Kokkos 4.2 Release Briefing

New Capabilities

December 13, 2023

4.2 Release Highlights

- Backend updates
- Build system updates
- SIMD updates
- Half type updates
- Distinct `serial` execution space instances
- `Kokkos::printf`
- Extended `parallel_scan` API
- Team-level Std Algorithms
- `Kokkos::sort` accepts a custom comparison functor
- 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

Kokkos

Star

The Kokkos C++ Performance Portability Ecosystem is a production level solution for writing modern C++ applications in a hardware agnostic way. The Ecosystem consists of multiple libraries addressing the primary concerns for developing and maintaining applications in a portable way. The three main components are the Kokkos Core Programming Model, the Kokkos Kernels Math Libraries and the Kokkos Profiling and Debugging Tools.

5 followers

kokkos

kokkos.github.io

Related Topics

c-plus-plus

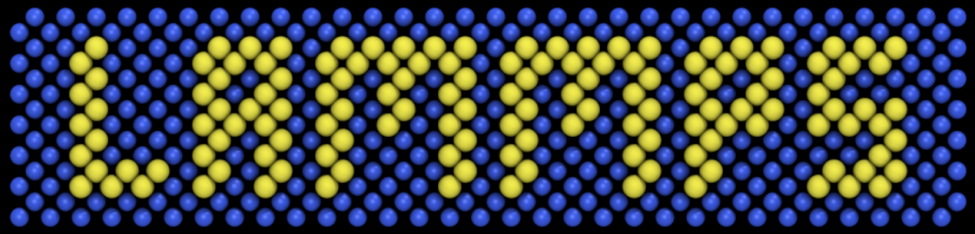
high-performance-computing

parallel-computing

Here are 61 public repositories matching this topic...

Language: All

Sort: Most stars



Large-scale Atomic/Molecular Massively Parallel Simulator

lammps / lammps

Star 1.9k

Backend Updates

Content:

- Backend Updates Cuda
- Backend Updates HIP
- Backend Updates SYCL and OpenACC

CUDA

- Fixed potential data race in `parallel_reduce`
- Use `cudaMallocAsync` by default
- Bugfix when specifying non-default device ID while launching threads after initialization
- Deprecate `Cuda(cudaStream_t,bool)` constructor

HIP

- New naming convention:
  `Kokkos_ARCH_VEGA90A` → `Kokkos_ARCH_AMD_GFX90A`
- Add initial support for gfx942
- Add support for ROCM 5.5 and 5.6
- Improve reduction performance
- Fix potential data race in HIP `parallel_reduce`
- Deprecate `HIP(hipStream_t,bool)` constructor
- Add support for `Kokkos::Graph`
- Fix concurrency calculation

SYCL

- Enforce external `sycl::queues` to be in-order
- Make in-order `sycl::queues` the default via macro
- Improve reduction performance
- Allow using the SYCL execution space on AMD GPUs
- Allow sorting via native `oneDPL` to support `Views` with `stride=1`

OpenACC

- Add support for `clacc` compiler

- Export `Kokkos_CXX_COMPILER_VERSION`
- Disable default `oneDPL` support in `Trilinos`

Kokkos SIMD

Content:

- Math and shift operations
- Generator constructors
- Conditionals: `gather_from` and `scatter_to`
- Miscellaneous

Math Operations

- `simd Kokkos::abs(simd const& a)`
- `simd<double, Abi> Kokkos::floor(simd const& a)`
- `simd<double, Abi> Kokkos::ceil(simd const& a)`
- `simd<double, Abi> Kokkos::round(simd const& a)`
- `simd<double, Abi> Kokkos::trunc(simd const& a)`

`floor`, `ceil`, `round`, `trunc` only operate on floating point simd data types

Shift Operators

- `simd operator>>(const simd& lhs, const simd& rhs)`
- `simd operator>>(const simd& lhs, const int rhs)`
- `simd operator<<(const simd& lhs, const simd& rhs)`
- `simd operator<<(const simd& lhs, const int rhs)`

For AVX2 `Kokkos::simd<std::int64_t>`, shift operators do not use intrinsics

Generator constructors

- template <class G> simd_mask(G&& gen)
- template <class G> simd(G&& gen)

```
template <typename ValueType, typename Abi>
void gen_ctor_test() {
  using simd_type = Kokkos::Experimental::simd<ValueType, Abi>;

  simd_type lhs;
  lhs.copy_from( /*...*/ );

  simd_type rhs(KOKKOS_LAMBDA(std::size_t i) { return /*...*/; });
  mask_type mask(KOKKOS_LAMBDA(std::size_t i) { return /*...*/; })
  simd_type blend(KOKKOS_LAMBDA(std::size_t i) {
    return (mask[i]) ? lhs[i] : rhs[i]; });
}
```

