Kokkos 4.5 Release Briefing

New Capabilities

2024-12-10

4.5 Release Highlights

- Organizational
- SequentialHostInit
- Feature Highlights
- General Enhancements
- Graphs Enhancements
- Backend updates
- Tuning changes demo
- Build system updates
- Deprecations and other breaking changes
- Bug Fixes

Online Resources:

- https://github.com/kokkos:
  - Primary Kokkos GitHub Organization
- https://github.com/kokkos/kokkos-tutorials/wiki/Kokkos-Lecture-Series:
  - Slides, recording and Q&A for the Full Lectures
- https://kokkos.org/kokkos-core-wiki:
  - Wiki including API reference
- https://kokkosteam.slack.com:
  - Slack workspace for Kokkos.
  - Please join: fastest way to get your questions answered.
  - Can whitelist domains, or invite individual people.

Would like to strengthen community bonds and discoverability

List of Applications and Libraries

- Add your app to https://github.com/kokkos/kokkos/issues/1950
- We are planning to add that to a Kokkos website.
- Helps people discover each other when working on similar things.

GitHub Topics

- Use *kokkos* tag on your repos.
- If you click on the topic you get a list of all projects on github with that topic.

Organizational

Content:

- HPSF and Kokkos Meeting 2025
- Targetting C++20 for Kokkos 5.0
- SequentialHostInit and Views of Views

Kokkos User Group Meeting 2025 @ HPSF Conference

- *When:* May 5th-8th 2025
- *Where:* Chicago
- *What:* 2-days HPSF plenary + 2-days Project meetings
- *KUG-Content Request:* Focused on user experiences
  - How do you leverage Kokkos?
  - What are pain points?
  - Kokkos based libraries of interest for the community

Registration will open in January!

Kokkos 5 is comming Summer 2025

We will require C++20!

Start preparing now:

- Check availability of compilers on your systems
- Test with C++20 enabled: start with a CPU build
- Minimum Compiler requirements will change (more details later)

Nothing wrong for your project to require C++20 now if you feel ready!

SequentialHostInit and Views of Views

Or: What to do if you need Views of "complicated" objects?

To create a normal `View` the objects it views need to be:

- default-constructible in the View associated execution space
- destructible in the View associated execution space

This Means: Default constructor and desctructor:

- must be `KOKKOS_FUNCTION` (or defaulted)
- can't allocate or deallocate data
- can't call Kokkos parallel operations (`parallel_for` etc.)
- can't create or destroy Views!

If your object does any of these things it is "Complicated"!

- Construction and Destruction of objects will happen sequentially on Host!
- Backported to 4.4.1

```
using Complicated = Kokkos::View<T*, SomeMemorySpace>
View<Complicated**, HostSpace> v(view_alloc("v", SequentialHostInit), 2, 3);
// copy assign elements
v(0,0) = Complicated("w00", 4);
v(1,0) = Complicated("w10", 5);
v(0,1) = Complicated("w01", 6);
// v.~View() handles properly elements destruction
```

Requirements for generic "Complicated" types:

- `T` is default constructible and destructible on host
- `SomeMemorySpace` is host accessible (e.g. `SharedSpace` or `HostSpace`)

Since Kokkos 4.4 programs may dead-lock when creating or destroying View's of complicated objects!

Previously legal code leveraging 'WithoutInitializing' with proper element clean-up before destruction continues to be legal!

Feature highlights

Kokkos can now be built with Run-Time Type Information (RTTI) disabled.

- RTTI is required to deremine the type of an object during program execution.
- It is used by:
  - Exception handling
  - `dynamic_cast`
  - `typeid` operator and `std::type_info` class
- It can be disabled at configuration via `-DCMAKE_CXX_FLAGS="-fno-rtti"`
- Typical use case is to reduce the binary size when targeting systems with limited amount of memory

Required when using Relocatable Device Code (RDC) with SYCL

- KOKKOS_RELOCATABLE_FUNCTION function annotation macro expands to
  - extern __host__ __device__ with CUDA or HIP
  - extern SYCL_EXTERNAL with SYCL
- -DKokkos_ENABLE_{CUDA,HIP,SYCL}_RECLOCATABLE_DEVICE_CODE=ON
- Can have non-trivial perfomance implications

```
// foo.cpp
#include <Kokkos_Core.hpp>
KOKKOS_RELOCATABLE_FUNCTION void foo() { Kokkos::printf("foo\n"); }

// bar.cpp
#include <Kokkos_Core.hpp>
KOKKOS_RELOCATABLE_FUNCTION void foo();
void bar() { Kokkos::parallel_for(1, KOKKOS_LAMBDA(int) { foo(); }); }
```

The SYCL backend is out of the namespace `Experimental::`.

