Managing Kokkos callbacks for benchmarking, profiling and unit testing

HPSF Conference

Arnst Maarten[1] Tomasetti Romin[1]

University of Liège
[1] Jointly first/last author

May 8th, 2025

Tooling with Kokkos: motivation and outlook

```
template <class ExecPolicy, class FunctorType, ...>
inline void parallel_for(const std::string& str, const ExecPolicy& policy,
                         const FunctorType& functor) {
  // Invoke begin parallel for callback.
  int kpID = 0;
  Kokkos::Tools::Impl::begin_parallel_for(policy, functor, str, kpID);

  // Dispatch.
  auto closure = ...;
  closure.execute();

  // Invoke end parallel for callback.
  Kokkos::Tools::Impl::end_parallel_for(policy, functor, str, kpID);
}
```

Adapted and annotated from Kokkos_Parallel.hpp.

In this talk, we will describe utilities that we designed to manage Kokkos callbacks. Our work is motivated by two use cases:

- benchmarking kernels running in user code
- verification of Kokkos and other API calls in user code

Outline

1. Current state
2. Modernizing `Kokkos` callbacks
3. Use cases
4. Takeaways

Tooling with Kokkos: current state (1/3)

The Kokkos instrumentation has a C-style API with function pointers.

```
typedef void (*Kokkos_Profiling_beginFunction)(const char*, const uint32_t,
                                               uint64_t*);
```

Extracted from Kokkos_Profiling_C_Interface.hpp.

The kokkos-tools repository provides tools that can interface with the instrumentation, such as a kernel logger, a kernel timer, ...
Typically, a tool is dynamically loaded at run time.

```
./my_application --kokkos-tools-libs=/path/to/kokkos-tool.so
```

Vendor profiling tools, such as NVIDIA Nsight Compute and AMD rocprofiler, can interface with the instrumentation to extract region markers, kernel names, memory operation information, ...

Tooling with Kokkos: current state (2/3)

Callbacks can also be set programatically.

```
Kokkos::Tools::Experimental::set_begin_parallel_for_callback(
    [](char const *label, uint32_t, uint64_t *) {
      std::regex re("^(ArborX::|Kokkos::).*");
      BOOST_TEST(std::regex_match(label, re),
                 "\"" << label << "\" does not match the regular expression");
    });
```

Adapted from `ArborX/test/tstKokkosToolsAnnotations.cpp`.

Observations

- The capture list is empty. It works for *non-capturing* lambdas because they can be converted to function pointers.
- `Kokkos` callbacks does not have a `void*` user data pointer argument. Stateful callbacks require some form of global state.

Tooling with Kokkos: current state (3/3)

Associate callback calls with events.

```
struct BeginParallelForEvent : public BeginOperation<BeginParallelForEvent> {
    BeginParallelForEvent(std::string n, const uint32_t devID, uint64_t k)
      : BeginOperation<BeginParallelForEvent>(n, devID, k) {}
};
```

Adapted from `ToolTestingUtilities.hpp`.

Validate API calls. Listen to associated callbacks.

```
ASSERT_TRUE(validate_event_set(
  [&]() { graph.reset(); }, [&](DeallocateDataEvent e) {
    if (e.name == alloc_label && e.ptr == ptr && e.size == ptr_size)
      return MatchDiagnostic{true};
    return MatchDiagnostic{false, {"Name or pointer or size doesn't match"}};
  }));
```

Adapted from `TestGraph.cpp`.

Observations

- Part of Kokkos tests. Users should not use it in their code base.
- Opportunities to use new language features to simplify the design.
- Overlap with GoogleMock container matchers.

Modernizing Kokkos callbacks

Utilities for managing Kokkos callbacks

Modernizing Kokkos callbacks: events

Simplify events by making them aggregates.

```
struct BeginParallelForEvent
{
    std::string name {};
    uint32_t dev_id = 0;
    uint64_t event_id = 0;
};
```

Use concepts to constrain events of similar nature.

```
template <typename EventType>
concept ProfileSectionManipulationEvent =
    Event<EventType> &&
    std::constructible_from<EventType, uint32_t> &&
    requires (EventType event) {
        { event.section_id } -> std::same_as<uint32_t&>;
    };
```

Goals

- Composable filtering of events based on type and attributes.
- Use template metaprogramming to manage callbacks.

Modernizing Kokkos callbacks: listeners

C++ flavor. A listener is a function object that:

- has one or more `operator()(const EventType&)` methods
- can be stateful

```
template <typename MatcherType, Event... EventTypes>
requires Matcher<MatcherType, Kokkos::Impl::type_list<EventTypes...>>
struct RecorderListener
{
    using event_type_list_t = Kokkos::Impl::type_list<EventTypes...>;

    template <EventOneOf<event_type_list_t> EventType>
    void operator()(const EventType& event) {
        if (matcher(event)) this->recorded_events.push_back(event);
    }

    Matcher matcher {};
    std::deque<std::variant<EventTypes...>> recorded_events {};
};
```

Recorder listener.

