

Kokkos 4.4 Release Briefing

New Capabilities

August 27, 2024

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SANDXXXX PE

4.4 Release Highlights

- ▶ Organizational
- ▶ New Feature: Kokkos::View from std::mdspan
- ▶ Backend updates
- ▶ General Enhancements
- ▶ Build system updates
- ▶ Deprecations and other breaking changes
- ▶ Bug Fixes
- ▶ View Of Views

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 channel 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:

- ▶ Linux Foundation
- ▶ develop Branch
- ▶ Targetting C++20 for Kokkos 5.0

Kokkos is now a member of the



HPSF supports the community development of key HPC projects

Member Organizations

- ▶ Labs: LLNL, SNL, ORNL, LANL, ANL, CEA
- ▶ Industry: HPE, AWS, NVIDIA, Intel, AMD, Kitware
- ▶ Academia: U-Oregon, U-Maryland, CDAC

Member Projects

- ▶ Spack, Kokkos, Trilinos, AMReX, WarpX, Viskores, HPCToolkit, E4S, Charliecloud, Apptainer

What will HPSF do?

- ▶ Provide framework for collaboration on common community concerns (working groups for CI, software security, training etc.)
- ▶ Help organize user-group meetings and trainings
- ▶ Pay for some community infrastructure (e.g. webpage, CI, Slack?)

Getting Involved:

- ▶ Talk to us - Christian, Damien and Julien are all active in HPSF
- ▶ We are still in bring-up phase: working groups are not yet constituted

develop is now the default branch of Kokkos Core and Kernels

- ▶ Switched with 4.3 release
- ▶ master branch is deprecated - still updated to 4.4 right now

Started to publish signed release artifacts

- ▶ Release page: <https://github.com/kokkos/kokkos/releases/latest>
- ▶ **Source distributions:** source archives uploaded by the Kokkos team
- ▶ **Summary files:** Hashes and keys to verify integrity of hash
- ▶ **Assets:** All the files. Note: **Source code (zip/tar.gz)** are GitHub autogenerated and do not promise a stable checksum

Commands for 4.4:

```
KOKKOS_RELEASES=https://github.com/kokkos/kokkos/releases
wget ${KOKKOS_RELEASES}/download/4.4.00/kokkos-4.4.00.tar.gz
wget ${KOKKOS_RELEASES}/download/4.4.00/kokkos-4.4.00-SHA-256.txt
wget ${KOKKOS_RELEASES}/download/4.4.00/kokkos-4.4.00-SHA-256.txt.asc
wget https://kokkos.org/downloads/signing-keys/dalg24.asc
gpg --import dalg24.asc
gpg --verify kokkos-4.4.00-SHA-256.txt.asc
grep kokkos-4.4.00.tar.gz kokkos-4.4.00-SHA-256.txt | sha256sum -c
```

Relevant Output:

```
gpg --verify kokkos-4.4.00-SHA-256.txt.asc
...
gpg: Good signature from "Damien Lebrun-Grandie <dalg24@gmail.com>" \[unknown\]
gpg: WARNING: This key is not certified with a trusted signature!
...
grep kokkos-4.4.00.tar.gz kokkos-4.4.00-SHA-256.txt | sha256sum -c
kokkos-4.4.00.tar.gz: OK
```

Kokkos 5 is coming 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!

mdspan Interoperability

- ▶ `std::mdspan` is a **non-owning** multidimensional view of data
- ▶ Has many similarities to `Kokkos::View`, but unlike `View`, does not own memory or reference count.
- ▶ Part of the C++23 standard, and is a major component of new standard C++ features like `std::linalg`
- ▶ `std::mdspan` improves interoperability between code for array-like data

- ▶ We added conversion functions to and from `std::mdspan`

```
explicit(trait::is_managed) View(const NATURAL_MDSPAN_TYPE &mds);
```

```
template<class El, class Ex, class L, class A>  
explicit(/*...*/) View(const mdspan<El, Ex, L, A> &mds);
```

```
template<class El, class Ex, class L, class A>  
constexpr operator mdspan<El, Ex, L, A>();
```

```
template<class A = Kokkos::default_accessor<typename trait::value_type>>  
constexpr auto to_mdspan(const A &other_accessor = OtherAccessorType{});
```

Conversion rules to and from `mdspan` follow the same principles as between different `View` or `mdspan` types respectively.

