Kokkos: C++ Standard Algorithms

Francesco Rizzi, *NexGen Analytics*

Kokkos User Group Meeting 2023

December 14, 2023

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-XXXX PE

- Cezary Skrzyński
- Jakub Strzebonski
- Antoine Meyer
- Kokkos team for reviewing many PRs

- In a nutshell
- API accepting Kokkos Views
- API accepting Kokkos iterators
- Using custom unary or binary functors
- Short examples
- Quiz: compiles? runs?
- Outlook: optimizations, tweaks, Kokkos "ranges"

Kokkos implementation of a (large, eventually growing) selection of std algorithms accepting Kokkos rank-1 `Views` or iterators.

- Header: `Kokkos_StdAlgorithms.hpp`
- Inside the `Kokkos::Experimental`
- **v3.6**: introduced API accepting execution policy instance
- **v4.2**: extended API for team-level support

- Documentation is available in the Kokkos wiki:
  https://github.com/kokkos/kokkos/wiki

Categories mirror the C++ standard

	Currently Supported in Kokkos
Minimum/maximum ops	`min_element`, `max_element`, `minmax_element`
ModifyingSequence ops	`fill`, `fill_n`, `replace`, `replace_if`, `replace_copy`, `replace_copy_if`, `copy`, `copy_n`, `copy_backward`, `copy_if`, `generate`, `generate_n`, `transform`, `reverse`, `reverse_copy`, `move`, `move_backward`, `swap_ranges`, `unique`, `unique_copy`, `rotate`, `rotate_copy`, `remove`, `remove_if`, `remove_copy`, `remove_copy_if`, `shift_left`, `shift_right`
NonModifyingSequence ops	`find`, `find_if`, `find_if_not`, `for_each`, `for_each_n`, `mismatch`, `equal`, `count_if`, `count`, `all_of`, `any_of`, `none_of`, `adjacent_find`, `lexicographical_compare`, `search`, `search_n`, `find_first_of`, `find_end`
Numeric ops	`adjacent_difference`, `reduce`, `transform_reduce`, `exclusive_scan`, `transform_exclusive_scan`, `inclusive_scan`, `transform_inclusive_scan`
Partitioning ops	`is_partitioned`, `partition_copy`, `partition_point`
Sorting ops	`is_sorted_until`, `is_sorted`

- How many of you already use them?
- How many of you would like to use them, but...?

Brief survey of usage of Kokkos std algorithms			
Currently supported	Do you use it in your code? (if yes, add 1 to the count)	NOT yet supported	Would you use it if Kokkos supported ... (if yes, add 1 to the count)
max	0	random_shuffle	0
max_element	0	shuffle	0
min	0	sample	0
min_element	0	partition	0
minmax	0	stable_partition	0
minmax_element	0	sort	0
clamp	0	partial_sort	0
copy	0	partial_sort_copy	0
copy_if	0	stable_sort	0
copy_n	0	nth_element	0
copy_backward	0	lower_bound	0
move	0	upper_bound	0
move_backward	0	binary_search	0
fill	0	equal_range	0
fill_n	0	merge	0
transform	0	inplace_merge	0
generate	0	includes	0
generate_n	0	set_difference	0
remove	0	set intersection	0

https://docs.google.com/spreadsheets/d/1vOolOyOtOya9M3QySpSRrMCkmZtzspLGxTjn_nGQWZc/edit?usp=sharing

API

API accepting Kokkos `Views`

```
1 template <class ExSpaceT, ...>
2 ret_type algo_name(const ExSpaceT& space, view(s), extra);
3
4 template <class ExSpaceT, ...>
5 ret_type algo_name(const std::string& label, const ExSpaceT& space, view(s), extra);
6
7 template <class TeamHandleT, ...>
8 KOKKOS_FUNCTION
9 ret_type algo_name(const TeamHandleT& teamHandle, view(s), extra);
```

```
1 template <class ExSpaceT, ...>
2 ret_type algo_name(const ExSpaceT& space, view(s), extra);
3
4 template <class ExSpaceT, ...>
5 ret_type algo_name(const std::string& label, const ExSpaceT& space, view(s), extra);
6
7 template <class TeamHandleT, ...>
8 KOKKOS_FUNCTION
9 ret_type algo_name(const TeamHandleT& teamHandle, view(s), extra);
```

