

# The Kokkos Lectures

## Module 8: Kokkos Kernels Math Library

May 20, 2026

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.  
SAND2020-9315 TR

## 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 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.

- ▶ Module 1: Introduction, Building and Parallel Dispatch
- ▶ Module 2: Views and Spaces
- ▶ Module 3: Data Structures + MultiDimensional Loops
- ▶ Module 4: Hierarchical Parallelism
- ▶ Module 5: Tasking, Streams and SIMD
- ▶ Module 6: Internode: MPI and PGAS
- ▶ Module 7: Tools: Profiling, Tuning and Debugging
- ▶ **Module 8: Kernels: Sparse and Dense Linear Algebra**
- ▶ Module 9: Fortran inter-op

## Kokkos Tools:

- ▶ Kokkos Tools provide an instrumentation interface **KokkosP** and **Tools** to leverage it.
- ▶ The interface is **always available** - even in release builds.
- ▶ Zero overhead if no tool is loaded during the run.
- ▶ Dynamically load a tool via setting `KOKKOS_TOOLS_LIBS` environment variable.
- ▶ Set callbacks in code for tools compiled into the executable.

## Kokkos Connector Tools:

- ▶ Connectors inject Kokkos specific information into vendor and academic tools.
- ▶ Helps readability of profiles.
- ▶ Removes need to put vendor specific instrumentation in codes.
- ▶ Growing list of tools support Kokkos natively.

## **Kokkos Tuning Hooks enable more performance portability**

- ▶ Avoid figuring out the right heuristic for every platform.
- ▶ Input variables describe the problem scope.
- ▶ Output variables describe the search space.

## **Implementing your own tools is easy!**

- ▶ Simply implement the needed C callback functions.
- ▶ Only implement what you need.
- ▶ The callback registration system allows to embed tools in applications.

## **Static Analysis**

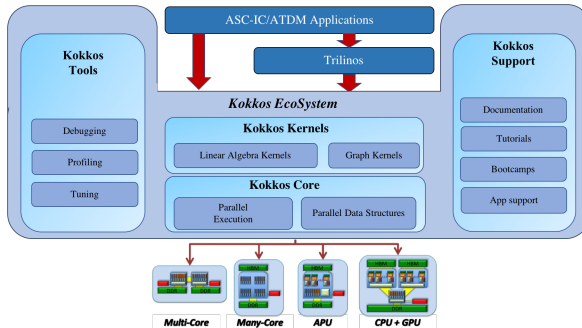
- ▶ Have semantic checks going beyond C++ errors.
- ▶ Integrates into your editors.

# Kokkos Kernels: Library Based Approach for Performance Portable Sparse/Dense linear algebra and Graph Kernels

## **Presented by:**

Siva Rajamanickam, S. Acer, L. Berger-Vergiat, V. Dang, N. Ellingwood, E. Harvey, B. Kelley, K. Kim, C.R. Trott, J. Wilke

## Kokkos Ecosystem for Performance Portability



**Kokkos Core:** parallel patterns and data structures; supports several execution and memory spaces

**Kokkos Kernels:** performance portable BLAS; sparse, dense and graph algorithms

**Kokkos Tools:** debugging and profiling support

Kokkos Ecosystem addresses complexity of supporting numerous many/multi-core architectures that are central to DOE HPC enterprise

Deliver **portable** sparse/dense linear algebra and graph kernels

- ▶ These are the kernels that are in 80% of time for most applications
- ▶ Key problems: Kernels might need different algorithms/implementations to get the best performance
- ▶ Ninja programming needs in addition to Kokkos
- ▶ Users of the kernels do not need to be ninja programmers
- ▶ **Focus on performance of the kernels on all the platforms of interest to DOE**

Kokkos Kernels delivers portable, high-performance kernels in a robust software ecosystem to support CSE applications

Deliver **robust software ecosystem** for other software technology projects and applications

- ▶ Production software capabilities that give high performance, portable and turn-key
- ▶ Tested on number of configurations nightly (architectures, compilers, debug/optimized, programming model backend, complex/real, ordinal types...)
- ▶ Larger release/integration testing with Trilinos and applications
- ▶ Kokkos Support, github issues, tutorials, hackathons, user group meetings, slack

Kokkos Kernels delivers portable, high-performance kernels in a robust software ecosystem to support CSE applications

Serve as **reference implementation** of key kernel needs of applications

- ▶ Actively work with vendors to develop high performance implementation in their libraries
- ▶ Provide interface to vendor implementations where they are better
- ▶ Actively publish the algorithms so the community develops even better variations

Kokkos Kernels delivers portable, high-performance kernels in a robust software ecosystem to support CSE applications

