Kokkos Core Status Update

Christian R. Trott, - Center for Computing Research
Sandia National Laboratories/NM

Unclassified Unlimited Release

D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, J. Ciesko,
Some Kokkos Stats Since 2015

- 18 Releases Since 2016
  - Only 5 since December 2017
- 50 Contributors
  - 17 with more than 10 commits
  - 11 with more than 10k lines touched
- 1345 Issues of which 1134 were resolved
  - 305 bug reports
  - 381 enhancement requests
  - 129 Feature Requests
- 766 pull requests
- 15k messages on kokkosteam.slack.com (Started in 2017)
Kokkos SIMD

- SIMD Support for diverse architectures
- Based on ISO C++ TS
- `simd<double,ABI>`
  - ABI are things like “AVX”, “AVX512”, “NEON”, ”SVE”, “FixedSize<16>”
- Differentiate storage SIMD type from temporary
  - Allow storage of 32 consecutive values
  - Load 1 value per CUDA thread on GPU
- For now: [https://github.com/kokkos/simd-math](https://github.com/kokkos/simd-math)
  - Will move into core Kokkos soon though.
Containers: ScatterView

- Encapsulates common design pattern in reduction algorithms using either data duplication and/or atomics
  - Data duplication is often faster on the host, but too memory expensive on GPUs.
  - Atomics are faster on GPUs, but extremely slow on the host

ScatterView<Datatype, Layout, ExecSpace, ReduceOp, DupMode, ContribMode>

ReduceOp: ScatterSum, ScatterProd, ScatterMax, ScatterMin
DupMode: ScatterNonDuplicated, ScatterDuplicated
ContribMode: ScatterNonAtomic, ScatterAtomic
Containers: ScatterView (cont’d)

ScatterView<
double, LayoutRight, Cuda, ScatterSum, ...> sv(...);
View<
double, LayoutRight, Cuda> v(...);

parallel_for(n, [=](int i){
    auto scatter_access = sv.access();
    int k = foo(i);
    double x = bar(x);
    scatter_access(k) += x;
});

contribute(v, sv);
UniqueToken

- Generates a unique ordinal based on the concurrency of the ExecutionSpace
  - Can be used to index into resources that are restricted by the amount of concurrency available
- Ordinals can be local to a single kernel instance or global across all kernels
- Threads first acquire a token and then release it afterwards
- For the best performance
  - Tokens should be acquired/released in as narrow of scope as possible, and
  - Tokens should be released before calling a team_barrier or similar construct
Asynchronicity Semantics

- **ParallelReduce/Scan**

```cpp
double result;
// parallel_for is always Synchronous
parallel_for("AsynchronousFor", N, F);
// parallel_reduce with Scalar as result is Synchronous
parallel_reduce("SynchronousSum", N, Fr, result);
// parallel_reduce with Reducer constructed from scalar is synchronous
parallel_reduce("SynchronousMax", N, Fr, Max<double>(result));
// parallel_reduce with any type of View as result is asynchronous
Kokkos::View<double, CudaHostPinnedSpace> result_v("R");
parallel_reduce("AsynchronousSum", N, Fr, result_v);
// Even with unmanaged view, and wrapped into Reducer
Kokkos::View<double, HostSpace> result_hv(&result);
parallel_reduce("AsynchronousMax", N, Fr, Max<double>(result_hv));
// Scans without total result argument are asynchronous
parallel_scan("AsynchronousScan", N, Fs);
// Scans with total result argument same rules as parallel_reduce
parallel_scan("SynchronousScanTotal", N, Fs, result);
```
CUDA Stream Interop

- Initial step to full coarse grained tasking
  - Discuss in more detail in future directions
- For now: make Kokkos dispatch use user CUDA streams
  - Allows for overlapping kernels: best for large work per iteration, low count

```cpp
// Create two Cuda instances from streams
cudaStream_t stream1, stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
Kokkos::Cuda cuda1(stream1), cuda2(stream2);

// Run two kernels which can overlap
parallel_for("F1", RangePolicy<Kokkos::Cuda>(cuda1, N), F1);
parallel_for("F2", RangePolicy<Kokkos::Cuda>(cuda2, N), F2);
fence();
```
DOE Machine Announcements