- Keep non-deprecated aliases for now but will deprecate in upcoming releases (potentially in 4.6)
- (Hopefully) very limited impact on user code when leveraging portable aliases (`DefaultExecutionSpace`, `SharedSpace`, `SharedHostPinnedSpace`, etc.)
- If you specialized something for SYCL and really have to spell the SYCL class names, you can always do

```
#if KOKKOS_VERSION_GREATER_EQUAL(4, 5, 0)
using MySyclExec = Kokkos::SYCL;
#else
using MySyclExec = Kokkos::Experimental::SYCL;
#endif
```

General Enhancements

Improved `View` initialization/destruction for non-scalar trivial types

- Skips fences and kernel launches to init or delete elements in a view
- Was only applied to scalar types previously
- Extended this to trivial types (trivially default constructible, trivially destructible)

Added getters for MDRangePolicy tile sizes

```
using tile_type = Kokkos::Array<array_index_type, rank>;

tile_type tile_size_recommended() const;
int max_total_tile_size() const;
```

- tile_size_recommended() returns a Kokkos::Array containing default tile sizes
- max_total_tile_size() returns the max valid tile size
- Independent of tile sizes specified during construction of MDRangePolicy

- Found that Kokkos::sort is slower than std::sort on host

```
std::sort(view.data(), view.data() + view.size());

Kokkos::sort(view); // 2x slower
```

- Caused by reference counting and view access in RandomAccessIterator
- Replaced to use raw pointers in RandomAccessIterator if View is strided

Added free functions, `begin()` and `end()`, that take in a `const` or a non-`const` `Kokkos::Array`. They are usable in constant expressions.

```
Kokkos::Array a = {1, 2, 3, 4};
for (auto x : a)
  Kokkos::printf("%d␣", x);
Kokkos::printf("\n");
// program output: 1 2 3 4
```

Functors can now be used as reducers for nested team `parallel_reduce`

```
Kokkos::parallel_for(Kokkos::TeamPolicy<ExecSpace>(...),
KOKKOS_LAMBDA(team_member_type const& team) {
  Kokkos::parallel_reduce(TeamThreadRange(...),
                          functor_as_reducer{}, result);
});
```

- If a reducer is available, the reducer is used for reduction
- Without a reducer, the reduction type is deduced from an optional `join` on the functor
- Otherwise, the reduction type is a `sum`

Updates to Kokkos::Atomics

Added more "reduction" atomic operations, that is, operations that do not "fetch" the old value.

- atomic_mod
- atomic_xor
- atomic_nand
- atomic_lshift
- atomic_rshift

```
template <class T>
KOKKOS_FUNCTION void atomic_[op](T* ptr, std::type_identity_t<T> val);
// Atomically executes { *ptr = op(*ptr, val); }
```

Allow using SequentialHostInit with Kokkos::DualView (7/7)

Fixed compilation error when SequentialHostInit is used with Kokkos::DualView

```
DualViewType dv(Kokkos::view_alloc("view", Kokkos::SequentialHostInit), N);
dv.resize(Kokkos::view_alloc(Kokkos::SequentialHostInit), M);
```

- SequentialHostInit property was considered during construction of the device-side view

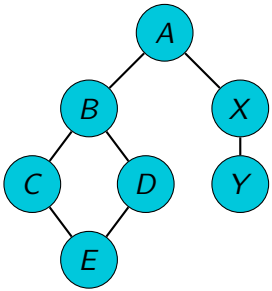
Restrictions

- Not compatible with WithoutInitializing
- Must not have a specifed execution space in the view allocation property, i.e.:

```
view_alloc(exec_space, Kokkos::WithoutInitializing, "ViewString");
```

Graphs Enhancements

`Kokkos::Graph` is an asynchronous execution model that requires all workloads to be defined ahead of execution - as opposed to *eager* execution that you get with a regular execution space instance.



