Kokkos 4.0 Release Briefing

New Capabilities

March 29, 2023

4.0 Release Highlights

- New minimum requirements
- Backend updates
- Build system updates
- Team- and thread-level sort
- Team-level MDRange policies
- SharedSpace
- Miscellaneous
- Deprecations and other breaking changes

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.github.io/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

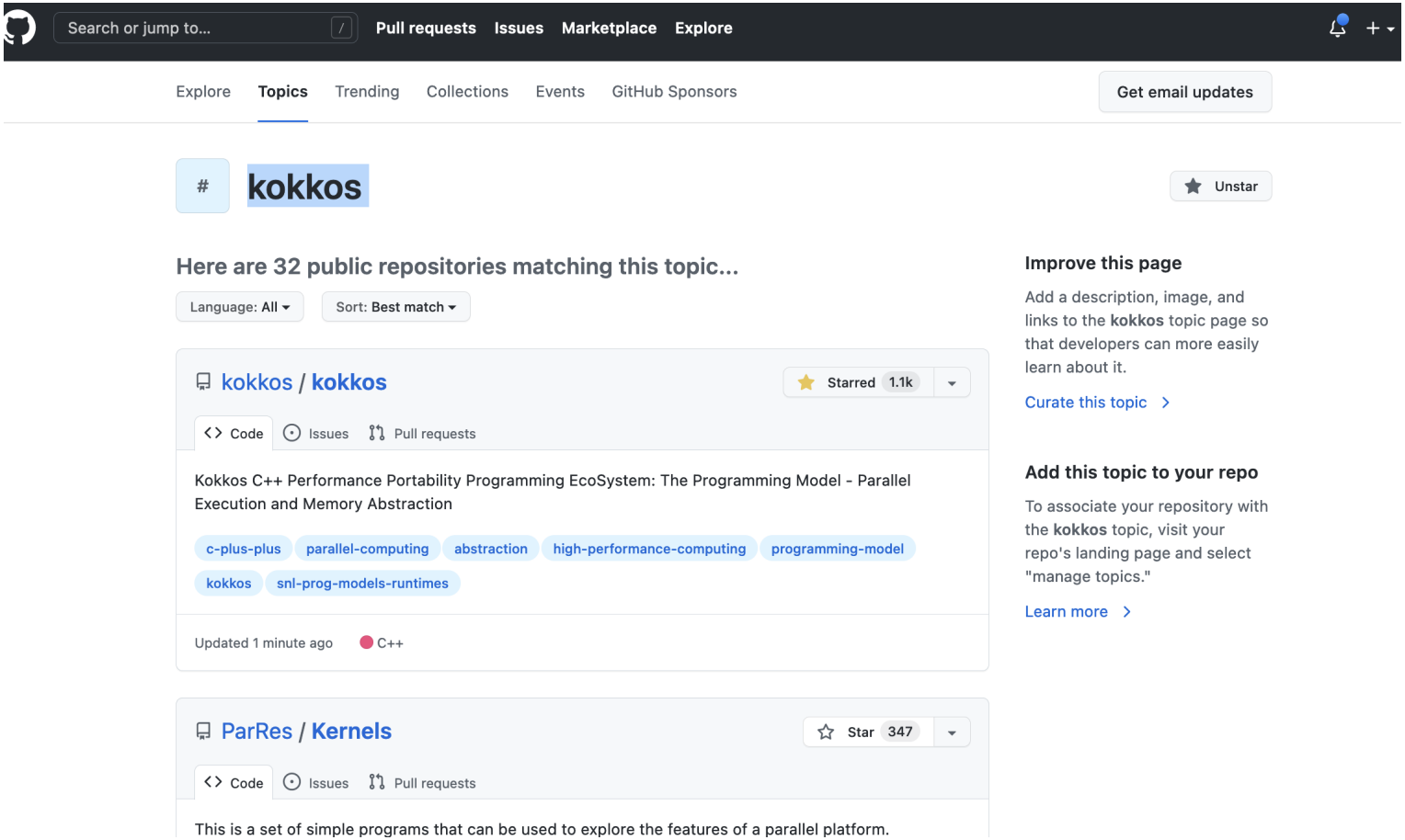
List of Applications and Libraries

- 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.

Kokkos Topic



- We are considering organizing a multi-day in-person user group meeting
- Likely in Albuquerque
- August/September time frame
- Tentative content
  - Updates from the Kokkos team (new features and planned work)
  - User experiences: porting to AMD and Intel GPUs
  - User experiences: performance portability studies
  - Best practices (also user-provided)
  - Students and Postdocs showcase
  - Feedback and discussion session

Lookout for survey to gauge interest

Kokkos 4.0 requires C++17!

