Kokkos 4.1 Release Briefing

New Capabilities

July 20, 2023

4.1 Release Highlights

- Backend updates
- Build system updates
- Multiple Reducers for Nested Parallel Reduce
- Bit Manipulation
- UnorderedMap Insertion Operation Types
- Miscellaneous
- Deprecations and other breaking changes
- Kokkos on Compiler Explorer

Find More

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

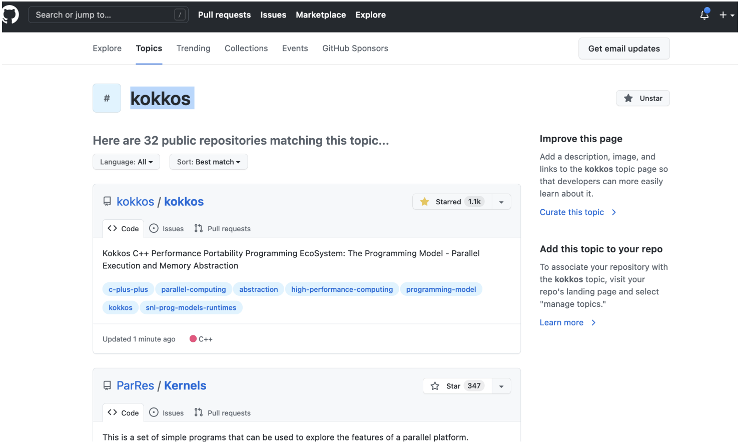
List of Applications and Libraries

- 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



Search or jump to... / Pull requests Issues Marketplace Explore

Explore Topics Trending Collections Events GitHub Sponsors

Get email updates

kokkos

☆ Unstar

Here are 32 public repositories matching this topic...

Language: All ▾ Sort: Best match ▾

kokkos / kokkos ★ Starred 1.1k

<> Code ⊙ Issues Pull requests

Kokkos C++ Performance Portability Programming EcoSystem: The Programming Model - Parallel Execution and Memory Abstraction

c-plus-plus parallel-computing abstraction high-performance-computing programming-model kokkos snl-prog-models-runtimes

Updated 1 minute ago ● C++

ParRes / Kernels ☆ Star 347

<> Code ⊙ Issues Pull requests

This is a set of simple programs that can be used to explore the features of a parallel platform.

Improve this page

Add a description, image, and links to the **kokkos** topic page so that developers can more easily learn about it.

Curate this topic >

Add this topic to your repo

To associate your repository with the **kokkos** topic, visit your repo's landing page and select "manage topics."

Learn more >

Kokkos user group meeting

- We are considering organizing a multi-day in-person user group meeting
- Likely in Albuquerque
- postponed to early December 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

Backend Updates

Content:

- Backend Updates I
- Backend Updates II

Backend Updates I

CUDA

- Allow NVCC 12 to compile using C++20 flag
- Remove ability to disable CMake option Kokkos_ENABLE_CUDA_LAMBDA and unconditionally enable CUDA extended lambda support.
- Drop unnecessary fences around the memory allocation when using CudaUVMSpace in views
  - Issues when relying on View initialization to fence when a execution space instance was passed.

HIP

- Improve performance for parallel_reduce. Use different parameters for LightWeight kernels.

Backend Updates II

`SYCL`

- Improve and simplify `parallel_scan` implementation

`OpenMPTarget`

- Improve hierarchical parallelism for Intel architectures
- Enable Cray compiler for the `OpenMPTarget` backend.

`HPX`

- Update HPX backend to use HPX's sender/receiver functionality
- Increase minimum required HPX version to 1.8.0
- Implement HPX::in_parallel

Build System Updates

- Export CMake Kokkos_CUDA,HIP_ARCHITECTURES variables
- Allow linking against build tree
- Kokkos can be used as an external dependency in Trilinos
- Drop Kokkos_ENABLE_LAUNCH_COMPILER option which had no effect
- Export variables for relevant Kokkos options with CMake

Multiple Reducers for Nested Parallel Reduce

Content: Team-level parallel reduce with multiple reducers

- Extended reducer capabilities in nested `parallel_reduce`
- Allow multiple reductions in a single team `parallel_reduce`
- Supported for `TeamThreadRange`, `ThreadVectorRange` and `TeamVectorRange` policies
  - Not available for `TeamMDRangePolicies` for now

Multiple Reducers Interface

```
template <typename TeamPolicy, typename FunctorType,
          typename... ReducerArgument>
Kokkos::parallel_reduce(const TeamPolicy& policy,
                        const FunctorType& functor,
                        const ReducerArgument&... reducer);

template <typename TeamPolicy, typename FunctorType,
          typename... ReducerArgumentNonConst>
Kokkos::parallel_reduce(const TeamPolicy& policy,
                        const FunctorType& functor,
                        ReducerArgumentNonConst&... reducer);
```

- The number of reducers and the number of functor's reducer value arguments must match.

