Kokkos 3.6 Release Briefing

New Capabilities

May 26, 2022

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.
SAND2020-7755 PE

3.6 Release Higlights

- Kokkos Core
  - CMake Language Build Support
  - UniqueToken Improvements
  - More View Allocation Properties Support
  - C++ Standard Algorithms
  - Math Traits
- KokkosKernels

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://github.com/kokkos/kokkos/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.

CMake Build Language

Content:

- Using CMakes Language Extension Support
- CUDA for Windows

What is CMake Language Support

Every source file CMake compiles has a *language*

- Default == file extension (.cpp, .c, .f90, ...)
- But you can set it: `set_source_files_properties(f.bar PROPERTIES LANGUAGE CXX)`

Why does this matter for Kokkos?

CMake treats some language extensions that way, but not all:

- CUDA $->$ CMake language `CUDA`
- HIP $->$ CMake language `HIP`
- OpenMP $->$ **Not** a CMake language.
- SYCL $->$ **Not** a CMake language.

Pre 3.6 behavior

- All Kokkos containing files are C++ files (CXX)
- Kokkos's build system adds compiler flags to make files CUDA, HIP, OpenMP, or SYCL
  - We add `-fopenmp, -x cu, ...`
- We use `nvcc_wrapper` to make CMake not choke on `nvcc`

Advantages:

- Flags can be obtained via depending on target, language `NOT`
- Support for HIP before CMake supported it
- No need for users to set the correct language for each file - *Note:* the *need* propagates, the setting doesn't ...

Drawbacks:

- Not fully using CMakes native CUDA and HIP support
- `nvcc_wrapper` only works on Linux

3.6 behavior

- Default: same as pre 3.6
- Set `-DKokkos_ENABLE_COMPILE_AS_CMAKE_LANGUAGE=ON` to use CMake Language mode.
- Use `set_source_files_properties` on each source file depending on Kokkos
  - We export `Kokkos_COMPILE_LANGUAGE` to make that portable

Pifalls:

- `CMAKE_CXX_FLAGS` unused by Kokkos files
  - `CMAKE_CUDA_FLAGS` for CUDA (equiv. for HIP)
  - `CMAKE_CXX_FLAGS` for SYCL/OpenMP etc.
- YOU need to set `CMAKE_CUDA_ARCHITECTURES` downstream
- YOU need to add `Kokkos_COMPILE_LANGUAGE` to your project!
- For libraries: your users need to set all this too ..
- Interaction with MPI Wrappers iffy ...

CMake Language Example

Configure Kokkos:

```
cmake -DKokkos_ENABLE_CUDA=ON -DKokkos_ARCH_VOLTA70=ON \
  -DKokkos_ENABLE_COMPILE_AS_CMAKE_LANGUAGE=ON ${KOKKOS_SOURCE}
```

Project CMakeLists.txt:

```
#find Kokkos before project declaration
find_package(Kokkos COMPONENTS separable_compilation)

project(Example CXX Fortran ${Kokkos_COMPILE_LANGUAGE})

set_source_files_properties(cmake_example.cpp PROPERTIES
  LANGUAGE \${Kokkos_COMPILER_LANGUAGE})

add_executable(example cmake_example.cpp bar.cpp foo.f)

target_link_libraries(example Kokkos::kokkos)
```

Configure Project:

```
cmake -DCMAKE_CUDA_ARCHITECTURES=70 ${PROJECT_SOURCE}
```

When Should One Use This?

Why should I use this, with all the complication?

- You may want to use native CMake CUDA/HIP support
- You may hate `nvcc_wrapper`
- But most importantly:

This works in Visual Studio for MSVC + NVCC!

Deprecated in release 3.6

Deprecated

Configure with `-DKokkos_ENABLE_DEPRECATED_CODE_3=OFF` to disable

- Array reductions with pointer return types
- `OpenMP::{validate_partition,partition_master}`
- `KOKKOS_ACTIVE_EXECUTION_MEMORY_SPACE_*` macros and `ActiveExecutionMemorySpace` alias
- `log2(unsigned) -> int`

Namespace Change

Not technically deprecation since it was in non-backward guaranteeing state!

