Kokkos 4.7 Release Briefing

08/27/2025

4.7 Release Highlights

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

Find More

Online Resources:

- https://github.com/kokkos:
  - Primary Kokkos GitHub Organization
- https://kokkos.org/kokkos-core-wiki/tutorials-and-examples.html:
  - Tutorials, video lectures, and examples
- 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 the 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

- Targeting C++20 for Kokkos 5.0
- Minimum Requirements for Kokkos 5.0
- Mailing Lists

Kokkos 5 is coming October-November 2025

This 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

Release 4.7 will be the last release before Kokkos 5.0, supported until July 2026

Kokkos 5 minimum requirements

	Kokkos 4.x	Kokkos 5.0
GCC	8.2.0	10.4.0
Clang (CPU)	8.0.0	14.0.0
Clang (CUDA)	10.0.0	16.0.0
IntelLLVM (CPU)	2021.1.1	2022.0.0
IntelLLVM (SYCL)	2023.0.0	2024.2.1
NVCC	11.0	12.2.0
NVC++	22.3	22.3
ROCM	5.2.0	6.2.0
MSVC	19.29	19.30

Note: Clang (CUDA) will require CUDA 11.8 as underlying runtime.

Note: MSVC is only actually tested with the latest version.

https://kokkos.org/kokkos-core-wiki/get-started/requirements.html

Deprecated Code 4 will be off by default and started to be removed!

- In Kokkos 5.0 `Kokkos_ENABLE_DEPRECATED_CODE_4=OFF` is the default!
- Some deprecated feature will immediately be removed.
- Features where we expect users still needing time, will stick around for a couple more minor releases.

Start preparing now with: `Kokkos_ENABLE_DEPRECATED_CODE_4=OFF`

Remember: raw Makefile support will be removed!

Kokkos Mailing Lists

Sign up for the Kokkos mailing list to stay up-to-date with the latest Kokkos news.

`https://kokkos.org/community/mailing-lists/`

Kokkos Trademark

Kokkos is a Linux Foundation project with a trademark!

Why trademark enforcement is important

- **Avoiding confusion**: Users will want to know what is part of Kokkos, with LF rules, versus a project by third parties.
- **Legal Protection**: Preserving the Kokkos project as an Open Source community may require us to protect the project against commercial takeover.
- **User Trust**: Allows us to build up the Kokkos brand as a sign of quality and maturity.

The Kokkos trademark does NOT prevent you from using the word "Kokkos" in your slides, papers and websites!

Trademark Do's and Don'ts

Do's

- We want you to publish and advertise your project using Kokkos!
- We want you to publish on ideas to improve Kokkos!

Don'ts

- Do not imply that your effort is part of the Kokkos project if it isn't.
  - Project Names are critical: Foo for Kokkos (GOOD); Kokkos-Foo (BAD)
  - The Kokkos project uses names such as Kokkos-Kernels, Kokkos-FFT, Kokkos-Comm that imply sub efforts in the Kokkos project.
- Do not imply that the Kokkos project endorsed your effort if it hasn't.

Feature highlights

MDSpan-based View Implementation

- This release contains an extensive refactoring of `Kokkos::View`
- View was refactored to use the `mdspan`, a C++23 addition to the standard library
  - `mdspan` is backported to C++17/C++20, and our implementation can be found at `github.com/kokkos/mdspan/`
- The goal of this refactor was to provide better library interoperability, more API flexibility, and reduced maintenance burden
- In principle, this update should be fully transparent; i.e. your existing code should work as it did before and we've done extensive testing to ensure this

MDSpan-based View Implementation

- What does this mean for applications?
  - We use the same customization points as mdspan now, including accessors and layout mappings
  - This is also how we are working on support for Sacado with the new view implementation
    - We don't have to special-case as much for Sacado inside of Kokkos anymore
    - Full Sacado support coming in patch release
  - In the future, we may provide a mechanism for users to customize these. Would that be useful for people?

