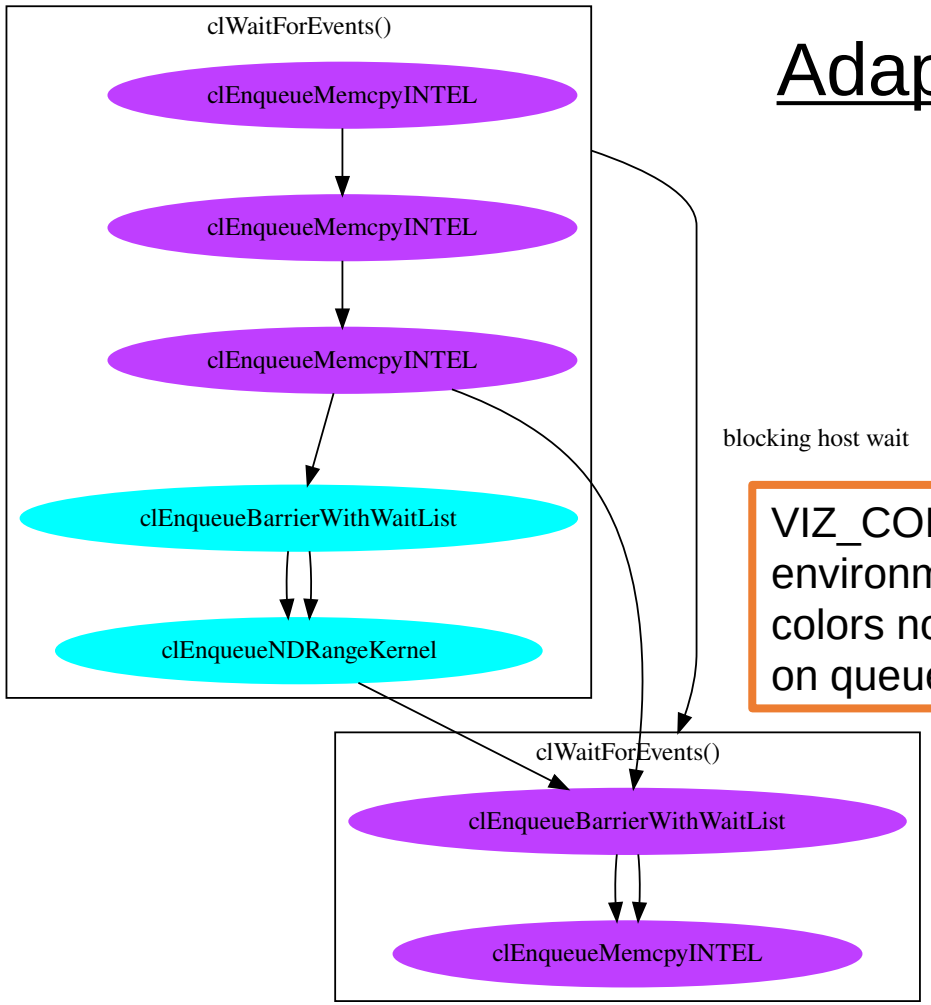
1
2
4
5

1. Buffers constructed with host pointer, buffer is initialized with this data.
2. Accessors tells runtime to make memory available on device associated with command.
3. Kernel command to run task on device
4. Buffer destruction, blocks until all queue work has completed using buffer and copies results back to host pointer used on
5.

```
  A { 1, 2, 3, 4 }
+ B { 4, 3, 2, 1 }
------------------
= C { 5, 5, 5, 5 }
```

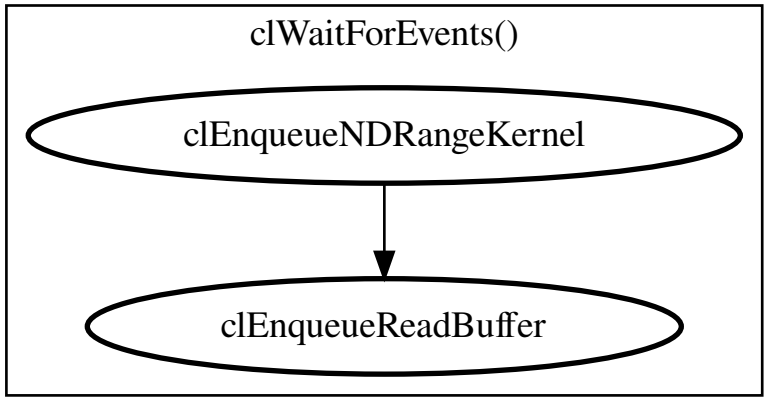
CLVizulayer: SYCL

AdaptiveCpp



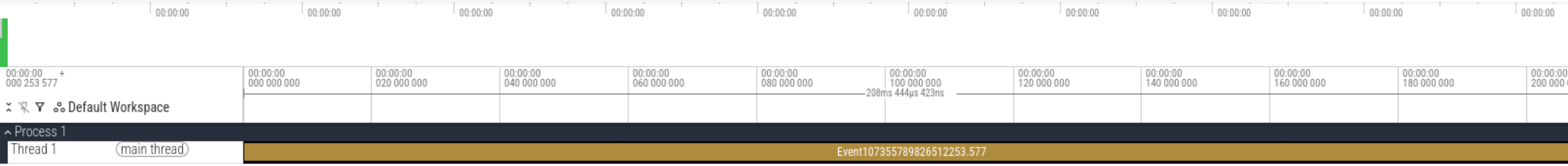
VIZ_COLOR=1 environment variable colors nodes based on queue