Actively partner with Applications to identify new opportunities for performance

- ▶ Actively publish the algorithms so the community develops even better variations
- ▶ **Team-level dense, sparse linear algebra**
- ▶ **Team-level data structures (hashmap) and utilities (sorting) for better performance**
- ▶ **Fused Kernels**
- ▶ **Symbolic and Numeric separation in interface design**

Kokkos Kernels delivers portable, high-performance kernels in a robust software ecosystem to support CSE applications

## NVIDIA

- ▶ Summit on Summit meetings
- ▶ Biweekly work stream meetings to guide NVIDIA's math libraries plans
- ▶ Kernel requirements prioritized by application needs and milestones
- ▶ Long history of interaction as part of COE
- ▶ SpGEMM, GEMM, Solvers are all improved

## ARM

- ▶ Working with the math libraries team both on algorithms
- ▶ SpGEMM, SpMV, Batched linear algebra in ARM PL

## AMD

- ▶ Just started the interactions on sparse, dense, batched linear algebra kernels, and sparse solvers
- ▶ Kokkos backend under-development
- ▶ Kokkos Kernels will be the performance test case

## Intel

- ▶ Compact API on KNL
- ▶ Kokkos backend under development
- ▶ Kokkos Kernels will be the performance test case

Kokkos Kernels team working with hardware vendors to support application needs on current **and** exascale platforms

**SPARC:** state-of-the-art hypersonic unsteady hybrid structured/unstructured finite volume CFD code

- ▶ **High performance line solvers; batched BLAS on CPUs and GPUs**
- ▶ **Performance-portable programming models**

**EMPIRE:** next-gen unstructured-mesh FEM PIC/multifluid plasma simulation code

- ▶ Scalable solvers for electrostatic and electromagnetic systems for Trinity and Sierra architectures
- ▶ **Thread-scalable, performance-portable, on-node linear algebra kernels to support multigrid methods**
- ▶ **Performance-portable programming models**
- ▶ Non-linear solvers, discretization, and automatic differentiation approaches

**Exawind:** next-gen wind simulation code

- ▶ **Scalable solvers for Trinity and Sierra architectures**
- ▶ **Thread-scalable, performance-portable, on-node linear algebra kernels to support multigrid methods**
- ▶ **Performance-portable programming models**

**QMCPACK:** Electronic structure code with Quantum Monte Carlo Algorithms

- ▶ Team level BLAS and LAPACK within the Kokkos ecosystem

Kokkos Kernels integrated into several applications in an agile manner at all stages from requirements solicitation, designing kernels and integration

## Dense Linear Algebra (BLAS and Batched BLAS)

- ▶ Motivation for BLAS/LAPACK functions
- ▶ Algorithm Specialization for Applications
- ▶ Calling BLAS/LAPACK functions

## Sparse Linear Algebra

- ▶ Sparse Containers (CrsMatrix, StaticCrsGraph, Vector)
- ▶ Sparse Matrix-Vector Multiplication (SpMV)
- ▶ Sparse Matrix-Matrix Addition (SpADD)
- ▶ Sparse Matrix-Matrix Multiplication (SpGEMM)

## Graph Kernels

- ▶ Distance-1 Graph Coloring
- ▶ Distance-2 Graph Coloring
- ▶ Bipartite Graph Partial Coloring

## Sparse Solvers

- ▶ Multicolor Gauss Seidel
- ▶ Cluster Gauss Seidel
- ▶ Two-Stage Gauss Seidel
- ▶ Sparse Incomplete LU Factorization (SpILUK)
- ▶ Sparse Triangular Solver (SpTRSV)

## Build System

- ▶ Using Kokkos Kernels in Your Project
- ▶ Configure, Build, and Install Kokkos Kernels
- ▶ Install with Spack

# BLAS and LAPACK

## Learning objectives:

- ▶ Motivation for BLAS/LAPACK functions
- ▶ Algorithm Specialization for Applications
- ▶ Calling BLAS/LAPACK functions

## KokkosKernels

- ▶ A single interface to vendor BLAS libraries on heterogenous computing platforms
- ▶ Support user-defined data type e.g., Automatic Differentiation, Ensemble, SIMD, types with Kokkos native implementation
- ▶ Customized performance solution for certain problem sizes
- ▶ Exploring new performance oriented interfaces

## Vendor Libraries

- ▶ A user needs to write a different function interface for different computing platforms e.g., MKL vs. CUBLAS
- ▶ Built-in real/complex data types and column/row major data layouts are only supported
- ▶ Code is highly optimized; in practice, higher performance is obtained from larger problem sizes

## Algorithm Specialization for Applications

- ▶ Dot-based GEMM
  - ▶ GEMM is used for orthogonalizing Krylov multi-vectors (long skinny matrix)
  - ▶ This particular problem shape does not perform well on CUBLAS
  - ▶ Algorithm is specialized for this shape performing multiple dot products instead of running standard GEMM algorithms
- ▶ Compact Batched BLAS
  - ▶ Application wants to solve many instances of tiny square block dense matrices; e.g., block dimensions of 3, 5, 7, 9, 11, etc.
  - ▶ Difficult to effectively use wide vector length such as AVX512 for this small problem size
  - ▶ A pack of block matrices are inter-leaved and solved simultaneously using vector instructions
  - ▶ Code is trivially vectorized 100% for the applied BLAS and LAPACK operations