`Kokkos::Impl::` − > `Kokkos::`

```
is_array_layout
is_execution_policy
is_execution_space
is_memory_space
is_memory_traits
is_space
is_view
SpaceAccessibility
Timer  // also header impl/Kokkos_Timer.hpp
```

`Kokkos::Experimental::` − > `Kokkos::`

```
Iterate
MDRangePolicy
Rank
```

Removed

Removed:

```
KOKKOS_ACTIVE_EXECUTION_MEMORY_SPACE_HOST/DEVICE
Kokkos::Impl::ActiveExecutionMemorySpace
Kokkos::Impl::verify_space
Kokkos::Experimental::MasterLock
OpenMP/HPX::partition_master
int log2(int) // we got double log2(INTEGRAL) and REAL log2(RE
```

is_space member types removed:

```
is_space::host_memory_space
is_space::host_execution_space
is_space::host_mirror_space
```

Backend Specific Things

CUDA and HIP Error management functionality removed, which should never have been public:

```
CudaSpace::access_error()
CudaUVMSpace::number_of_allocations()
CUDA_SAFE_CALL
HIPSpace::access_error()
HIP_SAFE_CALL
```

Only partially supported by backends:

```
TeamPolicy::vector_length() // only existed for some backends
```

Behavior change in `UnorderedMap`:

```
UnorderdMap::value_at(i) with i>=capacity()
UnorderdMap::key_at(i) with i>=capacity()
```

Interface Changes and Configure

Change in argument order:

```
// deprecated
create_mirror_view(space, view, WithoutInitializing);
// new
create_mirror_view(WithoutInitializing, space, view);
```

Array reductions:

- Array reductions, have a runtime length array as result.
- Only supported with functors, not lambdas (one has to define `value_type` and `value_count` as members)
- Deprecated the option to provide raw pointer as the result argument for `parallel_reduce`, use a `View` instead.

Configure options Removed:

```
Kokkos_ARCH_EPYC -> ZEN/ZEN2 depending on platform
Kokkos_ARCH_RYZEN -> ZEN/ZEN2 depending on platform
Kokkos_ARCH=
Kokkos_DEVICES=
```

Threads is now std::threads

Content:

- Configure Changes

std::thread instead of pthread

We are now using std::thread instead of raw pthread

- No code change necessary - implementation detail of Kokkos
- Makes the `Kokkos::Threads` backend work on Windows
- The backend is more interoperable with other C++ facilities

One change: can't use `Kokkos_ENABLE_PTHREADS` as CMake option.

- `Kokkos_ENABLE_THREADS` is the new option
- We still export `Kokkos_ENABLE_PTHREAD` for downstream users.

Improved `UniqueToken`

Content:

- Configure Changes

What is UniqueToken

`UniqueToken` is a portable way to acquire a unique ID for the calling thread.

- ID is within a given range
- Can be used similar to a *thread-id*
- Most commonly used to acquire a resource from a resource-pool
  - E.g. per-thread temporary memory buffer
  - Used internally for random generator pool

```
UniqueToken<ExecutionSpace> token;
int number_of_uniqe_ids = token.size();
RandomGenPool pool(number_of_unique_ids,seed);
parallel_for("L", N, KOKKOS_LAMBDA(int i) {
  int id = token.acquire();
  RandomGen gen = pool(id);
  ...
  token.release(id);
});
```

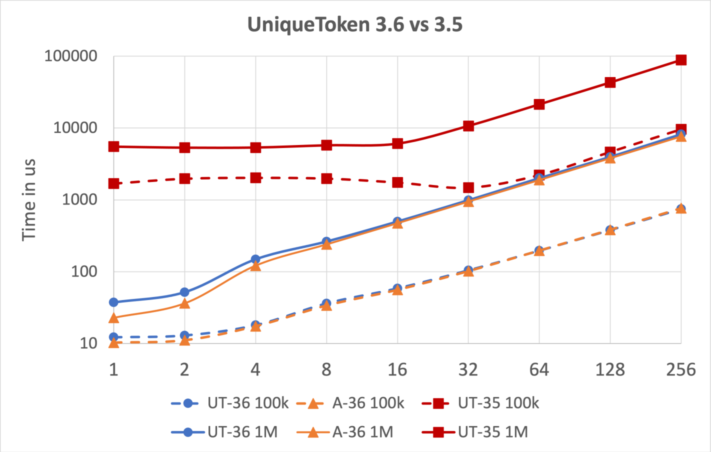
Identified Massive Performance Bug

```
UniqueToken<ExecutionSpace> token;
int N = token.size(); int M = N*x;
View<double**> dest(N,R), src(M,R);

parallel_for("UT", M, KOKKOS_LAMBDA(int i) {
  int j = token.acquire(); memory_fence();
  for(int k=0; k<R; k++) dest(j,k) += src(i,k);
  memory_fence(); token.release(j);
});

parallel_for("A" M, KOKKOS_LAMBDA(int i) {
  for(int k=0; k<R; k++) atomic_add(&dest(j,k), src(i,k));
});
```

Identified Massive Performance Bug



More Information

Reason for Performance Issue

- Unnecessary many conflicts in acquiring token.
- Indicies acquired by threads in the same warp tended to be far apart $->$ results in bad memory access pattern.