- Makes your code semantics clear and portable.
- Reduces CPU overhead due to scheduling (`Cuda` and `HIP`).
- Enables as many compiler/driver optimizations as possible.

Main PRs for the current release that modified `Kokkos::Graph`

- core(graph): **promote instantiate** to public API
- core(graph): allow submission onto an **arbitrary exec** space instance
- graph(fix): **defaulted graph submit** control flow
- core(graph): allow *create_graph* **without closure**
- graph: allow access to **native graph** object
- graph(diagnostic): enable compile-time diagnostic of **illegal reduction target**
- graph(global-kernel-launch): fix global launch for node kernel

- Using a `Kokkos::Graph` is a 3-phase process
  1. Definition
  2. Instantiation
  3. Submission
- `Kokkos` will instantiate the graph on the first `submit` if needed.
- `Kokkos` will `submit` the graph on the execution space instance provided during *definition* by default.

```
auto graph = Kokkos::Experimental::create_graph<...>(exec,
    [&](const auto& root) {... /* add nodes */ ...});

graph.instantiate(); // optional

graph.submit(/* other_exec */);
```

Then...

```
auto graph = Kokkos::Experimentalcreate_graph<Kokkos::HIP>(
  exec, [&](const auto& root) {
    auto node = root.then_parallel_...;
});
```

Now...

- Kokkos now allows graph *definition* outside of a closure.
- Kokkos allows access to the underlying backend graph (hipGraph_t and so on).

```
auto graph = Kokkos::Experimental::create_graph<Kokkos::HIP>(exec);
// Impl shall disappear at some point.
auto root  = Kokkos::Impl::GraphAccess::create_root_ref(graph);
auto node  = root.then_parallel_...;

size_t num_nodes;
hipGraphGetNodes(graph.native_graph(), nullptr, &num_nodes);
```

- The defaulted graph implementation received some attention (*exec* correctness), but can still be enhanced.
- Kernels nodes with global launch were fixed (`HIP` and `Cuda`).
- Better compile-time diagnostic for illegal reduction target. It must be device-accessible.

Backend Updates

CUDA Add unified memory support for Grace Hopper

- Make `CudaSpace` accessible on the host
- Need to opt-in with `Kokkos_ENABLE_IMPL_CUDA_UNIFIED_MEMORY`
- Introduced in 4.4.1

SYCL Move SYCL out of Experimental namespace

SYCL Add option `Kokkos_ENABLE_SYCL_RELOCATABLE_DEVICE_CODE`

- HIP Add support GFX1103 architecture: "Phoenix" APUs, Ryzen 8000G, Radeon 740M, 760M, 780M, 880M, or 890M (ROCm does not officially support this architecture)
- HIP Update maximum waves per CU used for consumer cards (GFX1YYY)
- HIP Warn if Kokkos runs on a different architecture than the one it was compiled for
- HIP Fix global launch of a kernel graph

HIP Add opt-in option `Kokkos_ENABLE_IMPL_HIP_MALLOC_ASYNC` to use `hipMallocAsync` instead of `hipMalloc`. This will be the default in the next release.

HIP Rework MI300 architecture flags

- For MI300A use `Kokkos_ARCH_AMD_GFX942_APU`
- For MI300X use `Kokkos_ARCH_AMD_GFX942`
- `Kokkos_ENABLE_IMPL_HIP_UNIFIED_MEMORY` has been removed, use `Kokkos_ARCH_AMD_GFX942_APU`

Threads	Fix compilation for `parallel_reduce` using `MDRangePolicy` with `Dynamic` scheduling
Threads	Fix race conditions on ARM architectures (use atomic types instead of `volatile`)
OpenMP	Fix run time behavior when compiling with `-fvisibility-hidden` (backported into 4.4.1)
OpenMP	Fix linking with Cray Clang-based compiler

OpenACC Add support for the Clacc compiler

OpenACC Workaround NVHPC bug when using `MDRangePolicy`

HPX Add `Experimental::partition_space` to create independent execution spaces

Serial `Kokkos_ENABLE_ATOMICS_BYPASS` also skips mutexes to remediate performance regression

