Not Paying for Events We Don't Use

Bringing "Zero-overhead principle" to SYCL events

Thomas Applencourt

July 6, 2023

Table of Contents

Context

Where did it start? 404

- Feedback from Gromacs developer at the April 25th SYCL Advisory Panel Meeting
- Gromacs performance is bounded by the latency of commands submissions [1]
- *https://github.com/KhronosGroup/SYCL-Docs/issues/404*

[1] See Talk as IOWCL: Comparing the Performance of SYCL Runtimes for Molecular Dynamics Applications

Typical Application Use-Case

Simple case:

```
1  Q.submit(...);
2  Q.submit(...);
3  Q.submit(...);
4  Q.wait();
```

The SYCL events are not used. But the SYCL runtime generates native events [2]. *Applications pay a price for events they don't use*

[2] No choice in Library only implementation, "poor" dead-code elimination for compiler-based implementation

What Changes can be Made to the SYCL API?

How to Pass the Information?

Users will need to opt-in to this optimization. How?

- New property to the queue?
- New property list to the command group handle?
- New command submission API?

What Changes can be Made to the SYCL API?

New Queue Property

New Queue Property

- Pro: Easiest to implement. Queue have already a property list (openSYCL and oneAPI followed this path).

```
1  //explicit queue(const property_list& propList = {});
2  queue Q{property::queue::optimize_away_event};
```

- Con: ”We need a per-submission setting. We don’t use per-queue settings currently.”

What Changes can be Made to the SYCL API?

Alternate Command Submission API

Alternate Command Submission API

- Pro: Easy to implement (for application and implementer). Conceptually Simple.

```
1  template <typename T>
2  void queue::enqueue(T cgh)
```

- Cons: No more shortcut (*parallel_for*,...)
  - Gromacs feedback about no-more-shortcut: ”That sounds reasonable”

What Changes can be Made to the SYCL API?

Command Group Property

Command Group Property With Shortcut

Adding a property list to queue::submit.

```
1  template <typename T>
2  event queue::submit(const property_list& prop_list, T cgf)
3
4  template <typename T>
5  event queue::parralel_for(range<Dims> numWorkItems,
6                            const property_list& prop_lis
7                            Rest&&... rest)
```

- Pro: Can use shortcuts, and not to big spec change (need to add a property list to command group handler)[3]
- Maybe be a breaking change[4]
- Side Note: Shortcut Potentially Hard to Implement for Implementer

[3] We will discuss later if this property needs to be a compile or runtime one, or if or we should return an event or not

[4] Saw code breaking when adding a new template default arguments in SYCLCplx

Command Group Property Without Shortcut

- Pro: Easier to implement.[5]

```
1  template <typename T>
2  event queue::submit(const property_list& prop_list, T cgf)
```

- Cons: No more shortcut (gromacs feedback about no-more-shortcut: ”That sounds reasonable”)
- Side note: Complexity a little bit the spec

[5] We will discuss if this property needs to be a compile or runtime one or if it should return an event or not later.

How to Implement the ”Zero-Overhead Principle” for SYCL Event?

How to Implement the ”Zero-Overhead Principle” for SYCL Event?

Changing the return type

Alternate Command Submission API

- Either: New Alternate Command Submission API

```
template <typename T>
void queue::enqueue(T cgh)
```

- Or Compile Time Property:

```
template <typename KernelName, int Dims, typename... Rest>
void queue::parallel_for<property_list_t property_list= {}
↪  >(range<Dims> numWorkItems, Rest&&... rest);
```

  - Pro: Cannot be misused
  - Cons: Hard to write generic code.
  - Note1: Not clear as an application user why we need this new API.
  - Note2: Will obviously not allow any profiling

Example of oneMKL

- oneMKL provide hundreds of APIs. Already duplicated for Buffer and USM.
- Will it it need to duplicated one more time?

```
namespace oneapi::mkl::blas::column_major {
    void scal(sycl::queue &queue,
              std::int64_t n,
              Ts alpha,
              sycl::buffer<T,1> &x,
              std::int64_t incx)
}
namespace oneapi::mkl::blas::column_major {
    sycl::event scal(sycl::queue &queue,
                     std::int64_t n,
                     Ts alpha,
                     T *x,
                     std::int64_t incx,
                     const std::vector<sycl::event> &dependencies = {})
}
```

How to Implement the ”Zero-Overhead Principle” for SYCL Event?

Performance Hint

Hint

Add a property list hint to specify that the event will not be used.

```
1 Q.parallel_for( propery_list={hint::event_not_used} ,{..});
```

Runtimes responsibility to leverage the hints or not[6]

- Pro: Easiest Adoption by existing code. Keep fredom to implementer on how to deal with special case (in-order, profiling, ...)
- Con: Can be ”miss-used”
- Note: Clear for Applications that it’s a performance flags.

[6] We already have a precedent with queue::prefetch

Example of Implementation: oneAPI

- Following *sycl_ext_oneapi_discard_queue_events spirit*, the runtime can return a ”dummy event”.
- And throw if people are using this event.


```
1  Q.submit(propery_list={hint::event_not_used}...);
2  Q.wait(); // Good
3  e = Q.submit(propery_list={hint::event_not_used}...);
4  e.wait(); // Throw
```

Example of Implementation: OpenSYCL

With CUDA/HIP backend:

```
e = Q.submit(...); // cudaLaunchKernel();  cudaEventRecord()
e.wait(); // cuEventSynchronize();
Q.wait(); // cuStreamSynchronize();
```

Following "as-if" optimization, this can be replaced with:

```
Q.submit(propery_list={hint::event_not_used}...); // cuLaunchKernel()
Q.wait(); // cuStreamSynchronize();
```

When people are still using event[7]:

```
e = Q.submit(propery_list={hint::event_not_used}...); // cuLaunchKernel()
e.wait(); // cuStreamSynchronize();
```

[7] they should not, they broke the contract. But let not broke their code

Example of Implementation: OpenSYCL

- No more CUDA events created/used! No more overhead if not used. If used, don't break code.
- Optimization "as-if." No semantic difference.
- This is openSYCL: `openSYCL_ext_coarse_grained_events`.
- Work with dependency, and `get_native`

Conclusion

Conclusion

- We need to decide what API change we can afford
- And then choose the implementation to use.