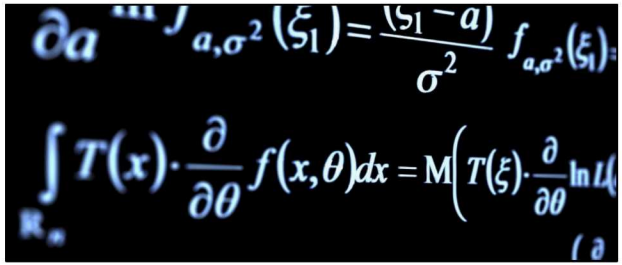
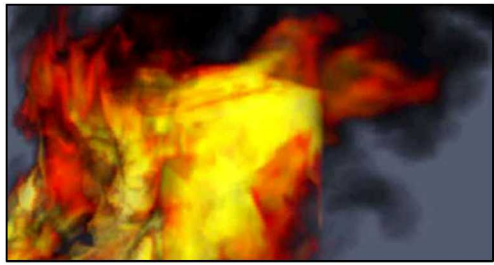
SAND2020-1507PE



$$\frac{\partial}{\partial a} f_{a,\sigma^2}(\xi_1) = \frac{(\xi_1 - a)}{\sigma^2} f_{a,\sigma^2}(\xi_1)$$

$$\int T(x) \cdot \frac{\partial}{\partial \theta} f(x,\theta) dx = \mathrm{M}\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L(\right.$$



Kokkos Core Status Update

Unclassified Unlimited Release

D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, J. Ciesko, H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin, J. Wilke, D. Arndt, R. Gayatri, J. Madsen

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

Sandia National Laboratories is a multimission 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-NA-0003525.

Kokkos Development Team

Kokkos Core:	***C.R. Trott,*** *D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, J. Ciesko, H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin, J. Wilke, D. Arndt, R. Gayatri, J. Madsen* *former:* ***H.C. Edwards****, D. Labreche, G. Mackey, S. Bova*
Kokkos Kernels:	***S. Rajamanickam,*** *N. Ellingwood, K. Kim, C.R. Trott, V. Dang, L. Berger,*
Kokkos Tools:	***D. Poliakoff,*** *S. Hammond, C.R. Trott, D. Ibanez, S. Moore*
Kokkos Support:	***C.R. Trott,*** *G. Shipman, G. Lopez, G. Womeldorff,* *former:* ***H.C. Edwards****, D. Labreche, Fernanda Foertter*

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

Sandia National Laboratories

- SIMD Support for diverse architectures
- Based on ISO C++ TS
- simd<double,ABI>
  - ABI are things like “AVX”, “AVX512”, “NEON”, ”SVE”
- 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
  - Will move into core Kokkos soon though.

Containers: ScatterView

Sandia National Laboratories

- 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)

Sandia National Laboratories

```
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

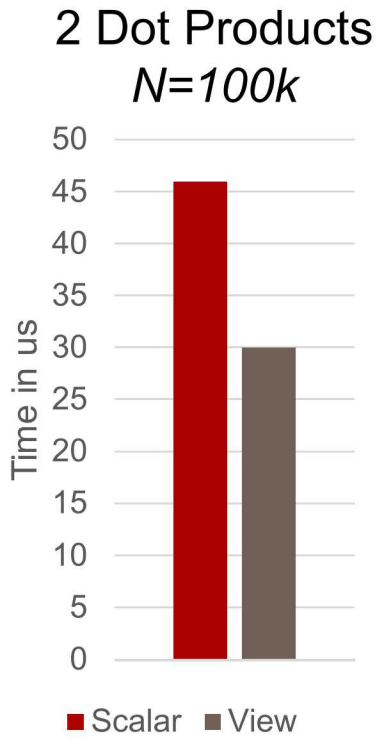
Sandia National Laboratories

- 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