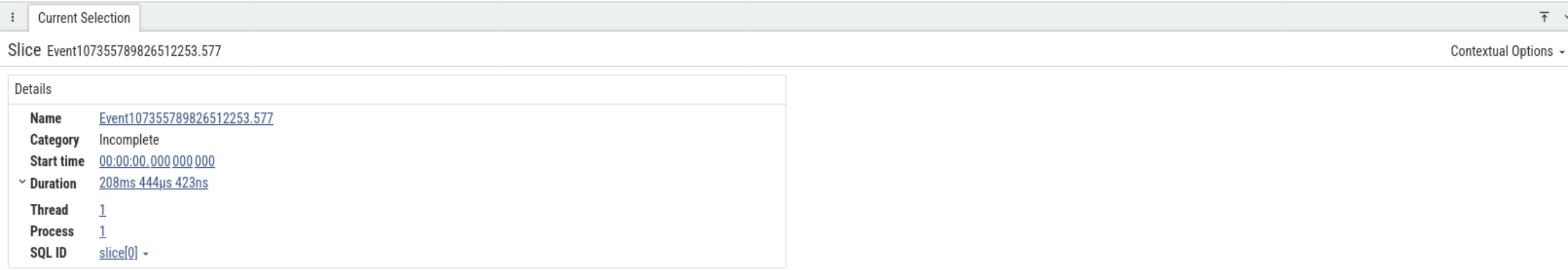
DPC++



Doesn't show clCreateBuffer + CL_MEM_USE_HOST_PTR

Comparison with SYCL DAG Trace





Tried tool from IWOCL 2026 “Lightweight Tracing Interface for SYCL’s USM Model Implemented in AdaptiveCpp” presented by Jakob Neissner. Tool intended for USM, so only shows a single incomplete event.

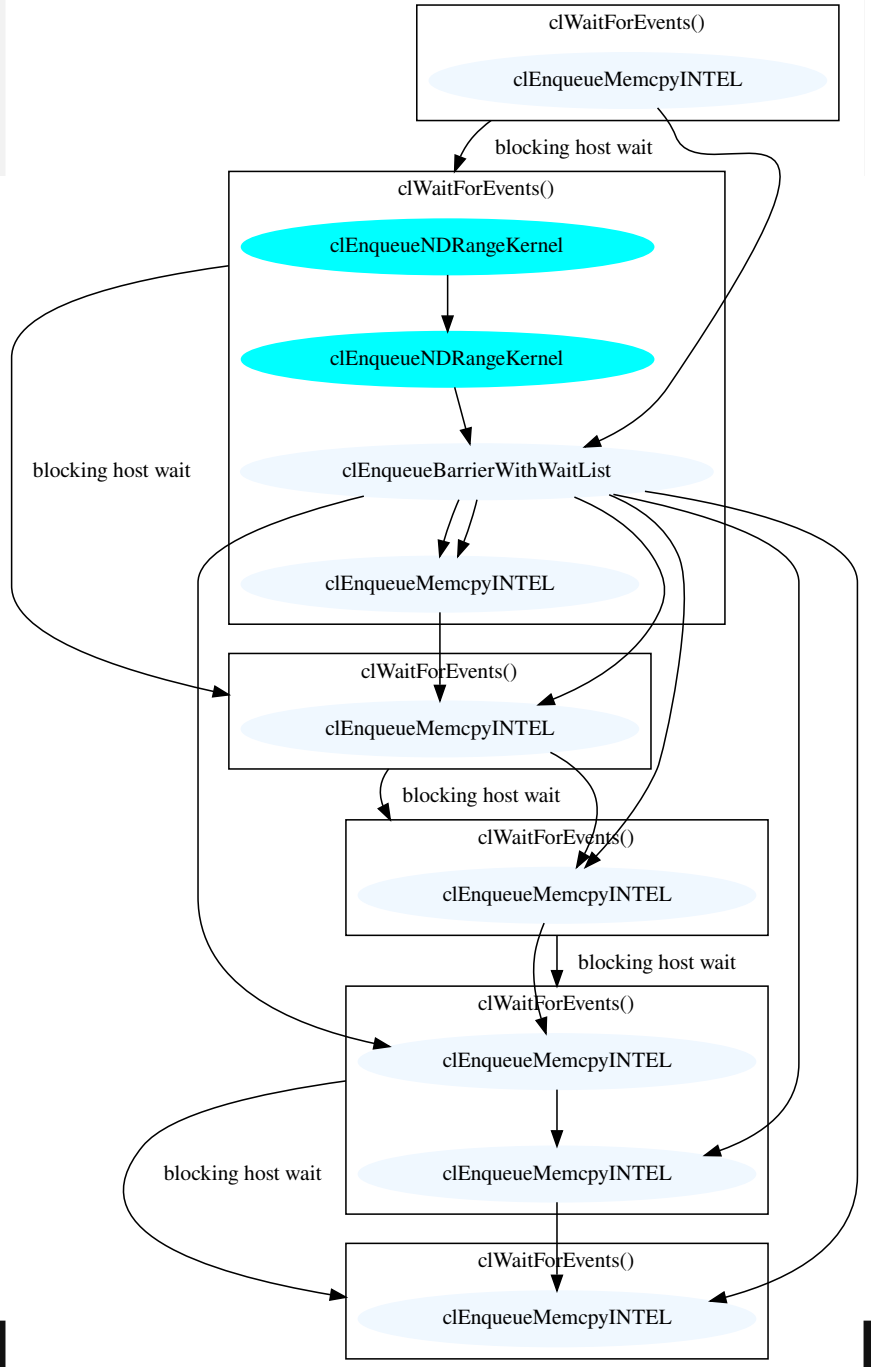
* https://github.com/AdaptiveCpp/AdaptiveCpp/pull/1946 commit 157ef19d22558c5bc6256a22373e6c9b02d3fde8
* https://github.com/jabruniessner/SYCL_tracing-examples/tree/master commit 2181979abb126bc7479cc7134623ee941cd5c379

```
SYCL_TOOL_LIBRARY=$HOME/SYCL_tracing-examples/build/libprint_dag.so ./vector_add/vector_add_acpp
```

SYCL: Shamrock

- Shamrock is an open-source, multi-GPU hydrodynamics framework for astrophysics that uses a SYCL backend.