Multiple Reducers Example

```
Kokkos::parallel_for(
  policy, KOKKOS_LAMBDA(team_member_type const& team) {
    /* ... */

    Kokkos::parallel_reduce(
      teamPolicy,
      [=](int& i, int& arg0, int& arg1, int& arg2, int& arg3) {
        /* ... */
      },
      result0, Kokkos::Prod<int>(result1),
      Kokkos::Max<int>(result2), result3);
  }
);
```

Bit Manipulation

Content:

- `Kokkos::` equivalents for C++20/C++23 components to access, manipulate and process both individual bits and bit sequences
  - `bit_cast`
  - `byteswap`
  - Integral powers of 2
    - `has_single_bit`, `bit_ceil`, `bit_floor`, `bit_width`
  - Rotating
    - `rotl`, `rotr`
  - Counting
    - `countl_zero`, `countl_one`, `countr_zero`, `countr_one`, `popcount`

bit_cast

~~constexpr~~ `To bit_cast<To>(From const& from) noexcept`

- Reinterpret the object representation of one type as that of another
  - `sizeof(From) == sizeof(To)`
  - `From` must be trivially copyable
  - To must be trivially copyable
- Not `constexpr` (differs from C++23 `std::bit_cast`)

```
double d1 = 19880124.0;
auto  u64 = Kokkos::bit_cast<uint64_t>(d1);
auto   d2 = Kokkos::bit_cast<double>(u64);

assert(d1 == d2);
```

`constexpr T byteswap(T value) noexcept`

- Reverses the bytes in the given integer value

- T is an integral type
  - bool, char, char8_t, char16_t, char32_t, wchar_t, short, int, long, long long, clang __128 (but not gcc __128)
    - signed and unsigned integer types

```
int32_t i1 = 0xdeadbeef;
auto    i2 = Kokkos::byteswap(i1);

assert(i2 == 0xefbeadde);
```

Integral Powers of 2

- `constexpr bool has_single_bit(T x) noexcept`
  - Checks if a number is an integral power of 2
- `constexpr T bit_ceil(T x) noexcept`
  - Finds the smallest integral power of two not less than `x`
- `constexpr T bit_floor(T x) noexcept`
  - Finds the largest integral power of 2 not greater than `x`
- `constexpr int bit_width(T x) noexcept`
  - Finds the smallest number of bits needed to represent `x`

- T is an unsigned integer type
  - `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long`, `unsigned long long`

```
uint64_t x = 5;  // 0b101
assert(Kokkos::has_single_bit(x) == false);
assert(Kokkos::bit_ceil(x) == 8);
assert(Kokkos::bit_floor(x) == 4);
assert(Kokkos::bit_width(x) == 3);
```

Rotating

- `constexpr T rotl(T x, int x) noexcept`
  - Computes the result of a bitwise left-rotation
- `constexpr T rotr(T x, int x) noexcept`
  - Computes the result of a bitwise right-rotation

- `T` is an unsigned integer type

```
uint16_t i16 = 0b1001110000111001;
assert(Kokkos::rotl(i16, 2) == 0b0111000011100110);
assert(Kokkos::rotr(i16, 3) == 0b0011001110000111);
```

Counting

- `constexpr int countl_zero(T x) noexcept`
  - Count the number of consecutive 0 bits, starting from the most significant bit
- `constexpr int countl_one(T x) noexcept`
  - Count the number of consecutive 1 bits, starting from the most significant bit
- `constexpr int countr_zero(T x) noexcept`
  - Count the number of consecutive 0 bits, starting from the least significant bit
- `constexpr int countr_one(T x) noexcept`
  - Count the number of consecutive 1 bits, starting from the least significant bit
- `constexpr int popcount(T x) noexcept`
  - Count the number of 1 bits in an unsigned integer

- `T` is an unsigned integer type

Counting

```
uint16_t bits = 0b1111101000110100;

assert(Kokkos::countl_zero(bits) == 0);
assert(Kokkos::countl_one(bits)  == 5);
assert(Kokkos::countr_zero(bits) == 2);
assert(Kokkos::countr_one(bits)  == 0);
assert(Kokkos::popcount(bits)    == 9);
```

- In namespace `Kokkos::Experimental::`
- Not `constexpr`
- Directly call the compiler builtin version, if beneficial

- `bit_cast_builtin`
- `byteswap_builtin`
- Integral powers of 2
  - `has_single_bit_builtin`, `bit_ceil_builtin`, `bit_floor_builtin`, `bit_width_builtin`
- Rotating
  - `rotl_builtin`, `rotr_builtin`
- Counting
  - `countl_zero_builtin`, `countl_one_builtin`, `countr_zero_builtin`, `countr_one_builtin`, `popcount_builtin`