`UniqueToken` is discussed in the Kokkos Lectures Module 4!

Remember: still in `Experimental` namespace.
Feedback is welcome!

More View Allocation Properties Support

Content: Support `WithoutInitializing` for

- `resize`
- `realloc`
- `create_mirror`
- `create_mirror_view`

View Allocation Properties

Often initialization is not required when allocating.

New overloads for `resize`/`realloc` and `create_mirror[_view]` supported for `View`-like types

- `DualView`
- `DynamicView`
- `DynRankView`
- `OffsetView`
- `ScatterView`
- `View`

Kokkos::resize

```
template <class I, class T, class... P>
void
resize(const I& arg_prop, View<T, P...>& v,
       const size_t n0, const size_t n1, const size_t n2,
       const size_t n3, const size_t n4, const size_t n5,
       const size_t n6, const size_t n7);

template <class I, class T, class... P>
void
resize(const I& arg_prop, View<T, P...>& v,
       const typename View<T, P...>::array_layout& layout);
```

Resizes `v` to have the new dimensions while preserving the contents for the common subview of the old and new `View`. The new `View` is constructed using the View constructor property `arg_prop`, e.g., `WithoutInitializing`.

Kokkos::realloc

```
template <class I, class T, class... P>
void
realloc(const I& arg_prop, View<T, P...>& v,
        const size_t n0, const size_t n1, const size_t n2,
        const size_t n3, const size_t n4, const size_t n5,
        const size_t n6, const size_t n7);

template <class I, class T, class... P>
void
realloc(const I& arg_prop, View<T, P...>& v,
        const typename View<T, P...>::array_layout& layout);
```

Resizes `v` to have the new dimensions while preserving the contents for the common subview of the old and new `View`. The new `View` is constructed using the View constructor property `arg_prop`, e.g., `WithoutInitializing`.

Kokkos::create_mirror

```
template <class ViewType>
typename ViewType::HostMirror
create_mirror(decltype(WithoutInitializing),
              ViewType const& src);

template <class Space, class ViewType>
ImplMirrorType
create_mirror(decltype(WithoutInitializing),
              Space const& space, ViewType const&);
```

Creates a new host accessible `View` with the same layout and padding as `src`. The new `View` will have uninitialized data.

Kokkos::create_mirror_view

```
template <class ViewType>
typename ViewType::HostMirror
create_mirror_view(decltype(WithoutInitializing),
                   ViewType const&);

template <class Space, class ViewType>
ImplMirrorType
create_mirror_view(decltype(WithoutInitializing),
                   Space const& space, ViewType const&);
```

If `src` is not host-accessible, it creates a new host-accessible `View` with the same layout and padding as `src`. The new `View` will have uninitialized data. Otherwise returns `src`.

Section Summary

- This release:
  WithoutInitializing support for resize/realloc and create_mirror[_view] for View-like types
- Upcoming release:
  Overloads taking Kokkos::view_alloc unifying the interfaces and allow, e.g., passing execution spaces.

C++ Standard Algorithms

Kokkos implementation of a (growing set) of std algorithms

Objectives:

- Kokkos iterators
- Overview of supported algorithms
- Differences between the Kokkos and std API
- Examples
- Summary

- Iterators and std algorithms:
  - Released with **Kokkos 3.6**
  - Include via header: `Kokkos_StdAlgorithms.hpp`
- Inside the `Kokkos::Experimental`
  - Please use them and send us feedback!
- Documentation is available in the Kokkos wiki: https://github.com/kokkos/kokkos/wiki

Kokkos random-access iterators

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

Declaration:

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

Kokkos random-access iterators

`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 random-access iterators

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

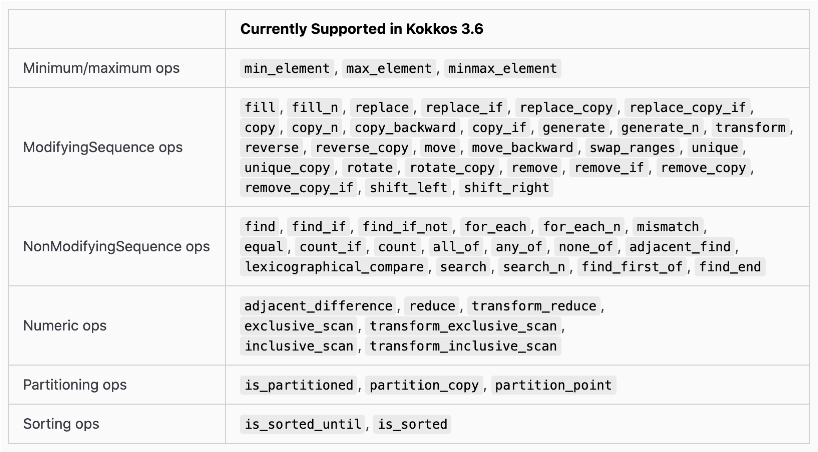
Declaration:

```
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);`

Algorithms: we use categories as in the C++ std



	Currently Supported in Kokkos 3.6
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`

Algorithms: the key difference of our API

- API accepting iterators:

```
template <class ExeSpace, ...>
<return_type> algo_name(const ExeSpace& exespace, <iterators>);

template <class ExeSpace, ...>
<return_type> algo_name(const std::string& label,
                        const ExeSpace& exespace, <iterators>);
```

- API accepting Kokkos rank-1 views:

```
template <class ExeSpace, ...>
<return_type> algo_name(const ExeSpace& exespace, <views>);

template <class ExeSpace, ...>
<return_type> algo_name(const std::string& label,
                        const ExeSpace& exespace, <views>);
```

Algorithms: the API details

```
template <class ExeSpace, ...>
<return_type> algo_name(const ExeSpace& exespace,    (1)
                        <iterators_or_views>);

template <class ExeSpace, ...>
<return_type> algo_name(const std::string& label,    (2)
                        const ExeSpace& exespace,
                        <iterators_or_views>);
```

- `exespace`: iterators/views MUST be accessible from it
- `label`: passed to the implementation kernels for debugging
  - For (1): "Kokkos::algo_name_iterator_api_default" or
    "Kokkos::algo_name_view_api_default"
- iterators: must be **random access iterators**, preferably use `Kokkos::Experimental::begin,end,cbegin,cend`
- views: rank-1, `LayoutLeft, LayoutRight, LayoutStride`

Algorithms: basic example

```
int main(){
  // ...
  namespace KE = Kokkos::Experimental;

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

  const double oldVal{2}, newVal{34};

  // act on the entire view
  KE::replace(Kokkos::DefaultHostExecutionSpace(),
              KE::begin(myView), KE::end(myView), oldVal, newVal);

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

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

Algorithms: example with custom functor for comparison

```
template <class ValueType1, class ValueType2 = ValueType1>
struct CustomLessThanComparator {
  KOKKOS_INLINE_FUNCTION
  bool operator()(const ValueType1& a, const ValueType2& b) const
  {
   // here we use < but you can put any custom logic needed
   return a < b;
  }
};

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>());
  //...
}
```

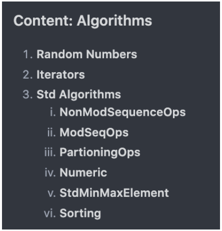
Algorithms: general considerations

- Implementations rely on Kokkos parallel for, reduce or scan.
- Debug mode enables several checks, e.g.: whether iterators identify a valid range, the execution space accessibility, etc., and error messages printed.
- If needed, algorithms fence directly the execution space instance provided:

```
template <class ExeSpace, ...>
<return_type> algo_name(const ExeSpace& exespace, ...)
{
  // implementation
  exespace.fence(/*string depends on algorithm*/);
}
```

Section Summary

- Starting with Kokkos 3.6, Kokkos offers many std algorithms
- Two main APIs: one for iterators and one for rank-1 views
- Checkout the documentation in the Kokkos wiki



Content: Algorithms

1. Random Numbers
2. Iterators
3. Std Algorithms
   i. NonModSequenceOps
   ii. ModSeqOps
   iii. PartioningOps
   iv. Numeric
   v. StdMinMaxElement
   vi. Sorting