Conditionals

gather_from

- void where_expression::gather_from(const data_type* mem, simd<std::int32_t, Abi> const& index)

scatter_to

- void const_where_expression::scatter_to(data_type* mem, simd<std::int32_t, Abi> const& index) const

```
template <typename ValueType, typename Abi>
void gather() {
  using simd_type = Kokkos::Experimental::simd<ValueType, Abi>;
  using index_type = Kokkos::Experimental::simd<std::int32_t, Abi>
  using mask_type = typename simd_type::mask_type;

  simd_type dst;
  mask_type mask(true);
  ValueType src[] = /*...*/;

  // Indicies to gather from src for each simd lane
  index_type index = /*...*/;

  where(mask, dst).gather_from(src, index);
}
```

- `gather_from` and `scatter_to` are Kokkos functions and are not part of the proposed interface for ISO C++ standard

- `<cmath>` functions in Kokkos SIMD are no longer in `experimental` namespace

copysign	max	tan	asinh	pow
abs	min	asin	acosh	hypot
sqrt	exp2	acos	atanh	atan2
cbrt	log10	atan	erf	
exp	log2	sinh	erfc	
log	sin	cosh	tgamma	
fma	cos	tanh	lgamma	

- Added *float* support to all simd types
- Converted all binary operators to hidden-friends
- `Kokkos_ENABLE_NATIVE` now detects and sets a supported SIMD types
- `Kokkos_ARCH_AVX2` is now on for ZEN2 AMD CPU

`half_t` (since 3.3) and `bhalf_t` (since 3.6) defined in namespace `Kokkos::Experimental::`

- Specialized numeric traits for `half_t` and `bhalf_t`
  - Half-precision types still cannot appear in constant expressions
  - Distinguished values are of an implementation-defined type convertible to half-precision

```
static_assert(
  !std::is_same_v<decltype(infinity_v<half_t>), half_t> &&
  std::is_convertible_v<decltype(infinity_v<half_t>, half_t>
);
```

- Added mathematical functions overloads
  - Currently falling back to `float` and not actually using intrinsics...
- Implemented support for mixed comparisons

```
x < 0.f   // OK
0.f < x   // error before but fine since 4.2
```

Kokkos::printf

- Defined in header <Kokkos_Printf.hpp> which is included from <Kokkos_Core.hpp>
- Prints formated output to the standard output stream
- Calling Kokkos::printf() from a kernel may affect register usage and affect performance.

```
#include <Kokkos_Core.hpp>

int main(int argc, char* argv[]) {
    Kokkos::initialize(argc, argv);
    Kokkos::parallel_for(4, KOKKOS_LAMBDA(int i) {
        Kokkos::printf("hi␣from␣thread␣%d\n", i);
    });
    Kokkos::finalize();
}
```

Serial execution space instances

- Allow creating non-default `Serial` exec space instances
- New constructor taking `NewInstance` tag as argument

```
Kokkos::Serial e1(Kokkos::NewInstance());

auto e2 = Kokkos::Experimental::partition_space(
  Kokkos::DefaultHostExecutionSpace(), 1)[0]; // better
```

- Thread safe since 3.5 but kernels were effectively serialized
- Now enabling overlap of computation on distinct instances

Serial execution space instances

```
#include <Kokkos_Core.hpp>
#include <thread>

template <class Exec> void foo(Exec exec) {
  parallel_for("foo", RangePolicy<Exec>(exec, 0, 3),
    KOKKOS_LAMBDA(int i) { printf("just␣doin␣my␣job␣%d\n", i); });
}

template <class Exec> void bar(Exec exec) { /* ... */ }

int main(int argc, char *argv[]) {
  Kokkos::ScopeGuard kenv(argc, argv);
  using Exec = Kokkos::DefaultHostExecutionSpace;
  auto instances = Kokkos::Experimental::partition_space(
    Exec(), 1, 1);
  std::thread t0(foo<Exec>, instances[0]);
  std::thread t1(bar<Exec>, instances[1]);
  t0.join();
  t1.join();
  return 0;
}
```

Kokkos Std Algorithms: added support for team-level

- Defined in header `<Kokkos_StdAlgorithms.hpp>`
- Extended API with a new overload for team-level support

```
template <class TeamHandleT, ...>
KOKKOS_FUNCTION
/*ret type*/ algo_name(const TeamHandleT& /*teamHandle*/,
                       /*view(s) or iterators*/,
                       /*extra*/);
```

- `teamHandle`: given in parallel region when using TeamPolicy
- `view(s)`: rank-1, `LayoutLeft,Right,Stride`
- `iterators`: must be random access (use `Kokkos::Experimental::begin,end,cbegin,cend`)
- views and iterators must be accessible from `space` or from the space associated with `teamHandle`
- `extra`: parameters that are specific to the algorithm

Kokkos::sort accepts a custom comparison functor

- Defined in header <Kokkos_Sort.hpp>
- Two new overloads to support a custom comparator.

```
template <class ExecSpace, class ViewType, class CompType>
void sort(const ExecSpace& exespace,      // (1)
          const ViewType & view,
          const CompType & comparator);

template <class ViewType, class CompType>
void sort(const ViewType & view,          // (2)
          const CompType & comparator);
```

- view must be rank-1 with LayoutLeft, LayoutRight, or LayoutStride and must be accessible from exespace
- (1) is potentially asynchronous
- (2) calls (1) using the view's execution space, and fences