- Using commit fb912782743a3a10dadc2f38b0337f2de747023f from 05/05/26 ran run_binary_orbit.py example on AdaptiveCpp OpenCL backend.



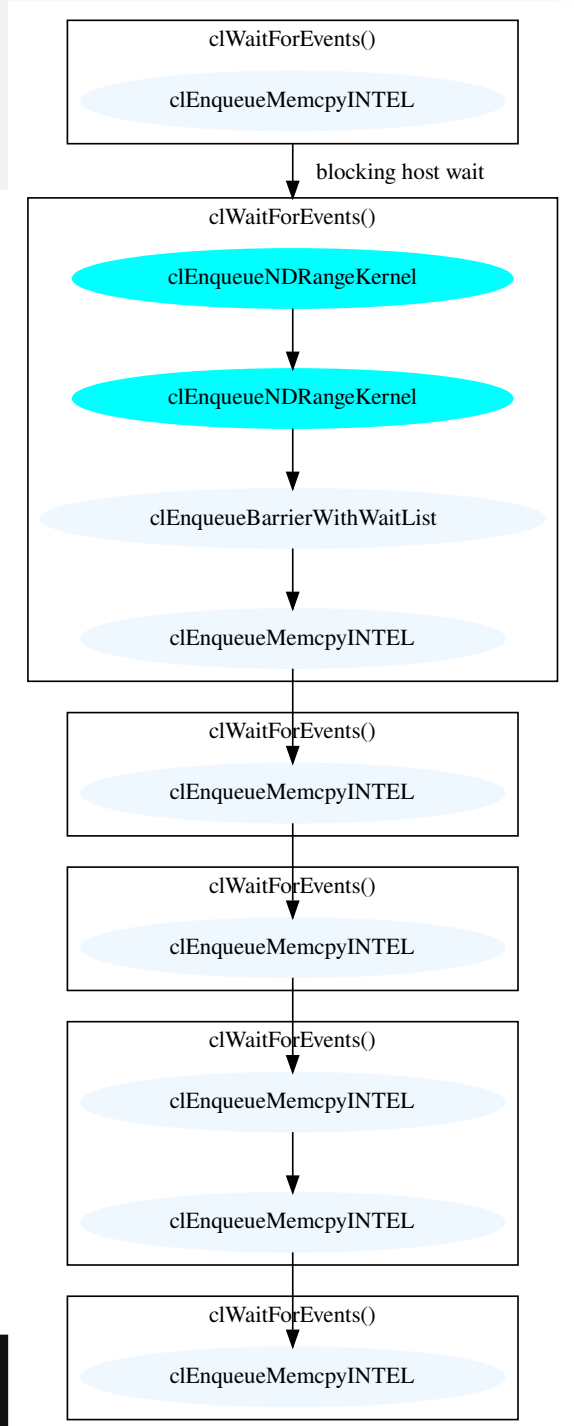
SYCL: Shamrock Tred

- Tred tool in standard graphviz installation for transitive reduction graph algorithm.

- Running tool show linear graph structure.

```
tred shamrock.dot > shamrock_tred.dot
```

- Availability of graph algorithm tooling in graphviz a benefit of using this format.



Case Study

Applications

I tried to find application using out-of-order queues that might give more interesting graph structure to analyze for this talk, but couldn’t.

Instead looks at graphs from a shortlist of the following applications with OpenCL backends, all of which create linear graphs.

- Leela Chess Zero
- GROMACS
- Llama.cpp
- LAMMPS

Extension cl_ext_dot_graph

STREAM
High Performance Computing

New Section 5.X - Dot Graph Printing

This section defines the mechanism for creating a DOT format file on disk containing a graph of the asynchronous device commands submitted to one or more command-queues. A handle to a dot graph object representing a file is created by passing a list of queues to print the submissions of. All user enqueue calls to those queues will be tracked in the dot file until the point when the dot file handle is destroyed.

To create a dot graph object, use the function

```
cl_dot_graph_ext clCreateDotGraphEXT(
   cl_uint num_queues,
   const cl_command_queue* queues,
   const cl_dot_graph_properties_ext* properties,
   const char* file_path,
   cl_int* errorcode_ret);
```

num_queues is the number of command-queues listed in queues.

queues is a pointer to a list of command-queues that a dot graph of commands will be created from. *queues* must be a non-NULL value and the length of the list equal to *num_queues*.

properties specifies a list of properties for the dot graph object and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. If a supported property and its value is not specified in *properties*, its default value will be used. *properties* may be NULL, in which case the default values for supported properties will be used. Supported properties defined by extensions are defined in the List of supported properties by clCreateDotGraphExt table.

file_path filesystem path to where dot file of commands will be created. If *file_path* is `NULL`, then a file named `clviz_YYYY-MM-DD_HH:MM:SS.dot` in the current working directory will be created. If *file_path* is an existing file, then that file will be overwritten.

errorcode_ret will return an appropriate error code. If *errcode_ret* is `NULL`, no error code is returned.