Updates on Kokkos Graphs

- Reminder: *Kokkos Graph* is an abstraction of computation represented as a DAG
- Located in *Kokkos::Experimental*
- Example of a diamond-shaped compute graph supported

```
auto graph = Kokkos::create_graph([&](auto root) {
auto nodeA = root.then_parallel_for("workloadA",policy,functor);
auto nodeB = nodeA.then_parallel_for("workloadB",policy,functor);
auto nodeC = nodeA.then_parallel_for("workloadC",policy,functor);
auto nodeD = Kokkos::when_all(nodeB, nodeC).
  then_parallel_for("workloadD",policy,functor);});
graph.instantiate();
graph.submit();
```

- Supports Kokkos patterns (`then_parallel_*`, `then`)
- We have two new features to support more use-cases (subgraphs via stream-capture and host-nodes)

Updates on Kokkos Graphs (Capture)

- Adds support for subgraphs created using stream-capture via `*_capture(...)`
  - Useful to include native code and libraries
  - Calls internally `*StreamBeginCapture`,`*StreamEndCapture`
- Example: Using cuBLAS

```
auto graph = Kokkos::Experimental::create_graph([&](const auto& root){
  auto handle = create_cublas_handle();
   /* NEW! */
  root.cuda_capture(exec,
  [=](const Kokkos::Cuda& exec_){
    /* Body of lambda using CUDA */
    cublasSetStream(handle.get(),exec_.cuda_stream()));
    cublasDgemv( handle.get(),CUBLAS_OP_N ,...);
  });
});
graph.instantiate();
graph.submit(exec);
```

- Supported backends: HIP, CUDA, SYCL (*_capture)

Updates on Kokkos Graphs (then_host)

- Adds support for host-side graph nodes
  - New API: `then_host(...)`
  - Calls internally `*GraphAddHostNode`
- Example: Using a host-side graph node

```
template<>
class functor<Kokkos::DefaultExecutionSpace>{ /* Device code */ };
template<>
class functor<Kokkos::HostSpace> { /* Host code*/ };
auto graph = Kokkos::Experimental::create_graph(exec,
    [&](const auto& root) {
      root.then ("NodeA",exec,functor<Kokkos::DefaultExecutionSpace>{})
      /* NEW! */
      .then_host("NodeB",functor<Kokkos::HostSpace>{});
});
```

- We explicitly use an execution policy with Kokkos::LaunchBounds$\langle 1\rangle$ to execute a *then* graph node

Updates on Kokkos Graphs (then_host)

- Example: Using a host-side graph node to allocate data

```
using view_t = View<int*,Kokkos::MemoryTraits<Unmanaged>>
view_t v;
template<>
class functor<Kokkos::HostSpace> {
  view_t _v;
  void operator()() const { _v = view_t(Kokkos::view_alloc("v",...),10);}
};
template<>
class functor<Kokkos::DefaultExecutionSpace>{ /* Device code */ };
auto graph = Kokkos::Experimental::create_graph(exec,
  [&](const auto& root) {
    /* NEW! */
    root.then_host ("allocate",functor<Kokkos::HostSpace>{v})
    .then_parallel_for("compute",10,
                   functor<Kokkos::DefaultExecutionSpace>{v});
});
...
```

Updates on Kokkos Graphs (other)

- Allow building Kokkos::Experimental::Graph object directly
  - Allows default-constructed Kokkos::Experimental::Graph
  - Graph has a new *root_node()* public member function (returns the graph's root node)
  - Example use:

```
/* NEW! */
Kokkos::Experimental::Graph graph{exec};
graph.root_node().then_parallel_for(1, func{});
graph.submit(exec);
```

Note

- The Kokkos Graph API is `Experimental`
- We would appreciate your feedback!