- Now publicly announced that DOE is buying both AMD and Intel GPUs
  - Argonne: Cray with Intel Xeon + Intel Xe Compute
  - ORNL: Cray with AMD CPUs + AMD GPUs
  - NERSC: Cray with AMD CPUs + NVIDIA GPUs

- Have been planning for this eventuality:
  - Kokkos ECP project extended and refocused to include developers at Argonne, Oak Ridge, and Lawrence Berkeley - staffing is in place
  - HIP backend for AMD: main development at ORNL
    - The current ROCm backend is based on a compiler which is now deprecated ...
  - SYCL for Intel: main development at ANL
  - OpenMPTarget for AMD, Intel and NVIDIA, lead at Sandia
OpenMP-Target Backend

- With Clang mainline we got a working compiler
  - Only “officially” supported compiler right now
  - Adding IBM XL, AMD aomp, Intel, NVIDIA and GCC as soon as we can verify them
- Testing in place
- Basic capabilities are working:
  - RangePolicy, MDRangePolicy
  - Data Movement
  - parallel_for/reduce
- Performance pretty spotty
**HIP Backend**

- Restart of the AMD work we previously did
- Work lead by ORNL
- Basic capabilities are in place
  - RangePolicy, MDRangePolicy
  - Data Movement
  - parallel_for/reduce
- Tests can be enabled
- Performance Ok-ish so far
OneAPI Backend

- Tools
  - DPC++ (OneAPI/SYCL compiler from Intel based on clang)
    - Need OneAPI extensions to implement Kokkos
      - Unnamed lambda support
      - Primitives for host vs. device memory
  - NEO Driver
    - Weird bugs: Couldn’t pass pointers in a struct to device
  - Longer term (may be years from now)
    - Intel OneAPI extensions proposed for SYCL

- Early days
  - Parallel_for
  - USMMemory space Rank 1
  - Functionality testing on Gen 9 hardware
<table>
<thead>
<tr>
<th>Feature</th>
<th>HIP</th>
<th>DPC++</th>
<th>OpenMP Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemorySpace</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>parallel_for RangePolicy</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>parallel_for MDRrangePolicy</td>
<td>X</td>
<td>03/20</td>
<td>X</td>
</tr>
<tr>
<td>parallel_reduce RP</td>
<td>X</td>
<td>02/20</td>
<td>X</td>
</tr>
<tr>
<td>parallel_reduce MDRP</td>
<td>05/20</td>
<td>Q4 20</td>
<td>05/20</td>
</tr>
<tr>
<td>Reducers</td>
<td>X</td>
<td>Q4 20</td>
<td>X</td>
</tr>
<tr>
<td>parallel_for TP</td>
<td>03/20</td>
<td></td>
<td>03/20</td>
</tr>
<tr>
<td>parallel_reduce TP</td>
<td>06/20</td>
<td></td>
<td>06/20</td>
</tr>
<tr>
<td>atomics</td>
<td>03/20</td>
<td></td>
<td>04/20</td>
</tr>
</tbody>
</table>
Modern CMake wants a clean separation of ‘building’ and ‘using’ libraries

- CMake 3 (first “modern” version) released June 2014
  - Clean separation of building and using (targets and properties) has been recommended method since release
- All options should be applied specifically to TARGETS (libs, exes)
  - No more directly modifying CMAKE_CXX_FLAGS
  - No more global setting include directories and compiler flags
  - Your compiler/linker flags should be specific and exact to an individual library
- All include directories and compiler flags should be clearly defined as:
  - PUBLIC: Flag needed to build Kokkos and needed downstream to use Kokkos
    - Kokkos headers
    - Flags like –fopenmp or CUDA flags needed for the backend
    - Minimum C++ standards
  - PRIVATE: Flag only needed to build Kokkos (not needed to use)
    - Certain warning flags
    - Certain optimization flags
What should CMake look like for using Kokkos?