Table 1. List of supported properties by **clCreateDotGraphEXT**

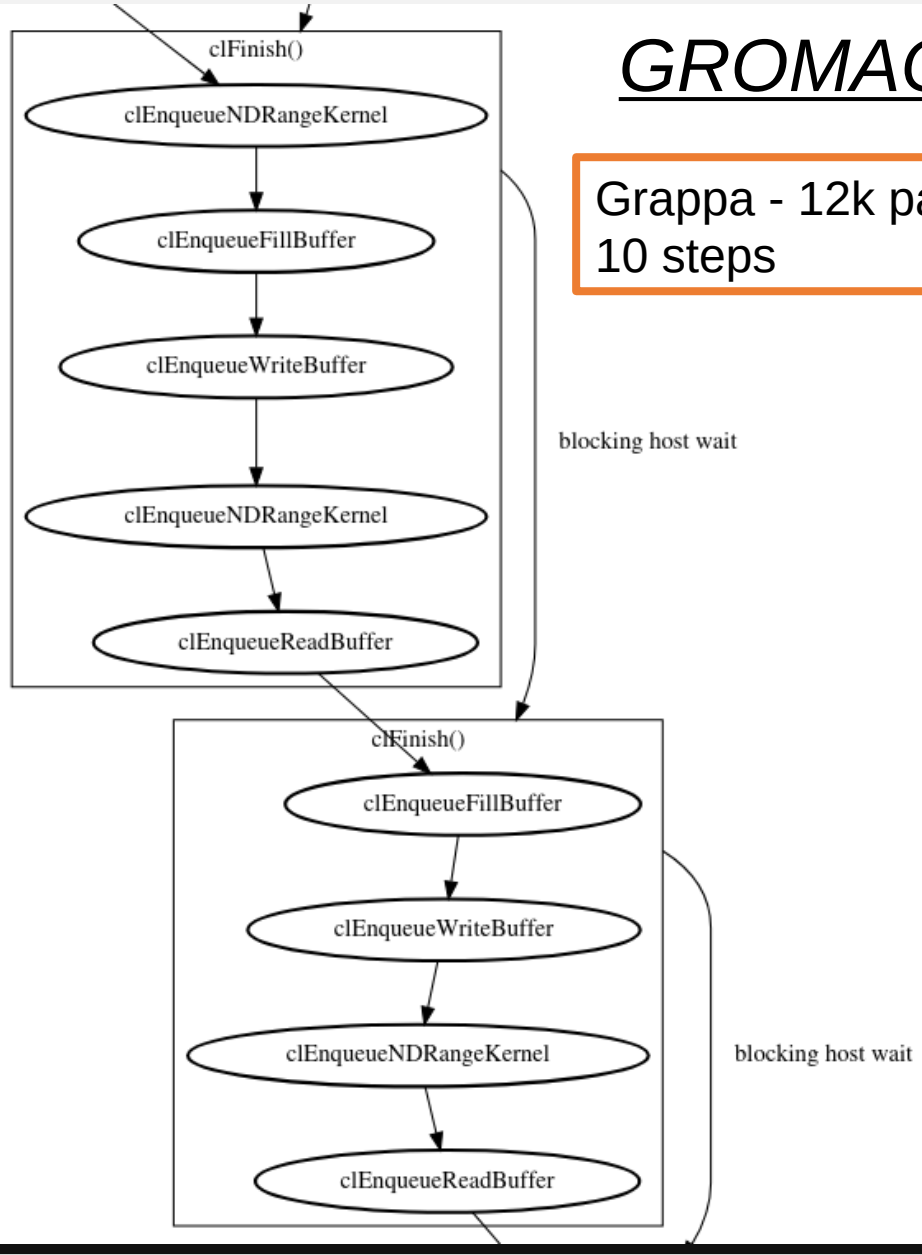
- Found that graphs printed can be too large to render, so defined cl_ext_dot_graph to allow developers to control what they want to capture.
- When VIZ_EXT=1 environment variable set, the layer will report extension rather than tracing full application

Snippets

STREAM
High Performance Computing

GROMACS

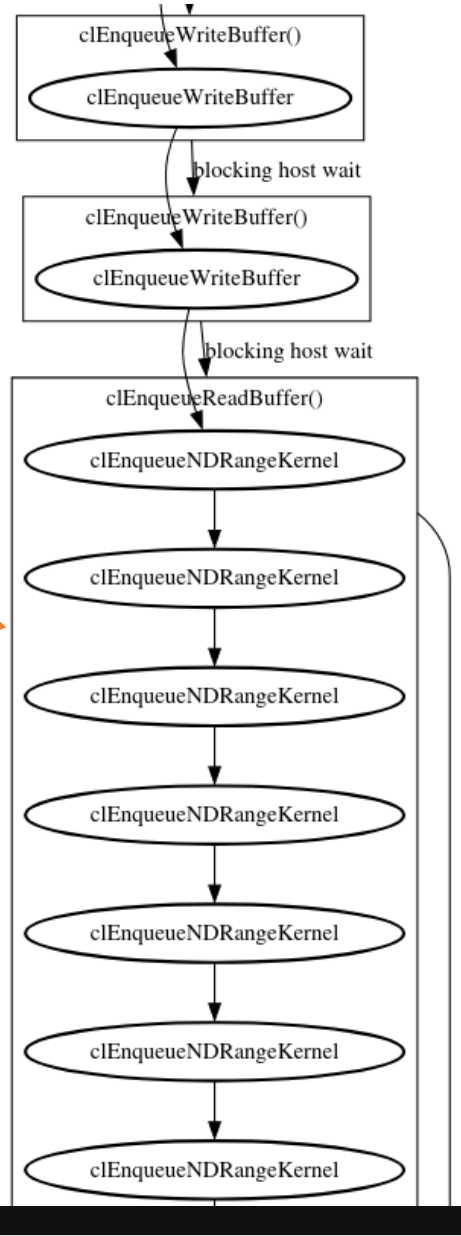
Grappa - 12k particles, 10 steps



Llama.cpp

```
./bin/llama-cli \
-no-cnv \
-m DeepSeek-R1-Distill-Qwen-1.5B-Q4_0.gguf \
-ngl 1 -n 1 -c 1024 \
 --no-mmap -sm none --color \
 -p "What are the top 10 most beautiful countries?"
```