- Supporting C++17, C++20 and C++23
- Allows us to keep the testing amount manageable
- Will enable new interfaces and streamlined implementation
  - Use of CTAD reduces the need to spell template arguments out
  - Fold expressions help with internal implementation, and improve compile times
  - `constexpr if` reduces use of clunky SFINAE patterns

New Compiler Minimums

Compiler	Version
GCC	8.2
Clang	8.0
Clang as CUDA compiler	10.0
Intel	19.0.5
CUDA-NVCC	11.0
CUDA with Clang as CUDA compiler	10.0.1
ROCM	5.2.0
IntelLLVM (CPU)	2021.1.1
IntelLLVM (SYCL)	2022.2.0
NVC++	22.3
MSVC	19.29
IBM XL	Not Supported
Classic PGI	Not Supported

- NVIDIA is working on making NVC++ a single-pass CUDA compiler
- Kokkos 4 recognizes NVC++ as a CUDA compiler
- While tested, this is still experimental and several compiler bugs have been reported
- To use NVC++ as the host compiler with NVCC like before, you must use `nvcc_wrapper` and pass `-ccbin nvc++` as cxx and linker flags or set the environment variable `NVCC_WRAPPER_DEFAULT_COMPILER`
- Note as before, NVC++ does not use the GCC in your environment. You must create a configure file with `makelocalrc` pointing to a GCC supporting C++17

Backend Updates

Content:

- SYCL
- OpenMPTarget
- CUDA and HIP

- For `RangePolicy` with `parallel_for`, the workgroup size can be specified manually:

```
parallel_for(
  RangePolicy<ExecutionSpace>(space, 0, N)
    .set_chunk_size(1024), *this);
```

- Intel compiler flags very aggressive, applications might need

```
-fp-model=precise
```

  or similar for correct results.

- The backend now allows selecting the default GPU which can be set by using `--kokkos-device-id=<number>`.
- The backend can now detect the number of devices on a single node.

- Allow CUDA PTX forward compatibility
  - code compiled for compute capability 5.2 will now run on device compute capability 7.5
- Improve CUDA cache config setting
  - let CUDA runtime decide what is the best usage of the cache (shared vs L1 balance)
- Do not rely on synchronization behavior of default stream
  - default instance does not synchronize the other instances
- Add support for Hopper architecture

- HIP, HIPSpace, HIPHostPinnedSpace, and HIPManagedSpace out of Experimental
  - backward compatible change
  - long term support of ROCm 5.2 and later
- Export AMD architecture flags when using Trilinos
  - fix issue when compiling on node without GPU
- Do not rely on synchronization behavior of default stream
- Dropped support for MI25 and added support for Navi1030

- There is a compiler bug ROCm 5.3 and 5.4 when using LocalMemory launch mechanism:
  - sometimes hangs
  - sometimes passes
  - often error out with *Reason: Unknown*
- To fix the issue, force GlobalMemory launch mechanism

```
parallel_for(
  Experimental::require(
    RangePolicy(0, N),
    Experimental::WorkItemProperty::ImplForceGlobalLaunch),
  ...);
```

- We do not apply unconditionally because it reduces performance

- CUDA and HIP fixes:
  - Fix incorrect offset when using parallel_scan for < 4 bytes data types
  - Fix max scratch size calculation for level 0 scratch
- HIP fixes:
  - Fix linking error when using amdclang (OLCFDEV-1167)
  - Fix race condition when using `HSA_XNACK=1`

Default instance synchronization behavior

- CUDA and HIP default instances used to implicitly synchronize with other instances as well as raw CUDA and HIP code
- This was NOT intentional behavior

```
parallel_for(N, f1);
// f2 would be sequenced after f1 in previous releases
parallel_for(RangePolicy<>(exec, 0 , N), f2);
```

- Call `DefaultExecutionSpace().fence()`
- Beware of non-Kokkos code calling CUDA or HIP
  - MPI, BLAS, etc.
  - Previously might have been implicitly synchronized with Kokkos code

C++ Standards Support

- `CMAKE_CXX_STANDARD=23` is supported
  - `KOKKOS_CXX_STANDARD` for the Makefile
- In CMake Kokkos will default to C++17 if no standard is specified

OpenMP and OpenMPTarget

- OpenMP flags are now determined by CMake's FindOpenMP
- Makefile: `libatomic` only linked in OpenMPTarget builds

CUDA

- `Kokkos_ENABLE_CUDA_LAMBDA` set to `ON` by default
- Fixes to RDC flags when using CMake CUDA language
- Fixed CUDA 12 when using `nvcc_wrapper` with CMake
- Fixes to using NVHPC as a compiler when CUDA is not enabled

Team- and Thread-Level (Nested) Sorting

Content: Sorting routines that use nested parallelism

- Callable from within a TeamPolicy kernel
- Sort a View using the calling team or thread
  - View may be in global or scratch memory
- `sort`, `sort_by_key` functions
- Allow custom comparators
- In-place, not stable

More sorting capabilities to come in the next releases.