Modernizing Kokkos callbacks: manager

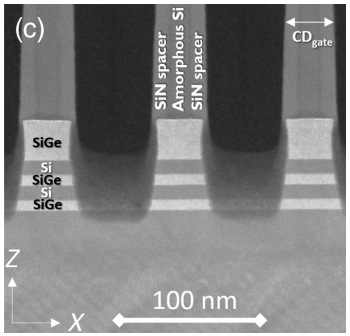
Bridge C++ listeners and C-style callbacks. The manager centralizes:

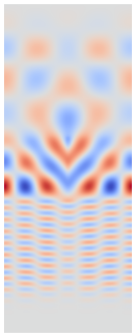
- setting the C-style Kokkos callbacks for the needed event types
- dispatching callback calls as events to registered listeners
- interoperability with a dynamically loaded tool

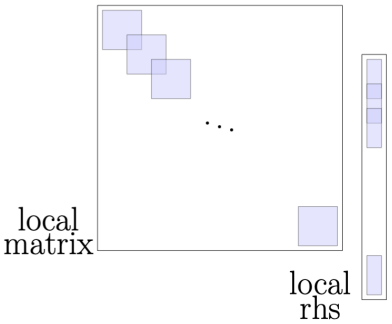
```
using recorder_t = RecorderListener<
    EventInProfileSectionMatcher<EventRegexMatcher>,
    BeginDeepCopy, EndDeepCopy
>;

const auto recorder = std::make_shared<recorder_t>(
    EventInProfileSectionMatcher{
        EventRegexMatcher{std::regex("Tpetra::Vector::update")}
});

Manager::register_listener(recorder);

my_workload.execute(exec);

Manager::unregister_listener(recorder.get());
```

Callback manager.

Benchmarking kernels (1/3)



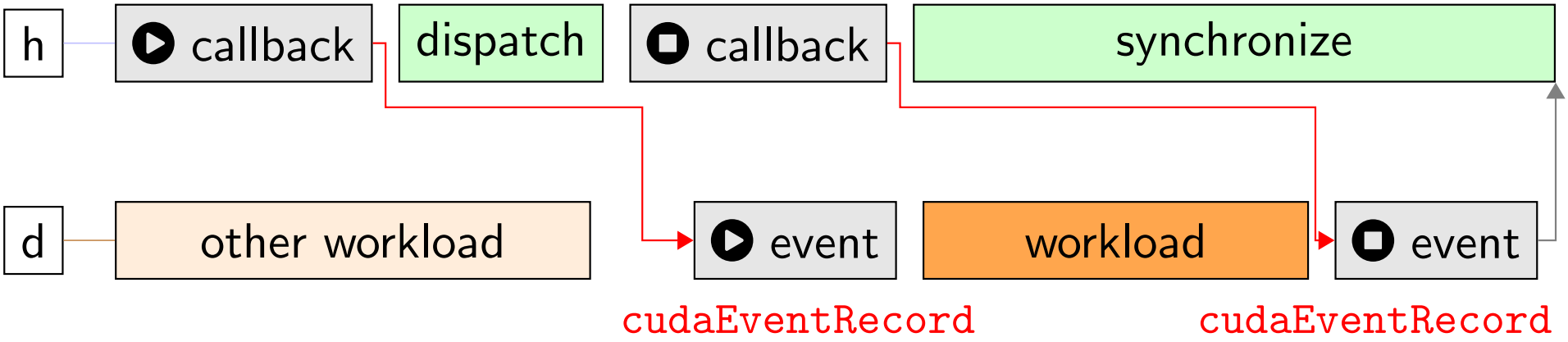




Benchmark kernel(s) running in user code. Nonintrusive approach.

```
void create_static_crs_graph(const Exec& exec)
{
    // other kernel dispatches

    Kokkos::parallel_for(
        "fill set representation",
        Kokkos::TeamPolicy(exec, num_rlvnt_elems, Kokkos::AUTO()),
        FillSetRepresentation{}
    );

    // other kernel dispatches
}
```

Benchmarking kernels (2/3)



Use `TimerListener` to record a `cudaEvent_t/hipEvent_t` into the stream when receiving a `BeginParallelForEvent` with a matching name and a `EndParallelForEvent` with a matching `Id`.

Using `cuda/hip` events:

- avoids the need for tool fences (that are everywhere)
- influences less the execution flow (no fence)
- reports more consistent timing w.r.t. a host timer

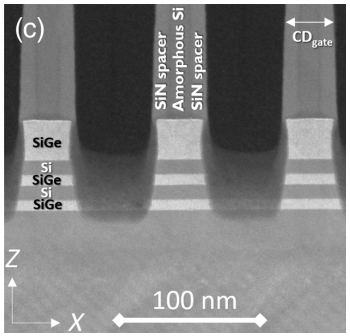
Benchmarking kernels (3/3)

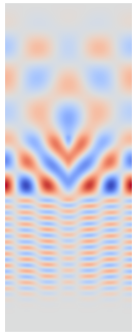
Using `TimerListener` with `Google Benchmark`.