- `space`: exec space instance
- `teamHandle`: handle given inside a parallel region when using a TeamPolicy
- `label`: passed to the implementation kernels for debugging
  For overload on line 2, defaults to "Kokkos::algo_name_view_api_default"
- view(s): rank-1, `LayoutLeft`, `LayoutRight`, `LayoutStride`;
  must be accessible from `space` or from the space associated with `teamHandle`
- extra: parameters that are specific to the algorithm

```
1 template <class ExSpaceT, ...>
2 ret_type algo_name(const ExSpaceT& space, iterators, extra);
3
4 template <class ExSpaceT, ...>
5 ret_type algo_name(const std::string& label, const ExSpaceT& space, iterators, extra);
6
7 template <class TeamHandleT, ...>
8 KOKKOS_FUNCTION
9 ret_type algo_name(const TeamHandleT& teamHandle, iterators, extra);
```

```
1 template <class ExSpaceT, ...>
2 ret_type algo_name(const ExSpaceT& space, iterators, extra);
3
4 template <class ExSpaceT, ...>
5 ret_type algo_name(const std::string& label, const ExSpaceT& space, iterators, extra);
6
7 template <class TeamHandleT, ...>
8 KOKKOS_FUNCTION
9 ret_type algo_name(const TeamHandleT& teamHandle, iterators, extra);
```

- space, teamHandle, extra: same as before
- iterators:
  - must be **random access iterators**
  - preferably use Kokkos::Experimental::begin,end,cbegin,cend (coming up)
  - must be accessible from space or from the exec space of teamHandle

`Kokkos::Experimental::{begin, cbegin, end, cend}`

Declaration:

```
template <class DataType, class... Properties>
KOKKOS_INLINE_FUNCTION
auto begin(const Kokkos::View<DataType, Properties...>& view);
```

- `view`: must be rank-1 with `LayoutLeft`, `LayoutRight`, or `LayoutStride`.
- Dereferencing iterators must be done in an execution space where ‘view‘ is accessible.

```
Kokkos::Experimental::distance(first, last);
Kokkos::Experimental::iter_swap(it1, it2);
```

- Kokkos API accepts both random access iterators and Views directly. This is similar to C++ algorithms operating on ranges (C++20).
- The Kokkos algorithms semantically "correspond" to the C++ std algorithms using `std::execution::parallel_unsequenced_policy`
- Implemented in terms of Kokkos `parallel_{for, reduce, scan}`.
- Debug mode enables several checks, e.g.: whether iterators identify a valid range, the execution space accessibility, etc., and error messages printed.
- Currently, algorithms fence directly the execution space instance or call the team barrier for the team handle. This kinds of contradicts the Kokkos semantics and discussions are ongoing to fix this to make them potentially non-blocking.

```cpp
namespace KE = Kokkos::Experimental;

Kokkos::View<double*, Kokkos::HostSpace> myView("myView", 13);
// fill myView somehow

const double oldVal{2}, newVal{34};
auto defHostSpace = Kokkos::DefaultHostExecutionSpace();

// act on the entire view
KE::replace(defHostSpace, myView, oldVal, newVal);

// act on just a subset
auto startAt = KE::begin(myView) + 4;
auto endAt   = KE::begin(myView) + 10;
KE::replace(defHostSpace, startAt, endAt, oldVal, newVal);

// pass label and execution space (assumed enabled)
KE::replace("mylabel", Kokkos::OpenMP(), myView, oldVal, newVal);
```

Example with custom functor for comparison

```cpp
template <class ValueType1, class ValueType2 = ValueType1>
struct CustomLessThanComparator {
  KOKKOS_INLINE_FUNCTION
  bool operator()(const ValueType1& a, const ValueType2& b) const
  {
    // return true if a is less than b, according to your custom logic
  }
};

int main(){
  // ...
  namespace KE = Kokkos::Experimental;
  Kokkos::View<double*, Kokkos::CudaSpace> myView("myView", 13);
  // fill a somehow
  auto res = KE::min_element(Kokkos::Cuda(), myView,
                             CustomLessThanComparator<double>());
  //...
}
```

Team-level example: `replace_if` for each row in a rank-2 View

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 int main(){
21   // ...
22   Kokkos::View<int**> v("v", Nr, Nc); // # rows(Nr), # cols(Nc), filled somehow
23   const int threshold(151), newVal(1);
24   Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace> policy(Nr, Kokkos::AUTO());
25   Kokkos::parallel_for(policy, TestFunctor(v, threshold, newVal));
26   // ...
27 }
```

Team-level example: `replace_if` for each row in a rank-2 View

```
1
2
3
4
5
6
7  template <class ViewType, class ValueType>
8  struct TestFunctor {
9    ViewType m_view; ValueType m_threshold; ValueType m_newVal;
10   // ...
11
12   template <class MemberType>
13   KOKKOS_INLINE_FUNCTION void operator()(const MemberType& member) const {
14     const auto myRowIndex = member.league_rank();
15     auto myRowSubView     = Kokkos::subview(m_view, myRowIndex, Kokkos::ALL());
16     GreaterThanValueFunctor predicate(m_threshold);
17     Kokkos::Experimental::replace_if(member, myRowSubView, predicate, m_newVal);
18   }
19 };
20 int main(){
21   // ...
22   Kokkos::View<int**> v("v", Nr, Nc); // # rows(Nr), # cols(Nc), filled somehow
23   const int threshold(151), newVal(1);
24   Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace> policy(Nr, Kokkos::AUTO());
25   Kokkos::parallel_for(policy, TestFunctor(v, threshold, newVal));
26   // ...
27 }
```

Team-level example: `replace_if` for each row in a rank-2 View