A single CMake function should populate build with all the necessary flags to build correctly and all the optimization/architecture flags to improve performance.

```cmake
find_package(Kokkos REQUIRED)
add_library(target ${SOURCES})
target_link_libraries(target PUBLIC Kokkos::kokkos)

find_package(Kokkos REQUIRED)
add_library(target ${SOURCES})
target_link_libraries(target PRIVATE Kokkos::kokkos)

KOKKOS_CHECK(
  DEVICES CUDA OPENMP
  OPTIONS CUDA_RELOCATABLE_DEVICE_CODE
  ARCH VOLTA70
)

Installed Kokkos: cmake -DKokkos_ROOT=<PREFIX>
In-tree Kokkos: add_subdirectory(kokkos)
```

I need Kokkos to build – and anyone using my API needs Kokkos

I need Kokkos to build – but using my API does not require Kokkos

Assert that the Kokkos configuration found meets expectations
Building Kokkos

- `cmake ${KOKKOS_SOURCE} -D{OPTION}:BOOL=ON -D{OPTION}:STRING=NAME`
  - Via command Line
- To get a list of options, use `ccmake`
  - `ccmake -DCMAKE_CXX_COMPILER={} ${KOKKOS_SOURCE}`
Building and using makes “smaller” interfaces between libraries, solves transitive dependencies

Application should only know about its direct dependencies

target_link_libraries(Ifpack2) makes C++ App depend transitively on Kokkos flags (PUBLIC)

Automake requires collecting and forwarding, e.g.

KokkosKernels_CXX_FLAGS = $(LOCAL_CXX_FLAGS) + $(Kokkos_CXX_FLAGS)

target_link_libraries(Ifpack2_C) does not make C App depend transitively on Kokkos flags (PRIVATE)
Kokkos Tools

- Profiling
  - New tools are coming out
  - Worked with NVIDIA to get naming info into their system
- Auto Tuning
  - Internal variables such as CUDA block sizes etc.
  - User provided variables
  - Same as profiling: will use dlopen to load external tools
- Debugging
  - Extensions to enable clang debugger to use Kokkos naming information
- Static Analysis
  - Discover Kokkos anti patterns via clang-tidy
Kokkos Tools Integration with 3rd Party

- Profiling Hooks can be subscribed to by tools, and currently have support for TAU, Caliper, Timemory, NVVP, Vtune, PAPI, and SystemTAP, with planned CrayPat support.
- HPCToolkit also has special functionality for models like Kokkos, operating outside of this callback system.