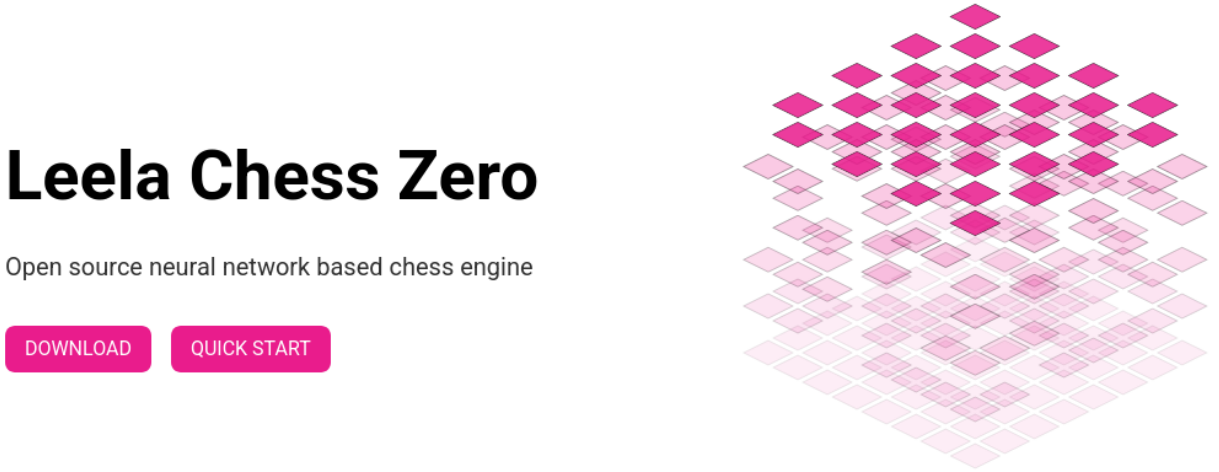
67 Kernels before next blocking wait



Leela Chess Zero (lc0)

STREAM
High Performance Computing

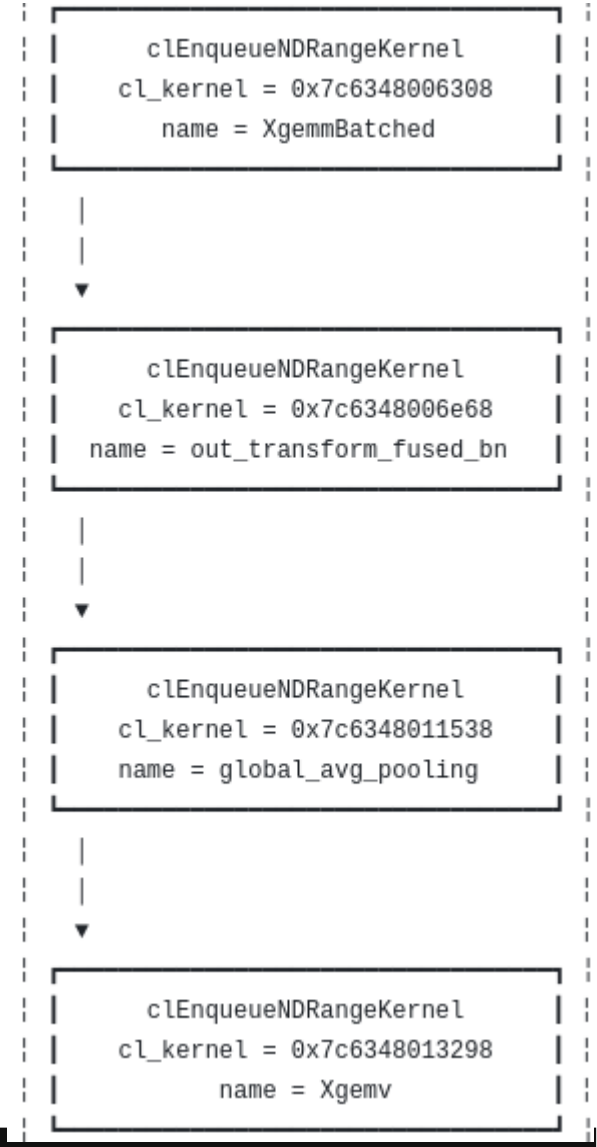
- Open source Universal Chess Interface (UCI) chess engine.
- Inspired by Deepmind's AlphaZero project, uses a neural network algorithm that learns through self play.
- Use Khronos OpenCL-HPP C++ headers
- OpenCL backend as well as cuda, metal, SYCL and more.



LC0 graph


```
OPENCL_LAYERS=libCLVizuLayer.so VIZ_VERBOSE=1 ./build/release/lc0 benchmark \
-w weights/T78 \
--num-positions=1 \
--nodes=1 \
--backend=opencl
```

VIZ_VERBOSE includes kernel names in DOT output



- Search a single position with a single node to keep graph small as possible.
- Linear graph with 378 nodes in a before a blocking command.