New functions:

- `#include "Kokkos_NestedSort.hpp"`
- In namespace `Kokkos::Experimental`:
  - `sort_team(teamMem, view[, compare])`
  - `sort_by_key_team(teamMem, keys, values[, compare])`
  - `sort_thread(teamMem, view[, compare])`
  - `sort_by_key_thread(teamMem, keys, values[, compare])`

Arguments to new functions:

- `teamMem`: the `TeamPolicy::member_type` passed to your kernel
- `sort` functions take a single view
  - `view`: a `View` to sort
- `sort_by_key` functions sort key/value pairs according to key:
  - `keys`: a `View` of keys to sort
  - `values`: a `View` to permute along with the keys
- `compare` (optional): a comparator object/predicate
  - If not provided, sort ascending (`operator<`)
  - Otherwise: `compare(a, b)` returns true iff. `a` precedes `b`
    - Defined as: `bool operator()(a, b) const`

Team-Level MDRange Policies

Content: Provide multidimensional support for nested parallelism

Additions to nested polcies:

- MD versions of nested team execution policies
  - Supports multi dimension in nested parallel pattern

- TeamThreadRange
- **TeamThreadMDRange**
- TeamVectorRange
- **TeamVectorMDRange**
- ThreadVectorRange
- **ThreadVectorMDRange**

API for TeamThreadMDRange, TeamVectorMDRange and ThreadVectorMDRange

```
parallel_for(
  TeamVectorMDRange<Rank<2>, TeamHandle>(team_handle, N, M),
  [=](int i, int j) { /* ... */ }
);
```

- Takes in `Rank<N,OuterDir,InnerDir>` that describes its iteration pattern
- Same behavior as regular `MDRangePolicy`
  - `N` is number of dimensions (required to be [2, 8])
  - `Iterate` is an `enum { Default, Left, Right }`
  - `Iterate` is used to choose iterating left-most dimension or right-most dimension
  - Only `OuterDir` is used for `TeamMDRange`

TeamThreadMDRange Example

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(TeamPolicy<>(N,AUTO),
             KOKKOS_LAMBDA (TeamHandle const& team) {
  int leagueRank = team.league_rank();
  auto teamThreadMDRange = TeamThreadMDRange<Rank<4>,
                           TeamHandle>(team, n0, n1, n2, n3);
  parallel_for(teamThreadMDRange,
    [=](int i0, int i1, int i2, int i3) {
      /* ... */
  });
});
```

ThreadVectorMDRange Example

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(TeamPolicy<>(N, AUTO),
             KOKKOS_LAMBDA(TeamHandle const& team) {
  int leagueRank = team.league_rank();
  auto teamThreadRange = TeamThreadRange(team, n0);
  auto threadVectorMDRange = ThreadVectorMDRange<Rank<3>,
                                TeamHandle>(team, n1, n2, n3);
  parallel_for(teamThreadRange, [=](int i0) {
    parallel_for(threadVectorMDRange,
      [=](int i1, int i2, int i3) {
        /* ... */
    });
  });
});
```

TeamVectorMDRange Example

```
using TeamHandle = TeamPolicy<>::member_type;

parallel_for(TeamPolicy<>(N,AUTO),
             KOKKOS_LAMBDA(TeamHandle const& team) {
  int leagueRank = team.league_rank();
  auto teamVectorMDRange = TeamVectorMDRange<Rank<4>,
                           TeamHandle>(team, n0, n1, n2, n3);
  parallel_for(teamVectorMDRange,
    [=](int i0, int i1, int i2, int i3) {
      /* ... */
  });
});
```

Thread and Vector Parallelism:

- Based on iteration direction (`OuterDir`)
- For now, at most 2 dimensions are parallelized
  - Thread parallelism is applied to the slowest dimension
  - Vector parallelism is applied to the fastest dimension

SharedSpace

Content:

- `SharedSpace`
- `SharedPinnedHostSpace`

SharedSpace & SharedHostPinnedSpace

Aliases for *MemorySpaces* that are accessible by every *ExecutionSpace*.

`SharedSpace` is memory that is moved and then accessed locally.

`SharedHostPinnedSpace` is memory that is pinned to the host and accessed via zero-copy access.