Tuning Changes Demo

- Kokkos has runtime auto-tuning support when configured with
  - `Kokkos_ENABLE_TUNING` CMake variable enabled at configuration time
- Kokkos internal auto-tuning support is enabled at runtime when
  - `--kokkos-tune-internals` command line variable enabled at run time
  - a Kokkos tool is used to provide the search
- Internal tunable parameters (available before 4.5)
  - `MDRangePolicy` - X, Y, Z block sizes for CUDA, HIP execution spaces
  - `TeamPolicy` - team size and vector length for CUDA, HIP execution spaces
- Arbitrary runtime parameters can be tuned with the same API, code modifications required - we hope to simplify (or abstract) that API

- New Internal tunable parameters (4.5)
  - `RangePolicy` - Occupancy (with code modification) for CUDA execution space
  - `MDRangePolicy` - Occupancy (with code modification) for CUDA execution space
  - `TeamPolicy` - Occupancy (with code modification) for CUDA execution space
- Occupancy value tuned between [5:100], step size of 5
- Natural extension of PR #3379: "Experimental feature: control cuda occupancy"

"...passing `prefer(policy, DesiredOccupancy(33))` to a `parallel_for()`, `parallel_reduce()`, or `parallel_scan` will bypass the block size deduction that tries to maximize the occupancy and adjust the launch parameters (by fixing the block size and requesting shared memory) to achieve the specified occupancy. The desired occupancy is in percent."

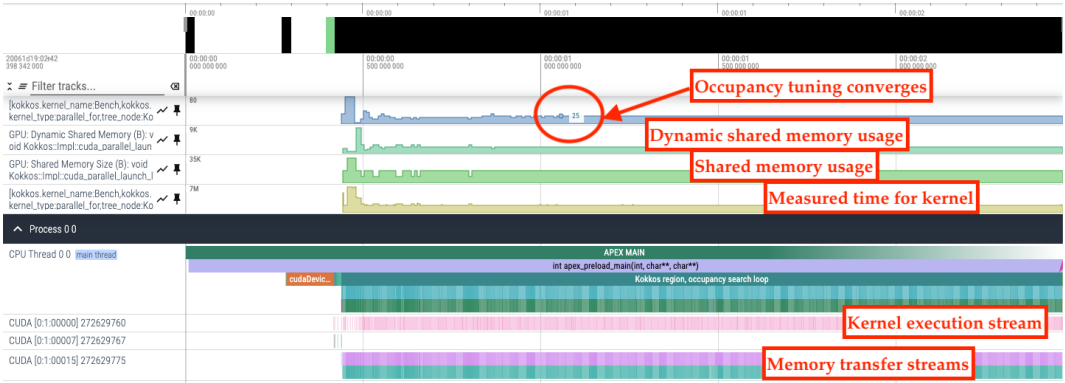
```
using memory_space = typename Kokkos::DefaultExecutionSpace::memory_space;
using view_type = Kokkos::View<double **, memory_space>;
view_type left("process_this", 1000000, 25);
/* Create a policy wrapper to request a tuned occupancy value from 0-100 */
auto const occupancy_policy = Kokkos::Experimental::prefer(
    Kokkos::RangePolicy<>(0, left.extent(0)),
    Kokkos::Experimental::DesiredOccupancy{Kokkos::AUTO});
const auto kernel = KOKKOS_LAMBDA(int i) {
        for (int r = 0; r < 25; r++) {
            double f = 0.;
            for (int m = 0; m < left.extent_int(1); m++) {
                f += left(i, m);
                left(i, m) += f;
            }
        }
    };
Kokkos::parallel_for("Bench", occupancy_policy, kernel);
```

- APEX 2.7.0 released: https://github.com/UO-OACISS/apex
- Included as git submodule in kokkos-tools: https://github.com/kokkos/kokkos-tools/tree/develop/tpls/apex
- Profiling and tracing support for both asynchronous tasking runtimes and "conventional" parallel models (MPI, OpenMP, OpenACC, OpenCL, CUDA, HIP, SYCL, Kokkos, Pthreads, HPX, PaRSEC, StarPU, Iris...)
- Runtime adaptation search strategies (exhaustive, random, simulated annealing, genetic search, nelder mead, auto) integrated with Kokkos tuning API
- Growing set of example cases: https://github.com/khuck/apex-kokkos-tuning
- Long article on Kokkos autotuning with APEX: https://github.com/UO-OACISS/apex/wiki/Kokkos-Runtime-Auto-Tuning-with-APEX

```
# Specify the search strategy
export APEX_KOKKOS_TUNING_POLICY=nelder_mead
# Specify the number of samples to be taken for each configuration
export APEX_KOKKOS_TUNING_WINDOW=4
# Run with apex_exec
apex_exec --apex:kokkos-tuning ./occupancy_example --kokkos-tune-internals

# Possible (truncated) verbose output:
...
Nelder Mead: New best! 0.00197833 k: 1, Bench: 30
Nelder Mead: New best! 0.00196357 k: 7, Bench: 20
Nelder Mead: New best! 0.00196042 k: 10, Bench: 25
Nelder Mead: New best! 0.00195894 k: 48, Bench: 25
...
Converged after 43 iterations.
Total func evaluations: 114
APEX: Tuning has converged for session 1.
[25]
```

