IWOCL 2024
The 12th International Workshop on OpenCL and SYCL

Events Events Events

James Brodman, Intel
Ben Ashbaugh, Michael Kinsner, Steffen Larsen, Greg Lueck, John Pennycook, Roland Schulz – Intel
Gordon Brown - Codeplay
I, FONE
WELCOME TOUR NEW EVENT OVERLORDS.

...clearly NOT using SYCL
Dependences in SYCL

Two views:
1. Data - buffers/accessors

```
int main() {
    auto R = range{num};
    buffer<int> A[R], B[R];
    queue Q;

    Q.submit([&](handler& h) {
        accessor out{A, h};
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 1 */
        });
    });

    Q.submit([&](handler& h) {
        accessor inout{A, h};
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 2 */
        });
    });

    Q.submit([&](handler& h) {
        accessor out{B, h};
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 3 */
        });
    });

    Q.submit([&](handler& h) {
        accessor in{A, h, read_only};
        accessor inout{B, h, read_write};
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 4 */
        });
    });
}
```
Dependences in SYCL

Two views:

2. Task - events

```cpp
int main() {
    auto R = range{num};
    int *A, *B = /* ... */;
    queue Q;

    event e1 = Q.submit([&](handler& h) {
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 1 */
        });
    });

    event e2 = Q.submit([&](handler& h) {
        h.depends_on(e1);
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 2 */
        });
    });

    event e3 = Q.submit([&](handler& h) {
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 3 */
        });
    });

    Q.submit([&](handler& h) {
        h.depends_on({e2, e3});
        h.parallel_for(R, [=](auto idx) {
            /* Kernel 4 */
        });
    };
}
```
SYCL Events

The hammer of synchronization and utility!

- In SYCL 2020, every command returns an event
- USM relies on event-based synchronization (OoO queues)
- Command execution status is reported via an event
- Profiling info is provided via an event

... if they’re so useful, what’s the problem?
Nothing is ever free.
Or ... you pay for what you get:

- Event creation has a cost
  - In some implementations, a backend event is ALWAYS created
  - Sometimes events are never used (or immediately destroyed)!
    - In-order queues
    - No cross-queue synchronization

- Event profiling is all-or-none
  - Profiling enabled via property at queue creation
  - Profiling enabled for ALL commands in queue
Event Profiling

Event profiling is enabled via a queue property.
- Enabled at queue creation
- Can’t be toggled on/off later

Works, but too coarse
- Sometimes only parts of execution matter
- Code may want to easily toggle profiling
- Semantic mismatch when migrating code from CUDA/HIP
Can we do better?

Existing Extensions for Unwanted Events:

- "Discard Events" – DPC++
  - `sycl_ext_oneapi_discard_queue_events`
  - Queue property that doesn’t *really* create events

- "Coarse Grained Events" – AdaptiveCpp
  - Queue/Command Group property to optimize behavior, assuming you won’t usually try to use the events
Didn’t that fix it?

Neither solution is ideal:

- Discard events leaves ticking time bombs in the code
- Both require programmer annotations
- Annotations may be separate from use
  - Harder to reason about what code does and how it acts
Do more to get less!
Solution: More Extensions!

How standards proliferate: (See: A/C chargers, character encodings, instant messaging, etc.)

SITUATION: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone’s use cases. YEAH!

SOON:

SITUATION: There are 15 competing standards.

https://xkcd.com/927/
Complexity should be “opt-in”

“Events by default”
- In SYCL for ages
- Semantic break from OpenCL
  - Events are optional in OpenCL
  - Why did SYCL make them mandatory?

Queue-based event profiling
- Too coarse
- Too inflexible

Time to fix our technical debt.
New Enqueue Functions to the Rescue!

The best way to eliminate events is to never create them at all.

- `sycl_ext_oneapi_enqueue_functions`
- Two new APIs:
  1) `template <typename CommandGroupFunc>
     void submit(sycl::queue q, 
                  CommandGroupFunc&& cgf);`
  2) `template <typename CommandGroupFunc>
     sycl::event submit_with_event(sycl::queue q, 
                                    CommandGroupFunc&& cgf);`
Profiling Tags

Let the user decide what to profile when.

- Profiling tags extension
  - `sycl_ext_oneapi_profiling_tag`

- One new API:
  - For OoO queues, synchronizes with a barrier that is used as a marker
  - For in-order queues, just enqueues a marker

```cpp
#include <iostream>
#include <sycl/sycl.hpp>
namespace syclx = sycl::ext::oneapi::experimental;
namespace prof = sycl::info::event_profiling;

static constexpr size_t N = 1024;

int main() {
    sycl::queue q; // q created WITHOUT profiling enabled

    // commands submitted here are not timed
    sycl::event start = syclx::submit_profiling_tag(q);
    syclx::parallel_for(q, {N}, [=](auto i) {/* first kernel */});
    syclx::parallel_for(q, {N}, [=](auto i) {/* second kernel */});
    sycl::event end = syclx::submit_profiling_tag(q);
    q.wait();

    uint64_t elapsed =
        end.get_profiling_info<prof::command_start>() -
        start.get_profiling_info<prof::command_end>();
    std::cout << "Execution time: " << elapsed << "(ns)\n";
    return 0;
}
```

*Works for most backends. OpenCL will require an extension.*
Summary

Events are an incredibly powerful part of SYCL!
- Tool for synchronization and measurement
- Current use in SYCL may have drawbacks

Proposed new extensions:
- New Enqueue Functions
- Profiling Tags

Pay for what you use - not for what you don’t!
Questions?

Check out DPC++: [intel/llvm: Intel staging area for llvm.org contribution. Home for Intel LLVM-based projects. (github.com)](github.com)

Check out oneAPI Toolkits: [oneAPI: A New Era of Heterogeneous Computing (intel.com)](intel.com)

Comment/Contribute for SYCL Upstreaming to Clang!