```
1  template <class ValueType>
2  struct GreaterThanValueFunctor {
3    ValueType m_val;
4    KOKKOS_INLINE_FUNCTION GreaterThanValueFunctor(ValueType val) : m_val(val) {}
5    KOKKOS_INLINE_FUNCTION bool operator()(ValueType v) const { return (v > m_val); }
6  };
7  template <class ViewType, class ValueType>
8  struct TestFunctor {
9    ViewType m_view; ValueType m_threshold; ValueType m_newVal;
10   // ...
11
12   template <class MemberType>
13   KOKKOS_INLINE_FUNCTION void operator()(const MemberType& member) const {
14     const auto myRowIndex = member.league_rank();
15     auto myRowSubView     = Kokkos::subview(m_view, myRowIndex, Kokkos::ALL());
16     GreaterThanValueFunctor predicate(m_threshold);
17     Kokkos::Experimental::replace_if(member, myRowSubView, predicate, m_newVal);
18   }
19 };
20 int main(){
21   // ...
22   Kokkos::View<int**> v("v", Nr, Nc); // # rows(Nr), # cols(Nc), filled somehow
23   const int threshold(151), newVal(1);
24   Kokkos::TeamPolicy<Kokkos::DefaultExecutionSpace> policy(Nr, Kokkos::AUTO());
25   Kokkos::parallel_for(policy, TestFunctor(v, threshold, newVal));
26   // ...
27 }
```

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int*> v("v", 10);
6 const bool b = KE::is_sorted(defaultExSpace, v);
```

```cpp
namespace KE = Kokkos::Experimental;

auto defaultExSpace = Kokkos::DefaultExecutionSpace();

Kokkos::View<int*> v("v", 10);
const bool b = KE::is_sorted(defaultExSpace, v);
```

COMPILES?

RUNS?

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int**> v("v", 10, 5);
6 const bool b = KE::is_sorted(defaultExSpace, v);
```

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int**> v("v", 10, 5);
6 const bool b = KE::is_sorted(defaultExSpace, v);
```

COMPILES?

error: static assertion failed: Currently, Kokkos standard algorithms only accept 1D Views

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int*> v("v", 10);
6 const bool b = KE::is_sorted(defaultExSpace, KE::begin(v), KE::cend(v));
```

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int*> v("v", 10);
6 const bool b = KE::is_sorted(defaultExSpace, KE::begin(v), KE::cend(v));
```

COMPILES?

```
error: no matching function
for call to ...
```

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int*> v("v", 10);
6 const bool b = KE::is_sorted(defaultExSpace, KE::cbegin(v), KE::cbegin(v));
```

```cpp
namespace KE = Kokkos::Experimental;

auto defaultExSpace = Kokkos::DefaultExecutionSpace();

Kokkos::View<int*> v("v", 10);
const bool b = KE::is_sorted(defaultExSpace, KE::cbegin(v), KE::cbegin(v));
```

COMPILES?

RUNS?

```cpp
namespace KE = Kokkos::Experimental;

auto defaultExSpace = Kokkos::DefaultExecutionSpace();

Kokkos::View<int*> v("v", 10);
const bool b = KE::is_sorted(defaultExSpace, KE::cend(v), KE::cbegin(v));
```

```
1 namespace KE = Kokkos::Experimental;
2
3 auto defaultExSpace = Kokkos::DefaultExecutionSpace();
4
5 Kokkos::View<int*> v("v", 10);
6 const bool b = KE::is_sorted(defaultExSpace, KE::cend(v), KE::cbegin(v));
```

COMPILES?

RUNS?

```
Kokkos contract violation:
Expected precondition
`last >= first` evaluated
false.
```

- Performance optimizations; keep consistency with C++ standard
- Support more algorithms (provide feedback in the survey document please)
- Kokkos "ranges" and interoperability with algorithms
  - https://github.com/fnrizzi/kokkos-tiny-ranges (fork it, contribute!)

```
Kokkos::View<int*> view("v", 1000);

auto p = view | Kokkos::nonlazy_filter(IsEven()) | Kokkos::reverse() | Kokkos::take(10);

Kokkos::parallel_for(p.size(), MyFunc(p));
```

  - Disclaimer: NOT official Kokkos work (yet), WIP but already works
  - Enabling interoperability with current algorithms API *should* be relatively smooth

- Survey: 
- https:://kokkos.github.io/kokkos-core-wiki/API/algorithms-index.html
- https://github.com/kokkos/kokkos/releases/tag/4.2.00
- https://github.com/fnrizzi/kokkos-tiny-ranges

Thank you! Questions?