Figure: Wiki documentation

- Useful to make your code more expressive, allowing us to worry about having performant implementations
- Please use them, and let us know of any issues!
- Try them with the new feature: `Kokkos::Experimental::partition_space`
- In progress: team-level implementations

Numerics

Content:

- Common mathematical functions
- Mathematical constants
- Numeric traits

Common math functions

Improvement/Bug fix

Unconditionally define `long double` overloads on the host side

```
namespace Kokkos::Experimental {
KOKKOS_FUNCTION float       sqrt ( float x );
KOKKOS_FUNCTION float       sqrtf( float x );
KOKKOS_FUNCTION double      sqrt ( double x );
                long double sqrt ( long double x ); // 3.6
                long double sqrtl( long double x ); // 3.6
KOKKOS_FUNCTION double      sqrt ( IntegralType x );
}
```

Looking ahead

Math functions promoted to the `Kokkos::` namespace in 3.7

Mathematical constants

- Defined in header `<Kokkos_MathematicalConstants.hpp>` which is included from `<Kokkos_Core.hpp>`
- Provides all mathematical constants from `<numbers>` (since C++20), such as `pi` and `sqrt2`
- All constants are defined in the `Kokkos::Experimental::` namespace since Kokkos 3.6

Improvement/Bug fix

- Add missing traits `denorm_min`, `reciprocal_overflow_threshold`, and `{quiet,silent}_NaN`
- Instantiate numeric traits on cv-qualified types

Section Summary

- Consistent and portable overload set for standard C library mathematical functions, such as `fabs`, `sqrt`, and `sin`
- Backport of the C++20 standard library header `<numbers>` and provides several mathematical constants, such as `pi` or `sqrt2`
- New facility that is being added to the C++23 standard library and is intended as a replacement for `std::numeric_limits`

`KOKKOS_IF_ON_{HOST,DEVICE}` macros

#ifdef __CUDA_ARCH__ idiom not portable

Motivating example

```
__host__ __device__ void terminate() {
#ifdef __CUDA_ARCH__
  asm("trap;");  // inline PTX assembly when called on device
#else
  _exit();       // OS call when called on the host
#endif
}
```

- NVIDIA HPC compiler uses a unified heterogeneous compilation model (single-pass)
- NVC++ cannot support `__CUDA_ARCH__` because that assumes split compilation

Overloading based on `__host__` and `__device__` attributes

```
struct MyS { int i; };
#ifdef __NVCC__
  #ifndef __CUDA_ARCH__
    __host__ MyS MakeStruct() { return MyS{0};}
  #else
    __device__ MyS MakeStruct() { return MyS{1};}
  #endif
#else
    __host__ MyS MakeStruct() { return MyS{0};}
    __device__ MyS MakeStruct() { return MyS{1};}
#endif
```

Different class on host/device (NOT SUPPORTED)

```
struct solver {
  // ...
  #ifndef __CUDA_ARCH__
  std::ofstream output_;
  #endif
};
```

KOKKOS_IF_ON_{HOST,DEVICE} macros

Revisit overloading on host and device example

```
struct MyS { int i; };
KOKKOS_FUNCTION MyS MakeStruct() {
  KOKKOS_IF_ON_HOST(( return MyS{0}; ))
  KOKKOS_IF_ON_DEVICE(( return MyS{1}; ))
}
```

Things to note

- Both macros introduce a new scope

  ```
  KOKKOS_IF_ON_HOST((
    int x = 0;
    std::cout << x << ’\n’;
  )) // scope of ’x’ ends here
  ```

- Cannot be used in a context that requires constant expressions (`constexpr`)
- Do not play nice with other preprocessor directives

Preprocessor directives

```cpp
KOKKOS_FUNCTION void host_compute() {
#if KOKKOS_VERSION >= 30700
  auto sqrt2f = Kokkos::sqrtf(2);
  // ...
#else
  auto sqrt2f = 1.41421356237f;
  // ...
#endif
}

KOKKOS_FUNCTION void device_compute() { /* ... */ }

KOKKOS_FUNCTION decltype(auto) compute() {
  KOKKOS_IF_ON_HOST(( return host_compute(); ))
  KOKKOS_IF_ON_DEVICE(( return device_compute(); ))
}
```

Section Summary

- #ifdef __CUDA_ARCH__ idiom is not portable
- Release 3.6 introduces two macros: KOKKOS_IF_ON_HOST and KOKKOS_IF_ON_DEVICE

Warning!