General Enhancements

Architecture Support and Performance

- Support for AMD Zen 5, SiFive RISC-V Y74MC and ARMv8.4 CPU architectures
  - Enable with `-DKokkos_ARCH_AMD_ZEN5=ON`
  - Enable with `-DKokkos_ARCH_RISCV_U74MC=ON`
- Exit early at initialize with `--kokkos-help`
  - Calling "`executable --kokkos-help`" now causes normal termination
- Symbol visibility fix-ups to support C++20 modules
  - Avoid static variables and functions in header
- Add `constexpr` specifier to `operator==` and `operator!=` for `Kokkos::complex`

- Change the return type of `partition_space` when the weights are known at compile time: `std::vector` → `std::array`
  - Allows to ”unpack” the elements of the expression into individually named variables
  - Breaks backward compatibility

```
/* OLD: still works because of auto */
auto my_vector = Experimental::partition_space(ExecSpace, 1, 1);
auto exec0 = my_vector[0];
auto exec1 = my_vector[1];
...
/* NEW! */
auto [exec0, exec1] = Experimental::partition_space(ExecSpace, 1, 1);
```

- NO changes when the weights are known at runtime

APIs and Behaviors (Continued)

- Add constructors for Random_XorShift*_Pool with execution space argument
  - Allows construction of instances with non-blocking initialization
- Add Kokkos::SIMD::SVE support for 128-bit and 256-bit SVE
  - Adds support for *Scalable Vector Extensions* for Kokkos SIMD types on ARM V8.4-compatible CPUs
  - Enabled with -DKokkos_ARCH_NATIVE, -DKokkos_ARCH_ARM_SVE and -DKokkos_ARCH_ARMV9_GRACE
  - Contributed by Minh Quan Ho from SiPearl
- Implement Kokkos::nextafter for fp16 types

```
Experimental::half_t a = 1, t = 2;
Experimental::half_t b = nextafter(a, t);
```

- Removes [[nodiscard]] attributes from the Kokkos SIMD interface to align with C++26

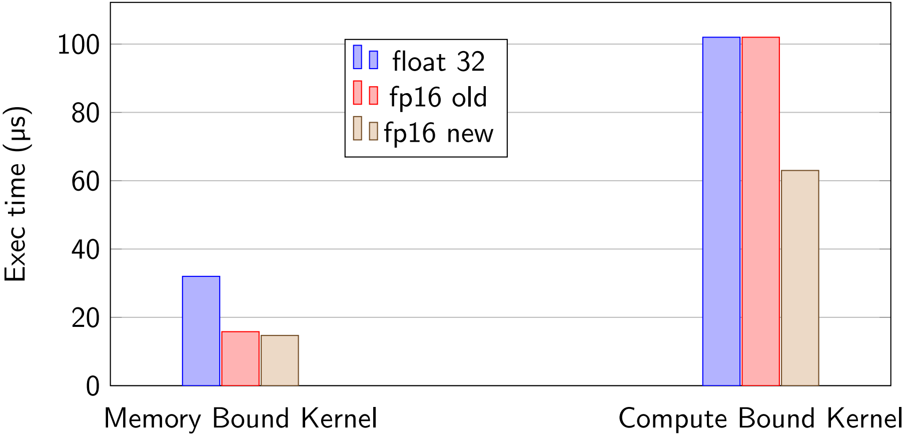
Backend Updates

CUDA and SYCL

- CUDA: Add support for AMPERE87 architecture (Jetson Orin Nano)
- CUDA: Support RDC with Clang 17+ and use new offload driver
- SYCL: Add support for Intel DG2 GPUs such as the Arc Alchemist GPUs
- SYCL: Allow using non-trivially-copyable comparators with oneDPL

Improve half-float performance for CUDA and SYCL backends

- CUDA AND SYCL: Directly use the available fp16 mathematical function instead of casting back and forth to fp32



Other Performance Improvements

- Improving atomic performance for `op_fetch`
  - `atomic_op_fetch` was not specialized as diligently as `atomic_fetch_op` to leverage hardware support or vendor APIs and was falling back to the compare-and-swap implementation
  - `atomic_op_fetch` is now being expressed in terms of ”op applied to the result of `atomic_fetch_op`” which means we get systematically more benefit from the specialization we had written
  - The specialized *atomic_add_fetch* is 10x to 100x faster than CAS on gpus
- Passing label *by reference* in all Kokkos Tools APIs (improving performance)

- Remove support for non-llvm compilers as part of the strategy to only support LLVM compilers in the backend.
- LLVM compilers support extensions to OpenMP directives on GPU that allow *grid* style kernel launches making it more suitable for GPUs and avoiding the overhead of OpenMP's fork-join model.