Perfetto trace of example tuned by APEX for Nelder Mead search strategy (6/7)



- Simplified/abstracted API for custom tuning in user code (or at least some helper functions)
  - `kokkosp_declare_[input,output]_type` - make it easier to create variables
  - `kokkosp_[begin,end]_context`
  - `kokkosp_request_values`
- Additional examples, performance studies
- Internal tuning for `RangePolicy`
- Internal support/testing for more execution engines (SYCL, OpenMP, OpenMPTarget)

Build Systems Updates

Adding a RISC-V CPU with vectorization

- Support RISC-V CPUs (RVA22V) with the latest vectorization specification.
- CMake option `Kokkos_ARCH_RISCV_RVA22V`

Major refactoring removing 'TriBITS' paths

- Remove full `TriBITS` build option via Trilinos.
- Adding necessary options to the CMake build system.

- Make CUDA related CMake options dependent options of `Kokkos_ENABLE_CUDA`
- CMake option `IMPL_CUDA_MALLOC_ASYNC` default is `OFF` if CUDA backend is not enabled. It should fix issues with some MPI implementations (Cray-MPICH, Bull, ...) or user codes relying on shared memory between processes.

Deprecations and other breaking changes

Some atomic operations deprecated (1/2)

`atomic_query_version()`	No users, and not clear how it would be used
`atomic_assign()`	Use `atomic_store()` instead
`atomic_compare_exchange_strong()`	Use `atomic_compare_exchange()` and compare result
`atomic_{increment, decrement}()`	Use `atomic_{inc, dec}()` instead

`is_asynchronous()`	Only existed in OpenMP and HPX backends and was unused
Tasking interface	Unused, and is a maintenance burden

Bug Fixes

- Fix storage lifetime of driver for global launch of graph nodes for CUDA and HIP
  - Previously, graph nodes were not stored by the graph itself
  - Undefined behavior for all future graph submit when a node goes out of scope
- Fix `TeamPolicy` array reduction for CUDA and HIP

  Previously, array reductions did not compile
- Fix potential out of bounds view access in `Kokkos::BinSort` with `Serial` host backend

- Fix for `deep_copy(exec, ...)` in case where multiple host backends are enabled
  - Previously, passing `Serial` execution space was ignored in favor of a default host parallel backend
- Using CUDA limits to set extents for blocks, grids in `MDRangePolicy`
  - Previous, Kokkos internal tuning was potentially overextend in the `block.z` dimension
- In `RangePolicy` construction, no longer require round-trip convertability between user provided `IndexType` and the internal `member_type`
- Allow extracting host and device views from DualView with const value type

- Set an initial value index during reductions with `MinLoc`, `MaxLoc`, or `MinMaxLoc`
  - Previously, invalid index could be given for location if view contain uniform values
- Make `value_type` for `RandomAccessIterator` non-`const` to match C++ standard
- Fix compilation error when using SYCL without architecture flags
  - `KOKKOS_ARCH_INTEL_GPU` was being assumed
- Allow copy assign between `simd_mask` with `KOKKOS_ARCH_AVX2=ON`

How to Get Your Fixes and Features into Kokkos

- Fork the Kokkos repo (https://github.com/kokkos/kokkos)
- Make topic branch from *develop* for your code
- Add tests for your code
- Create a Pull Request (PR) on the main project *develop*
- Update the documentation (https://github.com/kokkos/kokkos-core-wiki) if your code changes the API
- Get in touch if you have any question (https://kokkosteam.slack.com)