OpenCL Command-buffers

- cl_khr_command_buffer extension separates creation of a graph of device tasks from it's submission.
- A single submission reduces submission latency from many individual kernel submissions. Should be a good LC0 optimization based on DOT graph characteristics.
- The CUDA LC0 backend tries to use cuda-graph where possible

https://www.iwocl.org/wp-content/uploads/02-presentation-iwocl-syclon-2022-frankland.pdf

https://github.com/EwanC/lc0/pull/1

Command-buffer capture #1

Draft EwanC wants to merge 4 commits into `master` from `iwocl_dev`

Conversation 0 | Commits 4 | Checks 0 | Files changed 7

EwanC commented 2 days ago • edited — Owner

Prototype implementation of using `cl_khr_command_buffer` to capture the OpenCL network for the IWOCL 2026 talk on CLVizulayer.

Modifications:

- Update OpenCL-HPP headers to tip
- Prefer OpenCL devices with cl_khr_command_buffer_support
- Captures command-buffer for each batch size up to the max batch size when `OpenCLNetwork` is created by instantiating a temporary `OpenCLNetwork` object that sets up the command-buffers which live in an `OpenCLBuffer` object. Setup is run by running buffer `forward()` and using command-buffer append entry-points. On future invocations of the buffer object the created graph is executed.
- Use `cl_ext_command_buffer_dot_print` to print the captured graph in Debug build.
- Added `graph_capture` backend option so that users can opt-in to using command-buffer capture.

Modified LC0 graph

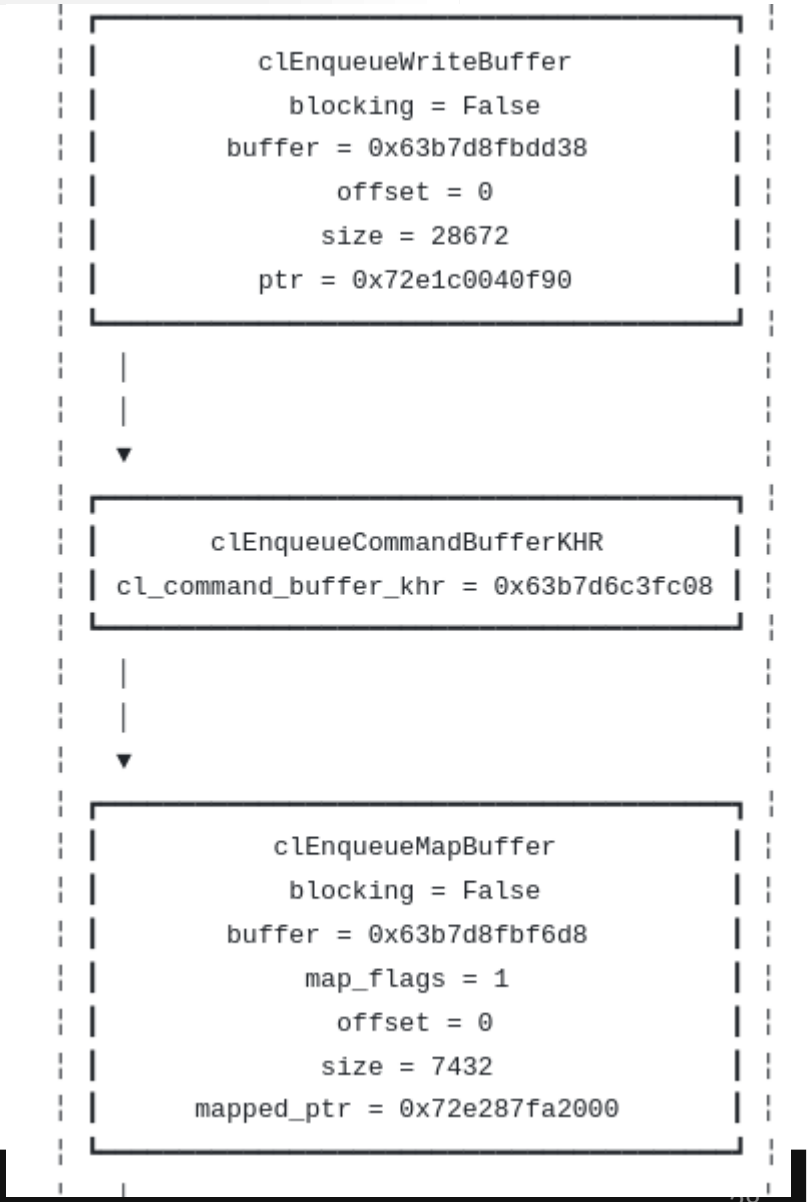
STREAM
High Performance Computing

```
OPENCL_LAYERS=libCLVizuLayer.so VIZ_VERBOSE=1 ./build/release/lc0 benchmark
-w weights/T78 \
--num-positions=1 \
--nodes=1 \
--backend=opencl \
--backend-opts=graph_capture=true
```

CLI option to enable command-buffer extension

Can see pattern of:

1. clEnqueueWriteBuffer call
2. clEnqueueCommandBuffer call with kernels.
3. Three clEnqueueMapBuffer calls
4. Three clEnqueueUnmapBuffer calls (after host memcpy mapped ptr to host data)