The **natural mdspan** of a View is the mdspan that is compatible with the view. An mdspan m of type M that is the natural mdspan of a view v of type V :

- ▶ `M::value_type` is `V::value_type`
- ▶ `M::index_type` is `std::size_t`
- ▶ `M::extents_type` is `std::extents<M::index_type, Extents...>` with `Extents...` being the static extents of the view
- ▶ `M::layout_type` is
 - ▶ `std::layout_left_padded<std::dynamic_extent>` if `V::array_layout` is `LayoutLeft`
 - ▶ `std::layout_right_padded<std::dynamic_extent>` if `V::array_layout` is `LayoutRight`
 - ▶ `std::layout_stride` if `V::array_layout` is `LayoutStride`
- ▶ `M::accessor_type` is `std::default_accessor<V::value_type>`

There are two primary use-cases where `mdspan` can be beneficial in the long run:

- ▶ interfaces with non-Kokkos code
- ▶ fully standardized replacement for *unmanaged* View
 - ▶ <https://eel.is/c++draft/views.multidim>

```
Kokkos::View<const double**> A = /* ... */;  
Kokkos::View<const double*> x = /* ... */;  
Kokkos::View<double *> y = /* ... */;  
// interop with C++26 linalg:  
std::linalg::matrix_vector_product(A.to_mdspan(), x.to_mdspan(), y.to_mdspan());
```

Backend Updates

- ▶ Improve compile-times when building with `Kokkos_ENABLE_DEBUG_BOUNDS_CHECK` (approx. 3x faster to compile)
- ▶ Add support for `--disable-warnings` flag into `nvcc_wrapper`
- ▶ Use `team_size_recommended()` as default team size

- ▶ Add unified memory support for MI300:
 - ▶ make HIPSpace accessible on the host
 - ▶ need to opt-in with `Kokkos_IMPL_HIP_UNIFIED_MEMORY`
 - ▶ introduced in 4.3.1
- ▶ Add options for user to control GPU compilation flags:
 - ▶ `Kokkos_IMPL_AMDGPU_FLAG` and `Kokkos_IMPL_AMDGPU_LINK`
 - ▶ we only set RDC flag
 - ▶ user still needs to set `Kokkos_ARCH_GFX` but we do not set the architecture flag
- ▶ Rework atomics to use builtins

- ▶ Add support for Graphs
- ▶ Fix multi-GPU support
- ▶ Improve performance for top-level `parallel_reduce` and `parallel_scan`, and `team_reduce`
- ▶ Fix lock for guarding scratch space in `TeamPolicy parallel_reduce`

- ▶ OpenACC: Make `TeamPolicy parallel_for` execute on the correct async queue
- ▶ OpenMPTarget: Honor user requested loop ordering in `MDRange` policy
- ▶ OpenMPTarget: Prevent data races by guarding the scratch space used in `parallel_scan`
- ▶ HPX: Fix the compilation of the HPX backend with `nvcc`

General Enhancements

Improve Array facility to align further with `std::array`

▶ Add `to_array()`

```
char a[] = { 'f', 'o', 'o', '\0' };  
auto b = Kokkos::to_array(a); // Kokkos::Array<char, 4>  
  
auto c = Kokkos::to_array({0, 2, 1, 3}); // Kokkos::Array<int, 4>  
auto d = Kokkos::to_array<long>({0, 1, 3}); // Kokkos::Array<long, 3>;
```

▶ Provide `kokkos_swap(Array<T, N>&, Array<T, N>&)` specialization

▶ Make `Array<T, N>` equality comparable

```
Kokkos::Array<int, 2> e = /* ... */;  
Kokkos::Array<int, 2> f = /* ... */;  
  
KOKKOS_ASSERT((e == f) != (e != f));
```

▶ Added CTAD deduction guides for TeamPolicy

```
TeamPolicy()                -> TeamPolicy<>;  
TeamPolicy(int, ...)       -> TeamPolicy<>;  
TeamPolicy(DefaultExecutionSpace, int, ...) -> TeamPolicy<>;  
  
static_assert(!is_same_v<SomeExecutionSpace, DefaultExecutionSpace>);  
TeamPolicy(SomeExecutionSpace, int, ...) -> TeamPolicy<SomeExecutionSpace>;
```

- ▶ Added tuple protocol to complex for structured binding support
 - ▶ Based on structured binding support for `std::complex` added to C++26
 - ▶ Add Tuple Protocol to complex
<https://wg21.link/P2819R2>

```
Kokkos::complex<double> z(11., 13.);  
auto&[r, i] = z;  
Kokkos::kokkos_swap(r, i);  
KOKKOS_ASSERT(r == 13. && i == 11.);
```


- ▶ Harmonize View and (internal) random access iterator convertibility

```
Kokkos::View<int *> x;  
Kokkos::View<const int *> const_y(x); // compiles  
//Kokkos::View<int *> y(const_x); // compiler error  
  
auto x_it = begin(x);  
decltype(begin(const_y)) const_it = x_it; // previously did not compile
```