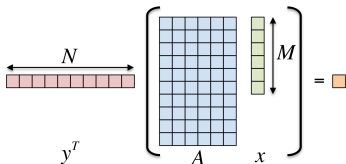
## Algorithm Specialization for Applications

- ▶ Extended Blas 1 interface: see axpby, update (a, c, b, y, g, z)
  - ▶  $y[i] = g * z[i] + b * y[i] + a * x[i]$
  - ▶ Trilinos Tpetra interface used in Belos iterative solvers
- ▶ See the wiki page for complete list of functions
  - ▶ <https://github.com/kokkos/kokkos-kernels/wiki>

KokkosKernels interacts with application teams and provides custom performance solutions for their needs

## Recall the Kokkos Inner Product exercise:

- ▶ Inner product  $\langle y, A * x \rangle$ 
  - ▶  $y$  is  $N \times 1$ ,  $A$  is  $N \times M$ ,  
 $x$  is  $M \times 1$
- ▶ Early exercise code looked like

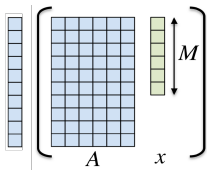


```
double result = 0;
Kokkos::parallel_reduce("yAz", N,
    KOKKOS_LAMBDA (int j, double &update) {
        double temp2 = 0;
        for (int i = 0; i < M; ++i) {
            temp2 += A(j, i) * x(i);
        }
        update += y(j) * temp2;
    }, result);
```

This can be naturally expressed as two BLAS operations:

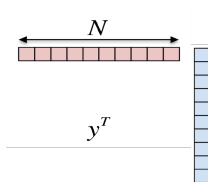
In Matlab notation:

```
// 1. gemv:
Ytmp = A * x
```



```
// 2. dot:
result = y' * Ytmp
```

□□



Different function signatures and APIs are used by different vendors  
e.g., on Cuda: cublasDgemv and cublasDdot

```
KokkosBlas::gemv (mode, alpha, A, x, beta, y);
```

**Interface:**

- ▶ mode [in]
  - ▶ "N" for non-transpose
  - ▶ "T" for transpose
  - ▶ "C" for conjugate transpose.
- ▶ alpha [in] Input coefficient of  $A*x$
- ▶ A [in] Input matrix, as a 2-D Kokkos::View
- ▶ x [in] Input vector, as a 1-D Kokkos::View
- ▶ beta [in] Input coefficient of  $y$
- ▶ y [in/out] Output vector, as a nonconst 1-D Kokkos::View

```
result = KokkosBlas::dot(x, y);
```

### Single Interface:

- ▶ x [in] Input vector, as a 1-D Kokkos::View
- ▶ y [in] Input vector, as a 1-D Kokkos::View
- ▶ result [out] Scalar result on host
- ▶ This interface calls Kokkos::fence on all execution spaces

```
KokkosBlas::dot(r, x, y);
```

### Single and Multi-vector Interface:

- ▶ x [in] Input (multi-)vector, as a 1-D or 2-D Kokkos::View
- ▶ y [in] Input (multi-)vector, as a 1-D or 2-D Kokkos::View
- ▶ r [in/out] Output result, as a rank-0 or 1-D Kokkos::View
- ▶ This interface is non-blocking.

**KokkosKernels:**

```
Kokkos::View<double*> tmp("tmp" , N);
KokkosBlas::gemv("N" , alpha ,A,x ,beta ,
  tmp);

double result = 0;

result = KokkosBlas::dot(y,tmp);
```

- ▶ Uses two BLAS functions
- ▶ Optionally interface to optimized vendor libraries
- ▶ For certain matrix shapes may choose specialized code path for performance

**User implementation:**

```
double result = 0;
Kokkos::parallel_reduce("yAx" , N,
  KOKKOS_LAMBDA (int j, double &
    update) {
  double temp2 = 0;
  for (int i = 0; i < M; ++i) {
    temp2 += A(j, i) * x(i);
  }
  update += y(j) * temp2;
}, result);
```

- ▶ Exploits a single level of parallelism only i.e., internal temp2 is summed sequentially
- ▶ Matrix-vector multiplication and dot product are fused in a single kernel

Related exercise available at: [Exercises/kokkoskernels/InnerProduct](#)

## Summary: BLAS/LAPACK

- ▶ Single interface for heterogeneous computing platforms
- ▶ Optimized vendor library interface when it is available
- ▶ Specialization of algorithms corresponding to application needs
- ▶ Native implementation supports strided data layout of a matrix

