Argonne NATIONAL LABORATORY

KHRONOS GROUP

SYCL and Kokkos

Similarities and Differences

Thomas Applencourt — *apl@anl.gov*

September 16, 2026

Argonne National Laboratory

Outline

About me

- Arrived here by chance, not a ”CS” guy by training
  - Part of the Performance Engineering Group at Argonne.
- Worked on Aurora for the past ∼10 years.
  - I may have some Stockholm syndrome / PTSD with Intel hardware.
  - But I’m now somehow considering myself as a runtimes guy (SYCL, OpenMP, L0...)
- Chair of the Khronos SYCL Working Group for the past year or so.
- Disclaimer: I will talk as Thomas, not as a SYCL chair

Before we start: portability layers are not that important

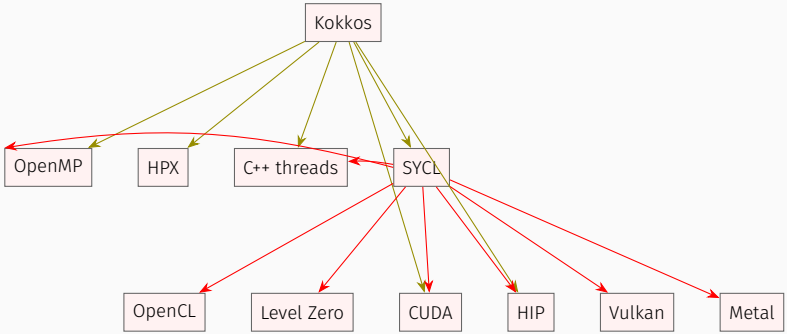
- They exist so people can do science, paper, whatever pay the bill
- They can all: allocate memory, transfer data, submit a kernel
- Good enough
- In my view, if they have drastic performance difference, it's a bug.

Just pick one, and enjoy. Just don't write your own pretty please.

- There are a LOT of driver bugs to work around if you want to be actually portable
- Not fun. A waste of your time. A waste of resources
- Instead: open issue, submit PR, improve the ecosystem[1]

[1] Kokkos is the good example: they needed some library, they pushed it into the C++ main spec (thanks ***mdspan***!)

Relation between SYCL and Kokkos: Layer

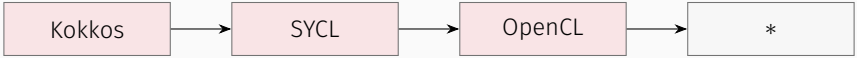


- On Aurora, the path is Kokkos → SYCL [2] (plus a few Intel extensions, as far as I know).

[2] Daniel is amazing. Thanks Daniel!

My dream, be nice don't laugh

Just to avoid duplication of efforts:



Outline

1. Where I am coming from
2. What SYCL is (and is not)
3. Similarities and differences with Kokkos
4. Naming convention
5. SYCL future
6. Conclusion

SYCL: 101

- SYCL is the specification than people are implementing
- The specification is shepherded by Khronos: the same people who do Vulkan, SPIR-V, and OpenCL
- SYCL borrows some concepts from OpenCL but has no ”requirement” on OpenCL whatsoever.
- Lots of institutions are part of the SYCL WG
  - Argonne, Qualcomm, University of Heidelberg, Intel,
  - Intel is a member of the working group like any other member. SYCL is not an Intel thing
- SYCL is 10 years old.
  - It predates accelerators being able to allocate memory and give you a pointer. Back then you had to use ***`image`*** or ***`buffer`***.
  - The recommended way now is USM. [3]

[3] USM == Unified Shared Memory => the device can dereference a pointer. ***`device`***, ***`host`***, and ***`shared / managed`*** allocations are all USM.

SYCL implementations

- Intel DPC++[4] (a.k.a. `icpx -fsycl`, a.k.a. oneAPI)
  - Backends: Level Zero, OpenCL, CUDA, HIP
- AdaptiveCpp
  - Backends: CUDA, HIP/ROCm, OpenMP host, and a generic JIT path (Metal, Vulkan in progress)
- Upstream LLVM
  - Starting to share a code base with OpenMP: `liboffload`
  - Exciting news: it can already compile hello-world-like apps
- ProtoSYCL
  - Used to easily implement extensions, test our Conformance Test-Suite, etc. Pure C++, std-thread.