Add a check precondition non-overlapping ranges for the adjacent_difference algorithm

- ▶ Disallow the overlapping of source and destination iterators (in debug mode). See <https://eel.is/c++draft/numeric.ops#adjacent.difference-8>
- ▶ DO NOT check overlapping if the source and destination iterators are constructed from a single multidimensional view and the strides of these iterators are not identical

```
// Case 0 No longer allowed (Source and destination iterators are the same)
Kokkos::View<double*> a("A",N0);
auto res1 = KE::adjacent_difference("label", exespace(), a, a, args...);

// Case 1 Still allowed (b0/b1 iterates over even/odd numbers only)
Kokkos::View<double[2]*> b("B",N0);
auto sub_b0 = Kokkos::subview(b, 0, Kokkos::ALL);
auto sub_b1 = Kokkos::subview(b, 1, Kokkos::ALL);
auto sub_first_b0 = KE::begin(sub_b0); // 0, 2, 4, ...
auto sub_first_b1 = KE::begin(sub_b1); // 1, 3, 5, ...
auto res2 = KE::adjacent_difference("label", exespace(),
    sub_first_b0, sub_first_b1, args...);
```

Use full vector width for 32 bit data types

- ▶ The vector width of `Kokkos::simd` was determined based on 64 bit data types in available vector registers
- ▶ For 32 bit data types, `Abi` can be specified to use larger vector width

```
{  
  // For AVX512  
  using namespace Kokkos::Experimental;  
  using native_type      = native_simd<float>;  
  using simd_type        = simd<float, simd_abi::avx512_fixed_size<8>>;  
  using simd_larger_type = simd<float, simd_abi::avx512_fixed_size<16>>;  
  
  static_assert(simd_type::size() == native_type::size());  
  static_assert(simd_type::size()*2 == simd_larger_type::size());  
}
```

Applied for: AVX2, AVX512, NEON

- ▶ We use `unlikely` attribute from C++20 to improve reference counting in views on host backends.
- ▶ This only impacts LLVM compilers.

Build Systems Updates

New minimum compiler version requirements for C++20 support

Clang(CPU)	14.0.0
Clang(CUDA)	14.0.0
Clang(OpenMPTarget)	15.0.0
GCC	10.1.0
Intel	not supported
IntelLLVM(CPU)	2022.0.0
IntelLLVM(SYCL)	2023.0.0
NVCC	12.0.0
HIPCC	5.2.0
NVHPC	22.3
MSVC	19.30

- ▶ Add nvidia Grace CPU architecture: `Kokkos_ARCH_ARMV9_GRACE`
When enabled, adds `-mcpu=neoverse-v2 -msve-vector-bits=128` flags
- ▶ Update Intel GPU architectures in Makefile to match CMake
- ▶ Fix incorrect path in Makefile.kokkos when using Threads
- ▶ Fix compilation with CUDA toolkit for CMake 3.28.4 and higher
- ▶ Do not require OpenMP support for languages other than CXX
- ▶ Fix use of OpenMP with Cuda or HIP as compile language
- ▶ Remove support for NVHPC as CUDA device compiler

Potentially Breaking Changes

- ▶ Dropped Array special treatment in View
 - ▶ was treated as an extra compile-time dimension in the view
 - ▶ now able to construct unmanaged view of arrays
- ▶ Got rid of `Experimental::RawMemoryAllocationFailure`
 - ▶ no known usage
 - ▶ internally catching them and rethrowing regular `std::runtime` exceptions
- ▶ Bug fix for thread safety can lead to deadlocks if user code violates Kokkos semantics (see **Bug Fixes - Thread Safety** and **View of Views**)

Deprecations

- ▶ Deprecated allocation step inside `deep_copy(UnorderedMap, UnorderedMap)`
Maps now must have the same capacity to `deep_copy`
- ▶ Deprecated implicit conversions of integers to `ChunkSize`
Behavior only introduced in 4.3
- ▶ Deprecated implicit conversions to all execution spaces

- ▶ Deprecated trailing Proxy template argument in Kokkos::Array

```
// DEPRECATED
// template <typename T = void,
//           size_t N = KOKKOS_INVALID_INDEX,
//           typename Proxy = void>
template <typename T, size_t N>
struct Array { /* ... */ };
```

- ▶ Deprecates *non-owning, dynamically sized* contiguous/strided functionality
- ▶ More in line with (always) *owning & statically sized* std::array

- ▶ Removed `Kokkos::Experimental::LayoutTiled` class template
 - ▶ Never useable
- ▶ Deprecated `is_layouttiled` trait
 - ▶ Not useful, but no rush to remove it
- ▶ Deprecated `Kokkos::layout_iterate_type_selector`
 - ▶ Not useful outside of Kokkos implementation