Avoid using as much possible. These macros are a last resort facility for differentiating between host and device inside a kernel. Consider other approaches such as partial template specialization on execution spaces.

- Upcoming support for NVC++ (in the next release or two)

Kernels update

- Architectures support
- Batched linear solvers
- Block Sparse Matrices
- Batched GEMM
- Mixed precision

Platforms/Architecture support (Brian, Luc)

Architecture support:

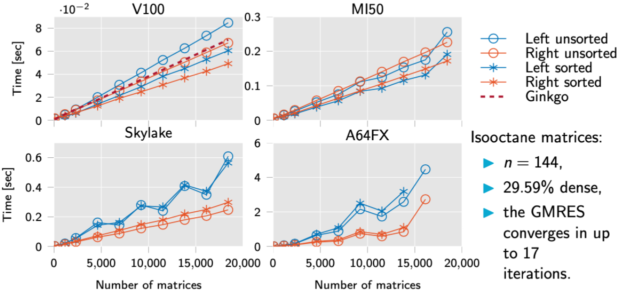
- Nvidia fully supported with Cuda backend
- AMD fully supported with HIP backend, still optimizing for performance
- Intel initial support with SYCL backend, more testing and performance optimization needed

Spack updated with release 3.6.0, build tested on Summit, Spock/Crusher and initial support on Arcticus.
Starting to support streams on device, inquire for details.

Batched Linear Solvers (Kim)

New batched linear solvers are introduced

- LU with static pivoting
- PCG
- GMRES



Isooctane matrices:

- $n = 144$,
- 29.59% dense,
- the GMRES converges in up to 17 iterations.

Block Sparse Matrices: BsrMatrix (NGA, Luc)

New BsrMatrix matrix format implemented, supports constant block size sparse matrix mostly geared toward multi-physics systems representation.

Currently supported algorithms:

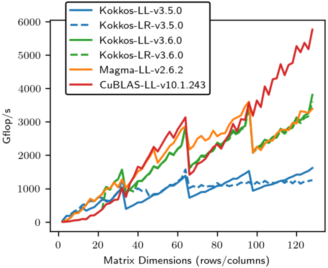
- BrsMatrix
- Matrix-Vector product using SpMV interface
- Matrix-Matrix product using SpGEMM interface
- Gauss-Seidel smoother

The new format requires less memory and exposes dense linear algebra usage within sparse linear algebra kernels, leading to increased performance compared to point CrsMatrix.

Batched GEMM improvements (Evan, Vinh)

New heuristics and improved interface included a unified interface for all levels of parallelism (TeamVector, Team and Serial) for simplicity.

- row-major speedup is 1.17x
- column-major speedup is 1.26x
- dimensions 2 to 24: single parallel-for with a RangePolicy over entries of C
- dimensions > 24: double buffering algorithm based on Magma's BatchedGemm, Kokkos team cooperatively works on a tile.



Future work includes additional optimizations to the Kokkos BatchedGemm algorithm.

Mixed precision algorithms (Jennifer)

- Kokkos Kernels provides mixed precisions linear algebra kernels.
- GMRES with iterative refinement runs in single precision, residual achieves double precision via iterative refinement
- Kokkos Kernels is also providing interfaces for experiments with 16-bit precisions

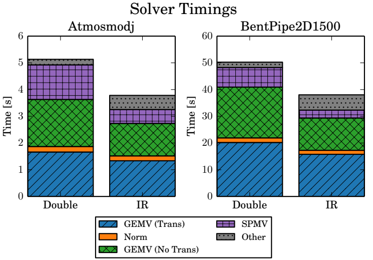


Figure: Solve times for GMRES(50) double (left) and IR (right) for the matrices Atmosmodj and BentPipe2D1500. Each bar represents total solve time, split up to give a breakdown of time spent in different kernels.

Multiple new and ongoing collaborations are cultivated

- Nvidia: LU factorization for dense systems
- Ginkgo: development of batched gmres
- PETSc: providing a portable algebra layer
- ExaWind: preconditioner techniques
- AMD: library optimization and MFMA usage
- ANL: porting and testing on Intel platform
- NASA: performance optimization of sparse matrix-vector product

and probably many more that I forget...

Focus of future work

- Performance optimization in Block Sparse algorithms
- Format conversion of sparse matrix: Csc2Csr, Coo2Csr
- Sparse ILU and TRSV performance improvements
- more batched solver features and new batched ODE solvers
- more stream support for BLAS and Sparse kernels
- fast iterative ILUt algorithm