```
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

```
// 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

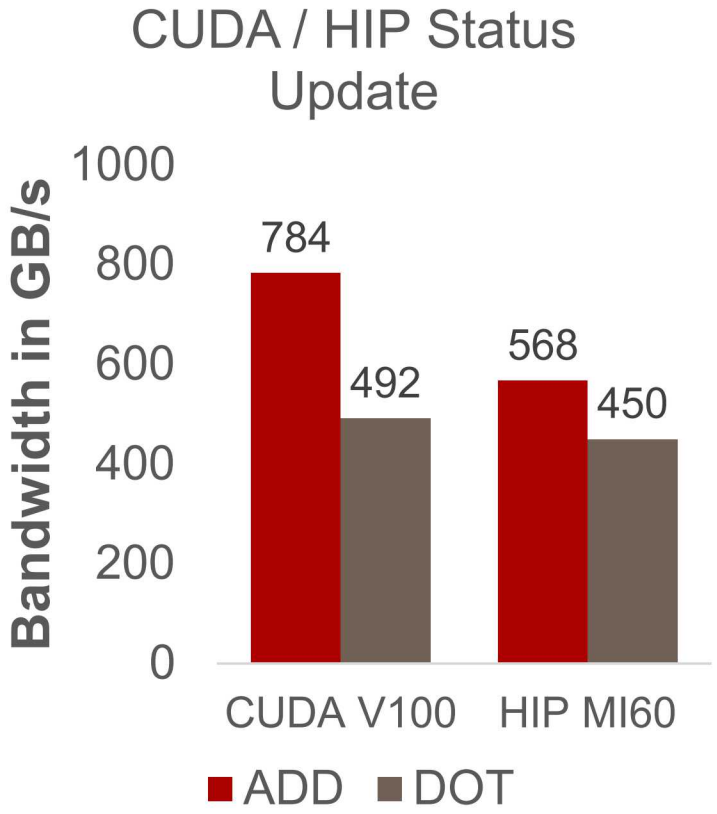
OpenMP-Target Backend

Sandia National Laboratories

- 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

Feature Timeline

Feature	HIP	DPC++	OpenMP Target
MemorySpace	X	X	X
parallel_for RangePolicy	X	X	X
parallel_for MDRrangePolicy	X	03/20	X
parallel_reduce RP	X	02/20	X
parallel_reduce MDRP	05/20	Q4 20	05/20
Reducers	X	Q4 20	X
parallel_for TP	03/20		03/20
parallel_reduce TP	06/20		06/20
atomics	03/20		04/20

Modern CMake wants a clean separation of 'building' and 'using' libraries

Sandia National Laboratories

- 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*

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

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

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

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