UnorderedMap Insertion Operation Types

Content: Extended UnorderedMap insertion behavior

- Default behavior is to insert a key, value pair exactly once
- Maintain default behavior via operation type `NoOp`
- Allow existing key, value pairs to be accumulated into via operation type `AtomicAdd`

UnorderedMap Insertion Interface

```
template <class ValueTypeView, class ValuesIdxType>
struct UnorderedMapInsertOpTypes {
  using value_type = typename ValueTypeView::non_const_value_type;
  struct NoOp {
    void op(ValueTypeView, ValuesIdxType, const value_type);
  };
  struct AtomicAdd {
    void op(ValueTypeView values, ValuesIdxType values_idx,
            const value_type v);
  };
};


template <typename InsertOpType = default_op_type>
insert_result insert(key_type const &key,
                     impl_value_type const &value,
                     InsertOpType arg_insert_op = InsertOpType());
```

- For other use-cases, more operation types can be added to `UnorderedMapInsertOpTypes`

UnorderedMap AtomicAdd Operation Type Example

```
using map_op_type
  = Kokkos::UnorderedMapInsertOpTypes<value_view_type, size_type>;
using atomic_add_type = typename map_op_type::AtomicAdd;
atomic_add_type atomic_add;
parallel_for(N, KOKKOS_LAMBDA (uint32_t i) {
  map.insert(i, values(i), atomic_add);
});
```

Miscellaneous

- Add Kokkos::Profiling::ScopedRegion

```
double myfunction()
{
 Kokkos::Profiling::ScopedRegion region("foo");
 if (..)
   return bar;
 else
   return eval();
}
```

- Add support for View::rank[_dynamic]()
- Detect incompatible relocatable device code mode to prevent ODR violations
- Add (experimental) support for 32-bit Darwin and PPC
- Add missing half and bhalf specialization of the infinity numeric trait

- Add `is_dual_view` trait and align template parameters with regular view
- Allow templated functors in `parallel_for`, `parallel_reduce`, and `parallel_scan`
- Define `KOKKOS_COMPILER_INTEL_LLVM` and only define at most one `KOKKOS_COMPILER*` macro

Deprecations and Behavior Changes

Content:

- Legacy Atomics Removal
- CUDA_LAMBDA always on
- Fencing and UVM
- Other

Legacy Atomics Fallback removed

DESUL atomics have been the default for a few releases, now removed option to use legacy atomics

Related are a few macro removals:

- `KOKKOS_ENABLE_CUDA_ASM*` - macros only used in legacy atomic implementation
- `KOKKOS_ENABLE_[CUDA/OPENMP/GNU/INTEL]_ATOMICS` - used to define legacy atomics

Deprecations - CUDA_LAMBDA Option

- Removed option to enable/disable CUDA lambdas - they are always on
- LAMBDA support is stable in enough in all supported compilers

Configure behavior depends on deprecation setting:

- `Kokkos_ENABLE_CUDA_LAMBDA` not set: **recommended!**
- `Kokkos_ENABLE_CUDA_LAMBDA=ON`: ok now, no warning - future release unused variable warning
- `Kokkos_ENABLE_CUDA_LAMBDA=OFF`
  - `Kokkos_ENABLE_DEPRECATED_CODE_4` not set: warning about setting lambda options, and that the setting is ignored
  - `Kokkos_ENABLE_DEPRECATED_CODE_4=ON`: warning about setting lambda options, and that the setting is ignored
  - `Kokkos_ENABLE_DEPRECATED_CODE_4=OFF`: configure error

Potential Breaking Changes - Fences

- Continue to work on removing fencing which isn't sematically required by Kokkos
- Now removed fence for `CudaUVMSpace` `View` creation when providing an execution space instance

```
// No change in behavior - implicit fence after init
View<int*, CudaUVMSpace> a("A", N);
a[0] = 3;

// Relaxed fencing in 4.1 to implement desired async behavior
View<int*, CudaUVMSpace> a(view_alloc(Cuda(), "A"), N);
// required fence in 4.1 to prevent race condition below
Cuda().fence();
a[0] = 3;
```

General Rule:

Kokkos operations taking an execution space instance are asynchronous!

Other Changes / Deprecations

Trilinos/TriBITS related:

- Removed TriBITS Kokkos subpackages - so if you get Kokkos from Trilinos, not more kokkos-core/algorithms/containers separation
- Removed associated (unused) `Kokkos[Algorithms,Containers]_config.h` files - we did not find any project including them

Sorting:

- Removed default constructors of `BinSort`, `BinOp1D`, and `BinOp3D` - the default constructed state was invalid and can lead to hard to understand failures.

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)