**TAU Example:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU application</td>
<td>0.143</td>
<td>96.743</td>
<td>1</td>
<td>832</td>
</tr>
<tr>
<td>Comm::exchange</td>
<td>0.001</td>
<td>0.967</td>
<td>6</td>
<td>142</td>
</tr>
<tr>
<td>Comm::exchange_halo</td>
<td>0.001</td>
<td>4.702</td>
<td>6</td>
<td>184</td>
</tr>
<tr>
<td>Comm::update_halo</td>
<td>0.004</td>
<td>31.347</td>
<td>95</td>
<td>1,330</td>
</tr>
<tr>
<td>Kokkos::parallel_for_CommMPI::halo_update_pack [device=0]</td>
<td>0.002</td>
<td>0.506</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>Kokkos::parallel_for_CommMPI::halo_update_self [device=0]</td>
<td>0.003</td>
<td>0.597</td>
<td>380</td>
<td>380</td>
</tr>
<tr>
<td>Kokkos::parallel_for_CommMPI::halo_update_unpack [device=0]</td>
<td>0.002</td>
<td>0.97</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>MPI_Irecv()</td>
<td>0.001</td>
<td>0.001</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Send()</td>
<td>29.268</td>
<td>29.268</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Wait()</td>
<td>0.001</td>
<td>0.001</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>OpenMP_Implicit_Task</td>
<td>0.041</td>
<td>1.985</td>
<td>760</td>
<td>760</td>
</tr>
<tr>
<td>OpenMP_Parallel Region parallel_for&lt;Kokkos::RangePolicy&lt;CommMPI::Ta</td>
<td>0</td>
<td>0.504</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>OpenMP_Parallel Region parallel_for&lt;Kokkos::RangePolicy&lt;CommMPI::Ta</td>
<td>0.08</td>
<td>0.968</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>OpenMP_Parallel Region void Kokkos::parallel_for&lt;Kokkos::RangePolicy &lt;</td>
<td>0.001</td>
<td>0.594</td>
<td>380</td>
<td>380</td>
</tr>
<tr>
<td>OpenMP_Sync Region_BARRIER parallel_for&lt;Kokkos::RangePolicy&lt;CommMPI</td>
<td>0.489</td>
<td>0.489</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>OpenMP_Sync Region_BARRIER parallel_for&lt;Kokkos::RangePolicy&lt;CommMPI</td>
<td>0.875</td>
<td>0.875</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>OpenMP_Sync Region_BARRIER void Kokkos::parallel_for&lt;Kokkos::RangePol</td>
<td>0.58</td>
<td>0.58</td>
<td>380</td>
<td>0</td>
</tr>
</tbody>
</table>
Kokkos Tools Static Analysis

- clang-tidy passes for Kokkos semantics
- Under active development, requests welcome
- IDE integration

```cpp
// Base case
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
        int a = 0;
        Kokkos::parallel_for(TTR(t, 1), [i](int i) {
            Lambda capture modifies reference capture variable 'a' that is a local
            a += 1;
            cv() += 1;
        });
    });

// One with variable Lambda
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
        int b = 0;
        auto lambda = [i](int i) {
            Lambda capture modifies reference capture variable 'b' that is a local
            b += 1;
            cv() += 1;
        };
        Kokkos::parallel_for(TTR(t, 1), lambda);
    };
```
Aligning Kokkos with the C++ Standard

- Long term goal: move capabilities from Kokkos into the ISO standard
  - Concentrate on facilities we really need to optimize with compiler

Move accepted features to legacy support

Propose for C++

Implemented legacy capabilities in terms of new C++ features

Back port to compilers we got
C++ Features in the Works

- First success: `atomic_ref<T>` in C++20
  - Provides atomics with all capabilities of atomics in Kokkos
    - `atomic_ref(a[i])+=5.0;` instead of `atomic_add(&a[i], 5.0);`
- Next thing: `Kokkos::View => std::mdspan`
  - Provides customization points which allow all things we can do with `Kokkos::View`
  - Better design of internals though! => Easier to write custom layouts.
  - Also: arbitrary rank (until compiler crashes) and mixed compile/runtime ranks
  - We hope will land early in the cycle for C++23 (i.e. early in 2020)
  - Production reference implementation: [https://github.com/kokkos/mdspan](https://github.com/kokkos/mdspan)
- Also C++23: Executors and **Basic Linear Algebra**: [https://github.com/kokkos/stdlibas](https://github.com/kokkos/stdlibas)
Links

- **https://github.com/kokkos** Kokkos Github Organization
  - **Kokkos:** *Core library, Containers, Algorithms*
  - **Kokkos-Kernels:** *Sparse and Dense BLAS, Graph, Tensor (under development)*
  - **Kokkos-Tools:** *Profiling and Debugging*
  - **Kokkos-MiniApps:** *MiniApp repository and links*
  - **Kokkos-Tutorials:** *Extensive Tutorials with Hands-On Exercises*

- **https://cs.sandia.gov** Publications (search for ’Kokkos’)
  - Many Presentations on Kokkos and its use in libraries and apps

- **http://on-demand-gtc.gputechconf.com** Recorded Talks
  - Presentations with Audio and some with Video

- **https://kokkosteam.slack.com** Slack channel for user support