Extension cl_ext_command_buffer_dot_print

STREAM
High Performance Computing

New Section 5.17.X - Command-Buffer Printing

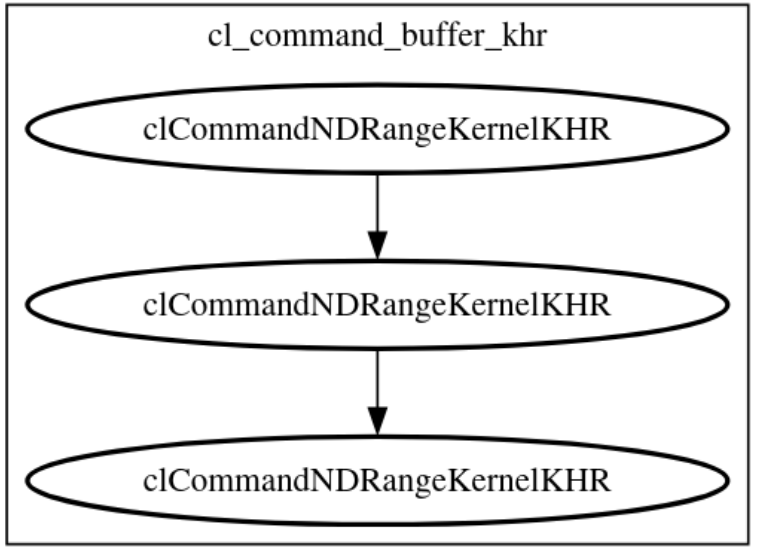
The commands in a command-buffer and their dependencies can be printed to a file in the DOT format using the function

```
cl_int clCommandBufferDotPrintEXT(
   cl_command_buffer_khr command_buffer,
   const cl_command_buffer_dot_print_properties_ext* properties,
   const char* file_path)
```

command_buffer handle to a command_buffer object to print a dot graph of commands.

properties specifies a list of properties for for configuring printing of the command-buffer object to the dot file, and their corresponding values. Each property name is immediately followed by the corresponding desired value. The list is terminated with 0. If a supported property and its value is not specified in *properties*, its default value will be used. *properties* may be NULL, in which case the default values for supported properties will be used. Supported properties defined by extensions are defined in the List of supported properties by clCcommandBufferDotPrintExt table.

file_path filesystem path to where dot file of commands will be created. If *file_path* is `NULL`, then a file named `clviz_YYYY-MM-DD_HH:MM:SS.dot` in the current working directory will be created. If *file_path* is an existing file, then that file will be overwritten.



Command-buffer graph

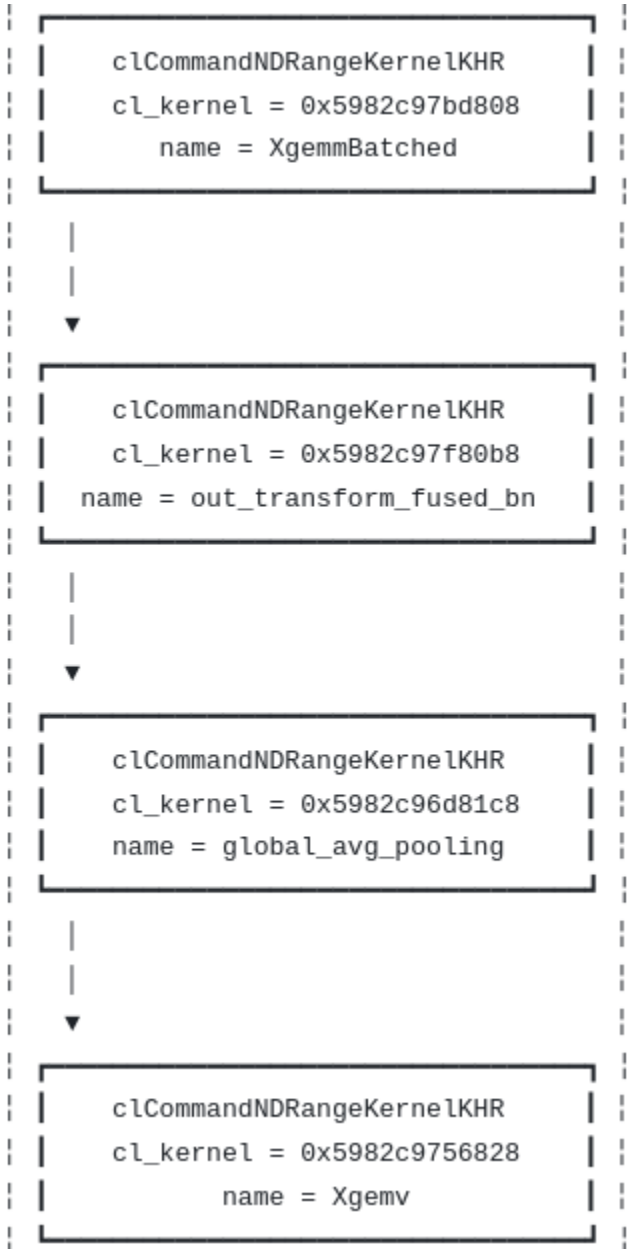
STREAM
High Performance Computing

```
OPENCL_LAYERS=libCLVizuLayer.so VIZ_EXT=1 ./build/debug/lc0 benchmark \
-w weights/T78 \
--num-positions=1 \
--nodes=1 \
--backend=opencl \
--backend-opts=graph_capture=true
```

VIZ_EXT to enable CLVizulayer cl_ext_command_buffer_dot_print extension

In `clDotPrintCommandBufferEXT` output can see 374 kernel calls in command-buffer

- Used `CL_COMMAND_BUFFER_DOT_PRINT_FLAGS_EXT` flag to enable verbose printing of kernel names.



Benchmarking

- Intel OpenCL CPU driver has native support for cl_khr_command_buffer, as opposed to needing the emulation layer.