```
KOKKOS_CHECK(
  DEVICES CUDA OPENMP
  OPTIONS CUDA_RELOCATABLE_DEVICE_CODE
  ARCH VOLTA70
)
```

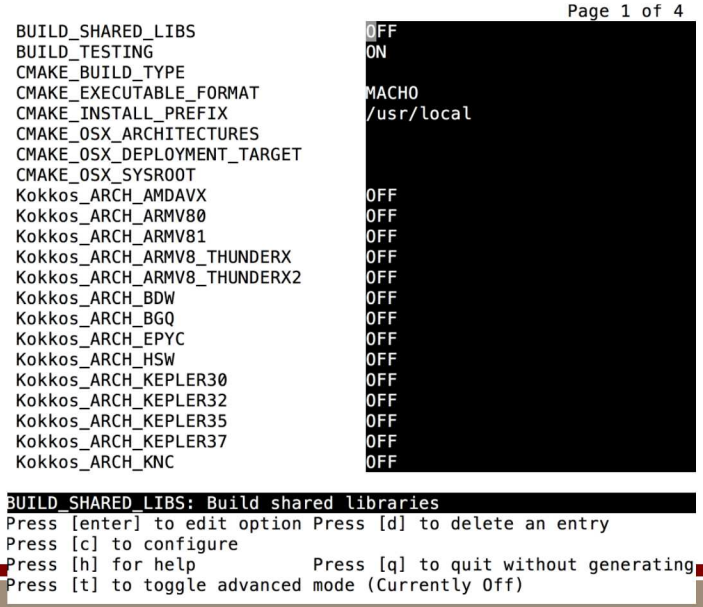
Assert that the Kokkos configuration found meets expectations

Installed Kokkos: cmake –DKokkos_ROOT=<PREFIX>

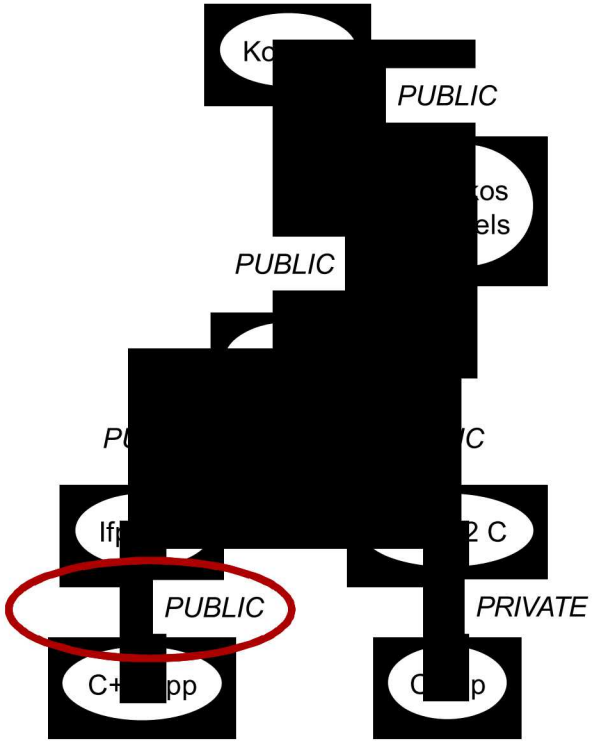
In-tree Kokkos: add_subdirectory(kokkos)

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(lfpack2) 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(lfpack2_C) does not make C App depend transitively on Kokkos flags (PRIVATE)

Kokkos Tools

Sandia National Laboratories

- 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:

TAU: ParaProf: Statistics for: node 0, thread 0 - examinimd_ompt_phase.ppk

Name	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	0.143	96.743	1	832
Comm::exchange	0.001	0.967	6	142
Comm::exchange_halo	0.001	4.702	6	184
Comm::update_halo	0.004	31.347	95	1,330
Kokkos::parallel_for CommMPI::halo_update_pack [device=0]	0.002	0.506	190	190
Kokkos::parallel_for CommMPI::halo_update_self [device=0]	0.003	0.597	380	380
Kokkos::parallel_for CommMPI::halo_update_unpack [device=0]	0.002	0.97	190	190
MPI_Irecv()	0.001	0.001	190	0
MPI_Send()	29.268	29.268	190	0
MPI_Wait()	0.001	0.001	190	0
OpenMP_Implicit_Task	0.041	1.985	760	760
OpenMP_Parallel_Region parallel_for<Kokkos::RangePolicy<CommMPI::Ta	0	0.504	190	190
OpenMP_Parallel_Region parallel_for<Kokkos::RangePolicy<CommMPI::Ta	0.08	0.968	190	190
OpenMP_Parallel_Region void Kokkos::parallel_for<Kokkos::RangePolicy<	0.001	0.594	380	380
OpenMP_Sync_Region_Barrier parallel_for<Kokkos::RangePolicy<CommMf	0.489	0.489	190	0
OpenMP_Sync_Region_Barrier parallel_for<Kokkos::RangePolicy<CommMf	0.875	0.875	190	0
OpenMP_Sync_Region_Barrier void Kokkos::parallel_for<Kokkos::RangePol	0.58	0.58	380	0

Kokkos Tools Static Analysis

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

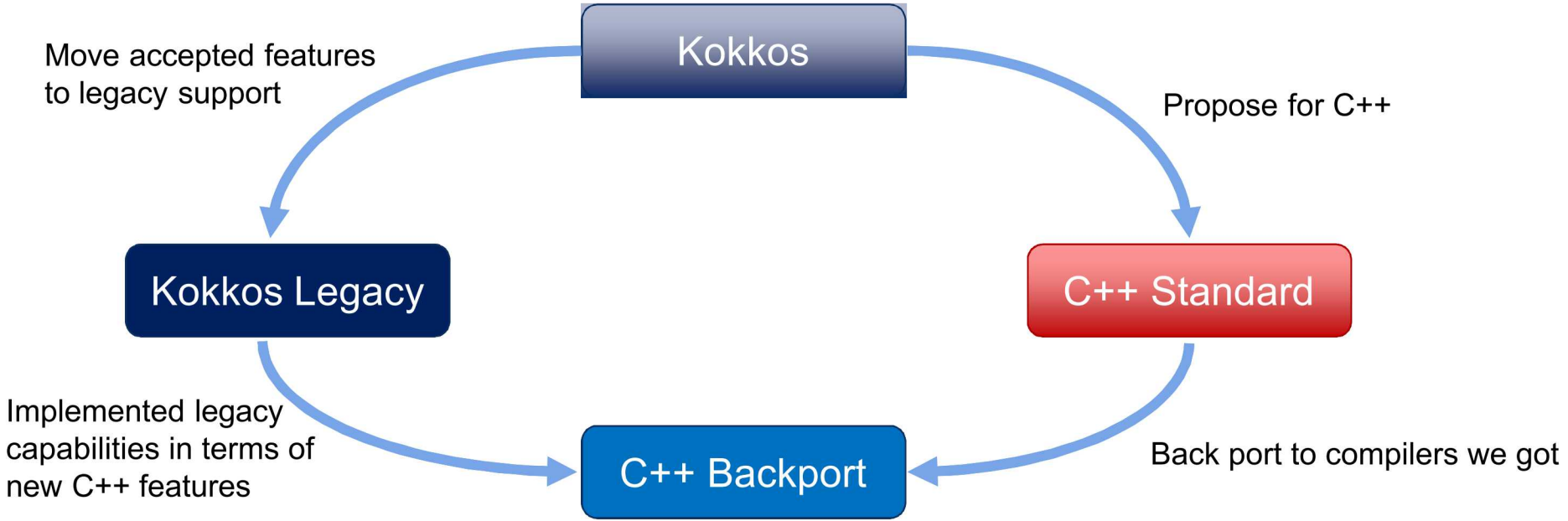
```
// Base case
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int a = 0;

      Kokkos::parallel_for(TTR(t, 1), [&](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 = [&](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



C++ Features in the Works

Sandia National Laboratories

- 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
- Also C++23: Executors and **Basic Linear Algebra:** https://github.com/kokkos/stdblas

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