[4] Qualcomm have announced a "plugin" for their SoC based on DPC++

SYCL ecosystem

- The SYCL WG contracted Kitware to make SYCL a first-class citizen in CMake.
  - *https://gitlab.kitware.com/cmake/cmake/-/merge_requests/12467*
  - Guarded behind an experimental flag in CMake $N + 1$, official in $N + 2$.
- SYCL Academy: teaching material.
- Public spec work: *https://github.com/KhronosGroup/SYCL-Docs*
- You can join the WG and attend the weekly call[5]
- We also have an advisory panel, free to join
- There is also the UXL Foundation, for SYCL ecosystem / libraries

[5] Not that expensive for academics: and you get to push for the features you want, or against the ones you don't.

Outline

First example: Triad, live demo

- Until vendors drop FP64 support, all* our codes are memory bound.
- Let me show you real code[6]

[6] And run it if the god of HPC blessed us with a node

Live Demo

Kokkos and SYCL are... the same?

- Not that many differences.
- View versus Raw pointer.
- Explicit queue, versus implicit execution space

I love benchmarks. Let me show you more benchmarks

- One where SYCL is 5x faster.
- One where Kokkos is 6x faster.

We will revisit those benchmarks later[7]

[7] Numbers never lie, benchmarks never lie! But one needs to be careful about what we measure and if comparisons are "fair"...

Second live demo[8]

[8] if the first one worked, if not I gave up already

Outline

A nice table (hopefully not too wrong)

Concept	Kokkos	SYCL
Bootstrap	`initialize` / `finalize`	nothing
Device	execution space	`sycl::device`
Stream	exec space instance	`sycl::queue`
Flat parallel loop	`parallel_for(N, f)`	`parallel_for(range<1>, f)`
Multi-dim loop	`MDRangePolicy<Rank<n>>`	`range<1|2|3>`
	team	work-group
	thread	work-item
Vector lane	vector level	sub-group
Team barrier	`team.team_barrier()`	`group_barrier(g)`
Global sync	`fence()`	`queue::wait()`
lambda	`KOKKOS_LAMBDA`	`[=]`

More about View

Kokkos

```
Kokkos::View<float**, LayoutLeft> a("a", M, N);
// pointer + extents + layout + space
//         + refcount + label
a(i,j) = 1.0f;   // layout-aware
```

SYCL

```
// Alocate Memory
float* a = sycl::malloc_device<float>(M*N, q);
// Computing the layout you want by hand
a[i*N + j] = 1.0f;
// Free the memory
// I know it feel weird for F77 people
sycl::free(a, q);
```

- SYCL have no build layout abstraction
- SYCL ***range*** stops at 3 dimensions; ***MDRangePolicy*** goes to 6.
- SYCL have no refcount, mirrors, ***deep_copy***, etc. You do the data-management by hand
- SYCL We use pointers...
- But, as an alternative, mdspan works in device code!

Live Demo mdspam in sycl

Queues are execution space instances

- Queues are by default ”out-of-order”, but can be made in-order
- Queues are bound to a device

AFAIK Kokkos provides the abstraction as ”execution_space”

Note on submission:

- Both async by default; both need an explicit sync to time or read.
- Queues allow for concurrency. Multiple in-order queues (Kokkos uses in-order behind the hood), or out-of-order queues without dependency

SYCL Dependency: Via Event

```
sycl::event e1 = q1.parallel_for([]);
sycl::event e2 = q1.parallel_for([]); //e1 and e2 can execute in parallel
sycl::event e3 = q1.parallel_for([], e2); // this co,mand  will wait for e2 to have finished
↪  to start executing
sycl::event e4 = q2.parallel_for([], e2); // Cross queue dependency
sycl::event e5 = q1.parallel_for([], {e3,e4});  // Can wait on vector of dependency
e5.wait(); // SYNC the dependency chain.
```

Hierarchical parallelism

Kokkos

```
using team_t = Kokkos::TeamPolicy<>::member_type;
Kokkos::parallel_for("t",
  Kokkos::TeamPolicy<>(nteams, tsize),
  KOKKOS_LAMBDA(const team_t& team) {
    int b = team.league_rank();
    Kokkos::parallel_for(
      Kokkos::TeamThreadRange(team, tsize),
      [&](int k){ /* ... */ });
    team.team_barrier();
  });
```

SYCL (new way)

```
q.parallel_for(
  sycl::nd_range<1>{nteams*tsize, tsize},
  [=](sycl::nd_item<1> it) {
    int b = it.get_group(0);
    int k = it.get_local_id(0);
    /* ... */
    sycl::group_barrier(it.get_group());
  });
```

- Kokkos nests a ***parallel_for*** in the team lambda;
- SYCL is now ”flat” with queries, and sync where you want.