- ▶ Deprecated specialization of Kokkos::pair for a single element

```
// DEPRECATED  
// template <typename T>  
// struct pair<T, void> { /* ... */ };
```

- ▶ Never supported in std::pair
- ▶ Never documented
- ▶ Never tested
- ▶ No known usage

Bug Fixes

- ▶ Fix using shared libraries and `--fvisibility=hidden`
 - ▶ Used in python wrappers, PETSc, RTLD_DEEPCBIND, ...
 - ▶ problematic with `inline static member variables`

- ▶ Submitting kernels from multiple threads to the same execution space instance allowed
- ▶ They are guaranteed not to run concurrently.
- ▶ Requires locks even in synchronous execution spaces like `Serial` and `OpenMP`.
- ▶ Impact on View of View misuse and kernel in kernel calls.

```
Kokkos::View<int> view("view");
Kokkos::View<int> error("error");
auto lambda = [=]() {
    Kokkos::parallel_for(
        Kokkos::RangePolicy<>(exec, 0, 1), KOKKOS_LAMBDA(int) {
            Kokkos::atomic_store(view.data(), 0);
            for (int i = 0; i < N; ++i) Kokkos::atomic_inc(view.data());
            if (Kokkos::atomic_load(view.data()) != N)
                Kokkos::atomic_store(error.data(), 1);
        });
};
std::thread t1(lambda);
std::thread t2(lambda);
t1.join();
t2.join();
```

- ▶ Return void for `Experimental::for_each`, matching `std::for_each`
- ▶ Support views with non-default constructible values in `realloc`
- ▶ Fix undefined behavior in `View` initialization or fill with zeros
- ▶ Fix compilation of `sort_by_key` when using a host execution space in the CUDA build
- ▶ Fix view reference counting when functor copy constructor throws in parallel dispatch
- ▶ Copy `print_configuration` settings when combining two `Kokkos::InitializationSettings` objects

View of Views

What happens when a view object gets out of scope?

```
{  
  View<T*, HostSpace> v("v", n);  
  // [...]  
} // calls view destructor, i.e. v.~View()
```

What happens when a view object gets out of scope?

```
{
  View<T*, HostSpace> v("v", n);
  // [...]
} // calls view destructor, i.e. v.~View()

// equivalent to:
//   parallel_for(
//     RangePolicy<DefaultHostExecutionSpace>(0, n),
//     KOKKOS_LAMBDA(size_t i) { v(i).~T(); }
//   );
//   kokkos_free(v.data());
```

What happens when a view object gets out of scope?

```
{
  View<T*, HostSpace> v("v", n);
  // [...]
} // calls view destructor, i.e. v.~View()

// equivalent to:
//   parallel_for(
//     RangePolicy<DefaultHostExecutionSpace>(0, n),
//     KOKKOS_LAMBDA(size_t i) { v(i).~T(); }
//   );
//   kokkos_free(v.data());
```

Now, what if T is a view, or some user-defined type that contains a view?

Our programming guide states it clearly (paraphrased):

Please don't.

But, if you do, here is the right way to do it:

Our programming guide states it clearly (paraphrased):

Please don't.

But, if you do, here is the right way to do it:

```
using Naughty = Kokkos::View<T*, SomeMemorySpace>
View<Naughty**, HostSpace> v(view_alloc("v", WithoutInitializing), 2, 3);
// create and initialize elements with a placement new
new &v(0,0) Naughty("w00", 4);
new &v(1,0) Naughty("w10", 5);
new &v(0,1) Naughty("w01", 6);
// [...]
// must **manually** call the elements destructor
v(0,0).~Naughty();
v(1,0).~Naughty();
v(0,1).~Naughty();
```

Lifetime management of element objects is the user's responsibility, and it must be done on the host, **not with a parallel region.**

What happens depends on how the (outer) view was constructed:

- ▶ If you passed the `WithoutInitializing` allocation property, you potentially leak resources
- ▶ Otherwise, your program may hang when you upgrade to 4.4
 - ▶ Outer view destructor launches a parallel region to end the lifetime of individual elements
 - ▶ If an individual element being destructed causes some non-empty (inner) view to go out of scope, Kokkos semantics are being violated
 - ▶ Inner view object being destroyed leads to an attempt to acquire the lock that is already engaged for the outer view cleanup



- ▶ Introduced new SequentialHostInit view allocation property in develop
- ▶ Does not support non-default-constructible element types
- ▶ May backport it to a 4.4.1 patch release if there is strong appetite for it

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

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 questions (<https://kokkosteam.slack.com>)