Build Systems Updates

General Build System Upates

- Require GCC 10.4 for C++20 builds to avoid an ISO C++20 bug
- Error out for `BUILD_SHARED_LIBS` and `RELOCATABLE_DEVICE_CODE`. The vendors don't support it, we just check for it now
- Support more `nvcc` arguments with `nvcc_wrapper`:
  `--ftz`, `--prec-div`, and `--prec-sqrt`
- Add NVIDIA Blackwell architecture support to the makefiles
  Makefiles are officially deprecated

Compiler and linker flag check

We now check the compiler and linker flags at configure time with the given CXX compiler

- Uses CMake's compiler and linker checks
- Uses `CMAKE_CXX_FLAGS` and the flags Kokkos sets
- Not used when `kokkos_launch_compiler` script is used
- If you suspect a false positive please tell us

Deprecations and other breaking changes

SYCL Backend

- The minimum required **IntelLLVM** version has been raised from **2023.0.0** to **2024.2.1**. This change aligns with the Intel HPC Toolkit used for CI testing and resolves critical issues with sorting algorithms.

DualView Debugging

- The option `Kokkos_ENABLE_DEBUG_DUALVIEW_MODIFY_CHECK` has been deprecated and is now **always enabled**. Previously, its default value was dependent on the `Kokkos_ENABLE_DEBUG` option.
- **Rationale:** Enabling this check provides valuable debug information for `DualView::modify[_{device,host}]` calls without a significant performance penalty. It also simplifies the configuration process for users by reducing the number of available build options.

Deprecations

- Deprecate `KOKKOS_MEMORY_ALIGNMENT[_THRESHOLD]` macros
- Deprecate `KOKKOS_NONTEMPORAL_PREFETCH_{LOAD,STORE}` macros
- Deprecate `Kokkos::MemoryManaged` as alias for default memory traits

```
using MemoryManaged [[deprecated]] = Kokkos::MemoryTraits<>;
//                                   ^^^^^^^^^^^^^^^^^^^^^^
//                                   added default template argument
//                                   to avoid spelling out the integer
//                                   value of the empty bitmask
```

Bug Fixes

General Bug Fixes

- Fix a memory leak from an early exit when using `--kokkos-tools-help`
- Add missing fences for async Random init with unified memory
- More robust checks on subview constructor

```
View<T**, LayoutLeft> a(N,N);

// Previously allowed, but data should have strided access.
View<T*, LayoutLeft> sub_a(a, 1, ALL); // Runtime Error
```

General Bug Fixes

- SIMD:
  - Fix compile errors with `Kokkos_ARCH_NATVE=ON`
  - Fix fallback simd masked reductions using incorrect identity elements
- Compilers:
  - Apply a workaround for a segfault issue in `SharedAllocationTracker` with gcc 12.2, 12.3 and 12.4
  - Fix compiling with C++23 supported compilers that provide an mdspan implementation

Backend Bug Fixes

- HPX: fix to constrain hpx_thread_buffer size used with TeamPolicy setup
- HIP and SYCL:
  - A `MDRangePolicy` of rank 4 or more would be incorrectly iterated, leading to some iterations being evaluated more than once for large enough loops
- HIP:
  - `ConstantMemory` launch mechanism would sporadically fail due to `hipEventSynchronize` error
  - Fix launch of intermediate size functors in graph
- Serial: memory leak in internal instance data
- OpenMP Target and OpenACC: An out-of-bounds access would occur in `Random_UniqueIndex` under certain circumstances

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`)