KHR for ”free-function”: ‘this_group‘ ‘this_nd_item‘.

Scratch memory, reductions, atomics

”Team local memory”[9]

```
policy.set_scratch_size(0, Kokkos::PerTeam(
  scratch_t::shmem_size(tsize)));
scratch_t s(team.team_scratch(0), tsize);
```

```
q.submit([&](sycl::handler& h) {      // handler REQUIRED
  sycl::local_accessor<float,1> s{sycl::range<1>{tsize}, h};
  h.parallel_for(nd, [=](sycl::nd_item<1> it){ ... });
});
```

Reductions: Kokkos reduces into a host scalar and fences for you; SYCL into a device-accessible pointer and does not sync. Same Group collectives.

Atomic: both converged on the `std::atomic_ref` shape.

[9] `local_accessor` only exists inside a `handler`, so this is the one case forcing the verbose form. We are working on a KHR to improve that

Feel good story time: Collaboration

A gap Kokkos hit, and what we did about it.

- Worked with Aymeric to provide device queries that were missing in SYCL[10]
- Maximum number of work-groups a device can submit
- Now a KHR extension: `sycl_khr_max_work_group_queries`
- And it trickled down to OpenCL too.

```
auto r = dev.get_info<sycl::khr::info::device::max_work_group_range<1>>();
auto n = dev.get_info<sycl::khr::info::device::max_work_group_range_size>();
```

[10] I think Kokkos required it, at least this is what Aymeric told me back in the day.

Now let's go back to the benchmarks and fix them

- At first approximation. If the performance of two programming model differs it's a bug.
- Let's try to find them (just so I know you are still awake...)

The two bugs

- Out of order queue, versus multiple Execution Spaces[11]
- Pointer Required versus Scalar for Reduction API[12]

But still bugs / features everywhere:

- Don't ask me to use N=6 for the partition space please.
- And same for the 20% of difference between the two reductions

We will go out of time :)

[11] Or you can do lots of in-order queues in SYCL if you have bad programming taste...
[12] hence Pinned Memory, Versus Scalar, Versus Shared

Section conclusion: same same, but different

- The execution model matches almost completely: lambdas over index spaces, async queues, teams, scratch, collectives, reductions, atomics.
- The data model differs a little bit: *`View`* has no SYCL equivalent. We are arguably "lower-level". We malloc and move memory

Outline

Where SYCL 2020 is today

Revision 11. Only bug fixes and clarifications go into the main spec now; new features ship as KHR extensions. There are 8:

sycl_khr_default_context	*sycl_khr_queue_flush*
sycl_khr_queue_empty_query	*sycl_khr_work_item_queries*
sycl_khr_group_interface	*sycl_khr_static_addrspace_cast*
sycl_khr_max_work_group_queries	*sycl_khr_dynamic_addrspace_cast*

What is coming

- Submission that does not always create an event. Big ask by the user community.
- Some in the WG want to be more CUDA-like: free functions for work-item queries, CUDA-style submit.
- Deprecation of buffers.
  - Buffers are from the old days.
  - Too verbose, over-engineered.
- In-order queues by default?

Outline

1. Where I am coming from
2. What SYCL is (and is not)
3. Similarities and differences with Kokkos
4. Naming convention
5. SYCL future
6. Conclusion

Conclusion

- Kokkos has a SYCL backend
- SYCL is not an Intel Thinks and doesn't required OpenCL
- If some features are missing, or performance is poor, please report. I consider that as a bug
- Major difference between both, is Kokkos View. Thanksfull C++ mdspan is working in device code in SYCL (thanks again to Kokkos for pushing that in the C++ standard!)
- SYCL plans to deprecate buffers
- Just pick a programming model and enjoy :)

apl@anl.gov

https://github.com/KhronosGroup/SYCL-Docs