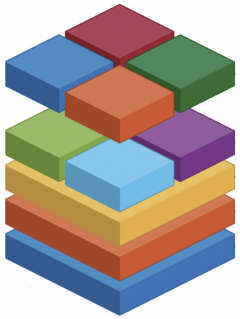
```
struct FillSetRepresentationBenchmark : public ::benchmark::Fixture {
    using par_for_timer_t = ParallelForTimerListener<EventRegexMatcher, Exec>;

    void SetUp(::benchmark::State& /* state */) {
        this->par_for_timer = std::make_shared<par_for_timer_t>(
            exec, EventRegexMatcher(std::regex("fill set representation")));
        Manager::register_listener(this->par_for_timer);
    }

    Exec exec {};
    std::shared_ptr<par_for_timer_t> par_for_timer;
};

BENCHMARK_F(FillSetRepresentationBenchmark, Test)(benchmark::State& state) {
    for (auto _ : state) {
        MyWorkload<Exec>{}.execute(exec);
        state.SetIterationTime(this->par_for_timer->template duration<seconds>().count());}
}
```

```
-------------------------------------------------------------------------------------------------------------------
Benchmark                                                                  Time             CPU   Iterations
-------------------------------------------------------------------------------------------------------------------
CreateStaticCrsGraphBenchmark_UnorderedMap/apply/100/100/3/min_time:5.000/manual_time  117831978 ns    119093552 ns           52
CreateStaticCrsGraphBenchmark_PartiallyOrderedSet/apply/100/100/3/min_time:5.000/manual_time   28655536 ns     46092562 ns          250
```

Verification of API calls (1/2)








```
static void axpby(const ExecSpace& space, const AV& alpha, const XV& X, const BV& beta, const YV& Y)
{
    Kokkos::Profiling::pushRegion("KokkosBlas::axpby[TPL_CUBLAS,complex<double>]");
    ...
    KokkosBlas::Impl::CudaBlasSingleton& s = KokkosBlas::Impl::CudaBlasSingleton::singleton();
    SAFE_CALL(cublasSetStream(s.handle, space.cuda_stream()));
    SAFE_CALL(cublasZaxpy(s.handle, N, reinterpret_cast<const cuDoubleComplex*>(&alpha),
                          reinterpret_cast<const cuDoubleComplex*>(X.data()), one,
                          reinterpret_cast<cuDoubleComplex*>(Y.data()), one));
    SAFE_CALL(cublasSetStream(s.handle, NULL));
    ...
    Kokkos::Profiling::popRegion();
}
```

Adapted from `KokkosBlas1_axpby_tpl_spec_decl.hpp`.

- Verify that a linear algebra op. in user code maps to intended TPL.

Verification of API calls (2/2)

Record events and check:

```
using Exec = Kokkos::Cuda;

Exec exec{};

ASSERT_THAT(
    RecorderListener<BeginPushRegionEvent>::record(
        [exec]{ MyWorkload<Exec>{}.execute(exec); }
    ),
    ::testing::Contains(
        APushRegionEventWithName(
            ::testing::StrEq("KokkosBlas::axpby[TPL_CUBLAS,complex<double>]")
        )
    )
);
```

- Use of `GoogleMock` container matcher to validate the event record.

Takeaways

- We described utilities that we designed to manage `Kokkos` callbacks.
- Opportunities to contribute to `Kokkos`?
  - Promoting tool test utilities to core and installing with `Kokkos`.
  - Using new language features to simplify the design of the events.
  - Introducing `GoogleMock` as a dependency. Using container matchers.
  - Contributing a "Callback manager" to `Kokkos` or `Kokkos::Tools`?
- Directions for future work:
  - Our `Manager` uses a singleton pattern to deal with the issue of the global state. Consider alternative design patterns.
  - Issue of thread safety. Optional.

Kokkos-utils

Our public repository with these utilities (and more):
https://github.com/uliegecsm/kokkos-utils

kokkos-utils / include / kokkos-utils /

romintomasetti callbacks: do not use `std::cout` for the defaulted argument

Name	Last commit message
..	
atomics	doc(kokkos): **@ref** for Kokkos::View is unstable so changed it to @c
callbacks	callbacks: do not use `std::cout` for the defaulted argument
concepts	view(slice): take a subview without specifying all trailing Kokkos::ALL
impl	callbacks: manager
timer	Add event time point.
view	view(slice): take a subview without specifying all trailing Kokkos::ALL