- Compared enabled vs disabled command-buffer path without the CLVizulayer enabled to see any benefits of the potential optimization.

```
 _ _ _
| | _ | |
|_|_|_|_| v0.33.0-dev+git.dirty built Apr 21 2026
Loading weights file from: weights/T78
OpenCL, maximum batch size set to 16.
Initializing OpenCL.
Detected 1 OpenCL platforms.
Platform version: OpenCL 3.0 LINUX
Platform profile: FULL_PROFILE
Platform name:    Intel(R) OpenCL
Platform vendor:  Intel(R) Corporation
Device ID:        0
Device name:      Intel(R) Core(TM) Ultra 7 155H
Device type:      CPU
Device vendor:    Intel(R) Corporation
Device driver:    2026.20.1.0.12_160000
Device speed:     0 MHZ
Device cores:     22 CU
Device score:     2530
Selected platform: Intel(R) OpenCL
Selected device: Intel(R) Core(TM) Ultra 7 155H
with OpenCL 3.0 capability.
Loaded existing SGEMM tuning for batch size 16.
Wavefront/Warp size: 128

Graph Capture enabled: 1
Max workgroup size: 8192
Max workgroup dimensions: 8192 8192 8192
```

Benchmark Results

STREAM
High Performance Computing

```
./build/release/lc0 benchmark -w weights/T78  --backend=opencl --backend-opts=graph_capture=true -t 1
```

	graph_capture=false	graph_capture=true
Total time (ms)	375519	376963
Nodes searched	1551	1713
Nodes/second	**4.1**	**4.5**

9.75% improvement

- Benchmarks 34 different board positions.
- No limit on nodes to search in tree per position, more enqueues of command-buffer.
- Single threaded: command-buffer modifications not yet thread safe

Conclusion

Future Work

- Interact with the Leela Chess Zero community to see if interest in cleaning up and upstreaming cl_khr_command_buffer modifications.
- Interact with OpenCL WG to see if any interest in formalizing the extensions.
- Bring experiences to discussions around SYCL tooling.

Summary

- OpenCL ICD Loader provides an layer mechanism that allow the user to easily load tooling for their application.
- CLVizulayer uses this mechanism to create a Graphviz DOT file of the asynchronous device submissions and the dependencies of each submission.
- This OpenCL vendor agnostic visualization can be used to debug and optimize an application as shown with Leela Chess Zero modifications.
- Please try it out and leave feedback as a repo Issue or Discussion.

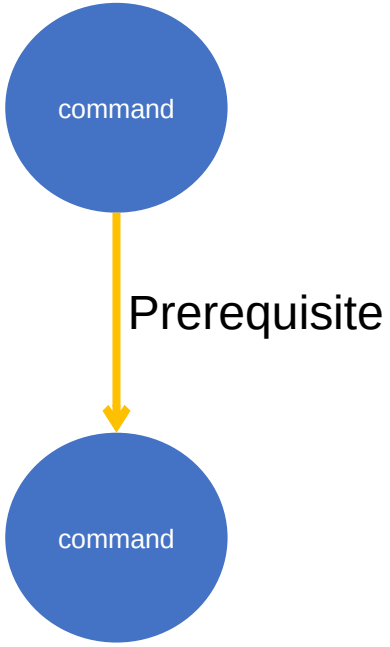
Backup

OpenCL Execution Model

STREAM
High Performance Computing

A command submitted to a device will not launch until prerequisites that constrain the order of commands have been resolved. These prerequisites have three sources:

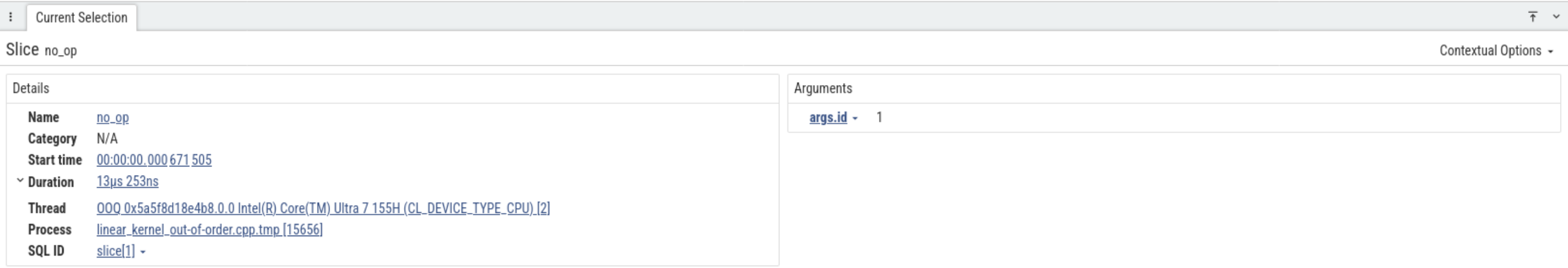
- The first source of prerequisites is implicit dependencies between commands enqueued to the same command-queue which arise as follows:
  - Commands enqueued after a command-queue barrier have the preceding barrier command as a prerequisite.
  - Commands enqueued in an in-order command-queue have the command enqueued before them as a prerequisite.
- The second source of prerequisites is dependencies between commands expressed through events. A command may include an optional list of events. The command will wait and not launch until all the events in the list are in the state `CL_COMPLETE`. By this mechanism, event objects define order constraints between commands and coordinate execution between the host and one or more devices.



https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#_execution_model

Closer look at timeline tracing

STREAM
High Performance Computing




```
./cliloader/cliloader -cdt linear_kernel_out-of-order
```

https://github.com/EwanC/CLVizulayer/blob/main/test/linear_kernel_out-of-order.cpp