Backend	SharedSpace	SharedHostPinnedSpace
CUDA	CudaUVMSpace	CudaHostPinnedSpace
HIP	HIPManagedSpace	HIPHostPinnedSpace
SYCL	SYCLSharedUSMSpace	SYCLHostUSMSpace
host	HostSpace	HostSpace

Miscellaneous

Content:

- `View` value type requirements
- `parallel_scan` with `View` return type
- Numerics update
- Drop volatile support from Atomic Views

Prior to Kokkos 4.0, the value type for a `View` must be default-constructible.
This is not required anymore if

- the `View` is created with `WithoutInitializing`
- the value type is implicit-lifetime (it doesn't require a constructor for the type to be properly initialized), or
- the user initializes the View using placement new in a subsequent kernel
- Kokkos will not call the destructor, it will just deallocate memory

View Value Type Requirements

```
#include <Kokkos_Core.hpp>

struct MyValueType
{
  double value;
  MyValueType(double d) : value(d) {}
};

int main() {
  using namespace Kokkos;
  ScopeGuard guard;

  // View<MyValueType*> view("view", 10); // doesn't compile
  View<MyValueType*> view(
    view_alloc("view", WithoutInitializing), 10);

  parallel_for(10, KOKKOS_LAMBDA(int i) {
    new (&view(i)) MyValueType(1.); // placement new
    view(i) = MyValueType(1.); // simple assignment
    printf("%f\n", view(i).value);
  });
}
```

paralle_scan Interface (also without std::string):

```
template<class ExecPolicy, class FunctorType>
parallel_scan(const std::string&, const ExecPolicy&,
              const FunctorType&);

template<class ExecPolicy, class FunctorType, class ReturnType>
parallel_scan(const std::string&, const ExecPolicy&,
              const FunctorType&, ReturnType&);
```

New: ReturnType can be a View

Reminder: parallel_scan is (potentially) asynchronous, just like parallel_reduce depending on the memory space of the return type if any.

- Promoted math constants to `Kokkos::numbers::` namespace
- Added overloads of `hypot` that take 3 arguments
- Added `fma` fused multiply-add math function
- Support finding `libquadmath` with native compiler support
- Dropped `reciprocal_overflow_threshold` numeric trait
- Moved `reduction_identity` out of `<Kokkos_NumericTraits.hpp>` into a new `<Kokkos_ReductionIdentity.hpp>` header (guarded with `#ifdef KOKKOS_ENABLE_DEPRECATED_CODE_4`)

Drop `volatile` support from Atomic Views

Historically, CUDA used `volatile` because it had a non-standard memory model.
This lead to problems when using custom types with Atomic Views.

```
struct Custom {};
// ...
View<Custom[1], MemoryTraits<Atomic>> v(&a);
v[0] = a;
```

```
core/src/impl/Kokkos_Atomic_View.hpp:70: error:
passing ‘volatile AtomicDataElement<...>::value_type’
{aka ‘volatile Custom’} as ‘this’ argument discards
qualifiers [-fpermissive]
   70 |      *ptr = val;
      |      ~~~~~~~~~~
TestCustom.hpp: note: in call to
‘constexpr Custom& Custom::operator=(const Custom&)’
   |      struct Custom {};
   |             ^~~~~~
```

Previously, one would have to add volatile declarations to their custom types:

```
struct Custom {
    Custom& operator=(const Custom&) = default;
    void operator=(const Custom& src) volatile { /* ... */ }

    // As well as other volatile qualified member functions
};
```

However, internally Kokkos no longer uses the volatile overloads, and CUDA no longer requires combining volatile with atomic.

Kokkos changes to internal `Impl::AtomicDataElement`:

- Dropped `volatile` overloads
- `operator=` uses `atomic_store(..., memory_order_relaxed)`
- `operator value_type()` uses `atomic_load(..., memory_order_relaxed)`

Users

- Drop the (now unused) `volatile` overloads at your convenience

Deprecations and other breaking changes

- Do not rely on `-DKokkos_ENABLE_DEPRECATED_CODE_3=ON` to build your code
- We reserve the right to remove code deprecated in the 3.X series at any time without prior notice
- New `Kokkos_ENABLE_DEPRECATED_CODE_4` configuration that is `ON` by default (for now)
- Will be supported for the remaining of the 4.X series
- As usual, unless you explicitly turn `Kokkos_ENABLE_DEPRECATION_WARNINGS OFF` we will warn you when something is being deprecated

Removing DEPRECATED_CODE_3

- Code that was deprecated during the 3.X release series is now being removed
  - There are just a handful exceptions we will leave in for one or two more minor releases to give more transition time
- Refer to the changelog or the deprecation pages in the wiki
- Best effort to keep the promise

  if your code builds against 3.7 with C++17 and deprecated code off, it will build against 4.0

  Except for ...
  - `KOKKOS_ACTIVE_EXECUTION_MEMORY_SPACE_*` (correcting oversight in release 3.6)
  - `Kokkos_ENABLE_CUDA_LDG_INTRINSIC` configuration option and macro (effectively was not used for a while)

Freshly deprecated (DEPRECATED_CODE_4)

- `ExecutionSpace::concurrency()` becomes a non-static member function
- Some `volatile` support in `Kokkos::pair` and `Kokkos::complex`
- `Kokkos_ENABLE_CUDA_UVM` configuration option