Kokkos::sort accepts a custom comparison functor

- comparator must be callable from the execution space passed
- comparator must be callable with two arguments a,b of type (possible const-qualified) value_type, where value_type is the non-const value type of the view.
- Snippet:

```
struct MyComp {
KOKKOS_FUNCTION bool operator()(int a, int b) const{
  // return true if a is less than b,
  // according to some, potentially non-trivial logic
}

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

`parallel_scan`: new overload for nested policies with return value

Content:

- API
- Example

parallel_scan: new overload API

- New overload with return value for nested policies

```
template<class ExecPolicy, class FunctorType, class ReturnType>
KOKKOS_FUNCTION
Kokkos::parallel_scan(const ExecPolicy &policy,
                      const FunctorType &functor,
                      ReturnType &return_value);
```

- Valid policies: `ThreadVectorRange`, `TeamThreadRange`
- `return_value` is **overwritten**
- Only valid inside a parallel region executed via `TeamPolicy` or `TaskTeam`.
- `ReturnType` must be compatible with the type of functor

parallel_scan: new overload's representative snippet

```
template<class ViewType, class TeamHandleType>
struct Functor{
  ViewType m_view;

  KOKKOS_FUNCTION void operator()(const TeamHandleType& handle) const
    const auto leagueRank = handle.league_rank();
    // ...
    int accum;
    Kokkos::parallel_scan(
       Kokkos::TeamThreadRange(handle, 0, m_view.extent(1)),
       KOKKOS_LAMBDA(int i, value_type& val, const bool final) {
         val += m_view(leagueRank, i);
         if (final) { // do something }
       }, accum);
  }};

using view_t        = Kokkos::View<int**>;
using policy_t      = Kokkos::TeamPolicy<>;
using team_hande_t = typename policy_t::member_type;
view_t v("v", numRows, numCols);
Kokkos::parallel_for(policy_t(numRows, Kokkos::AUTO),
                     Functor<view_t, team_hande_t>(...));
```

- New headers
  - `<Kokkos_Abort.hpp>` providing `Kokkos::abort` that causes abnormal program termination and is callabled from withing a kernel.
  - `<Kokkos_Assert.hpp>` providing the `KOKKOS_ASSERT` macro that aborts the program if the user-specified condition is not `true` and may be disabled for release builds.
- Add missing `is_*_view` traits and `is_*_view_v` helper variable templates for `DynRankView`, `DynamicView`, `OffsetView`, `ScatterView` containers.

```
static_assert(Kokkos::Experimental::is_scatter_view_v<SV>);
```

Bug fixes

- Missing memory fence in `RandomPool::free_state` functions. Possible race condition sometimes led to repeated random number sequences on different threads with CUDA.
- Fix wrong behavior for corner case in `Kokkos::Experimental::is_partitioned` algorithm.
- Wrong implementation for `cyl_bessel_i0` yielding unexpected NaNs. Special mathematical functions still belong to `Kokkos::Experimental::`.
- Bug in OpenMPTarget with init/join in reducers.
- Resolve symlink creation issue at configuration time on Windows.
- Minor fixes to `Kokkos::Array` (mostly anecdotal).

Bug fixes

- Fix corner cases when deep copying empty views that can lead to undefined behavior.
- Uninitialized variable in `parallel_for(TeamPolicy)` affecting level 1 scratch pad memory in the CUDA backend (introduced in 4.1).
- Fix bug in CUDA PTX for atomic min/max operations of 64-bit integers. Wrong answer when mixing negative and positive values.

Fixes to be included in 4.2.01

- Accidental finalization of tools when non-default host execution space instance is deleted.
- Missing `inline` in new HIP graph implementation leading to multiple definition linking error downstream.
- Add warp synchronization in CUDA `parallel_reduce` to avoid possible race conditions reported by `cuda-memcheck`.
- Workaround bogus deprecated declarations warning for GCC versions less than 11
- Fix MSVC CUDA build (missing header include)

- Ensure that `Kokkos::complex` only gets instantiated for cv-unqualified floating-point types.

```
Kokkos::abs<int>(val);  // used to compile and probably did
                        // not do what you think it does...
```

- Removed (`DEPRECATED_CODE_3`) support for volatile join operators in reductions.
- Raise an error at compile time when passing `view_alloc` with invalid arguments to `create_mirror[_view]` instead of silently ignoring them.

- Purge `#include <iostream>` from Kokkos headers

```
#include <Kokkos_Core.hpp>
// Include what you use

int main(int argc, char *argv[])
{
  Kokkos::intialize(argc, argv);
  // error: 'cout' is not a member of 'std'
  std::cout << "hi␣world\n";
  Kokkos::finalize();
  return 0;
}
```

- Deprecated `Kokkos::vector`
- Use `std::aligned_alloc` for all host allocations
  - Removed code that performed allocations with other mechanisms
  - Deprecated `PosixMemAlign`, `PosixMMap`, `IntelMMAlloc` enumerators from `RawMemoryAllocationFailure::AllocationMechanism` which is defined in `Kokkos::Experimental::`
  - Deprecated the `HostSpace::AllocationMechanism` enumeration and the `HostSpace(AllocationMechanism)` explicit constructor

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)