# Batched BLAS and LAPACK

## Learning objectives:

- ▶ Motivation for batched functions
- ▶ Two namespaces with BLAS and LAPACK functions
- ▶ Calling batched functions

Batched BLAS/LAPACK is **simple** i.e., BLAS/LAPACK in a parallel loop

```
auto A = Kokkos::View<double***>('A', N, B1k, B1k);
Kokkos::parallel_for( RangePolicy(N), /// users' parallel execution policy
  KOKKOS_LAMBDA(int &i) {
  auto AA = Kokkos::subview(A, i, ALL, ALL);
  KokkosBatched::SerialLU(AA); /// functor-level interface
});
```

Kokkos batched BLAS/LAPACK is made up of following two components

- ▶ Kokkos parallel execution policy with `parallel_for`
- ▶ A functor-level interface to be used in `operator()`

Hierarchical functor interface is required matching to Kokkos' hierarchical parallelism

## Serial Interface

- ▶ can be used in a flat `parallel_for` i.e., `Kokkos::RangePolicy`
- ▶ can be used in the most inner loop of nested `parallel_for`'s

### Serial with RangePolicy

```
parallel_for(RangePolicy ,  
            KOKKOS_LAMBDA(int &idx){  
                KokkosBatched::SerialDoThing();  
            });
```

### Serial in Hierarchical parallel loops

```
parallel_for(TeamPolicy ,  
            KOKKOS_LAMBDA(member_type &member){  
                parallel_for(TeamThreadRange) {  
                    parallel_for(ThreadVectorRange) {  
                        KokkosBatched::  
                            SerialDoSomething();  
                    };  
                };  
            });
```

## TeamVector Interface

- ▶ internally uses two nested `parallel_for` with `TeamThreadRange` and `ThreadVectorRange`
- ▶ requires the member (thread communicator) as an input argument

### TeamVector with TeamPolicy

```
parallel_for(TeamPolicy,  
            KOKKOS_LAMBDA(member_type &member){  
                KokkosBatched::TeamVectorDoSomething(member);  
            });
```

## Team Interface

- ▶ internally use TeamThreadRange only
- ▶ in general is used with SIMD or Ensemble types where vector parallelism is expressed within the type
- ▶ can include ThreadVectorRange

### Team without ThreadVectorRange

```
parallel_for (TeamPolicy ,  
             KOKKOS_LAMBDA(member_type &member){  
             KokkosBatched :: TeamDoThing(member);  
             });
```

### Team with ThreadVectorRange outside

```
parallel_for (TeamPolicy ,  
             KOKKOS_LAMBDA(member_type &member){  
             parallel_for (ThreadVectorRange) {  
             KokkosBatched :: TeamDoSomething(  
             member);  
             }); });
```

Consider a batched **block matrix inversion** which can be used for  
a block Jacobi solver.

## KokkosKernels

```
using ViewTypeAs = Kokkos::View<double***>;

using ScratchSpaceView = Kokkos::View<double*,
    Kokkos::DefaultExecutionSpace::scratch_memory_space,
    Kokkos::MemoryTraits<Kokkos::Unmanaged>>;
```

## KokkosKernels

```
ViewTypeAs As("As", N, Blk, Blk);
Kokkos::parallel_for(TeamPolicy,
  KOKKOS_LAMBDA(member_type &member) {
    auto A = Kokkos::subview(As, i, ALL, ALL);
    auto T = ScratchSpaceView(member, Blk, Blk);
    KokkosBatched::TeamVectorLU(member, A);
    KokkosBatched::TeamVectorCopy(member, T, A);
    KokkosBatched::TeamVectorSetIdentity(member, A);
    KokkosBatched::TeamVectorLowerTrsm(member, T, A);
    KokkosBatched::TeamVectorUpperTrsm(member, T, A);
  });
```

- ▶ Multiple BLAS/LAPACK operations can be fused in a single kernel
- ▶ Temporal locality via single kernel launch
- ▶ Local cache memory can be used as scratch space
- ▶ Team size can be tuned for problem
- ▶ Poor performance when poorly tuned

## Vendor Libraries

```
As = Kokkos::View<double***>("As", N, Blk, Blk);  
Ts = Kokkos::View<double***>("Ts", N, Blk, Blk);  
batch_parallel_lu(As);  
batch_parallel_copy(Ts, As);  
batch_parallel_set_identity(As);  
batch_parallel_lower_trsm(Ts, As);  
batch_parallel_upper_trsm(Ts, As);  
  
/// or if you are lucky to find an inversion routine,  
batch_parallel_invert(As, Ts);
```

- ▶ Each batched kernel is highly optimized
- ▶ In a sequence of batch operations, the workflow can be suboptimal
- ▶ Multiple kernel launches can cause increased latency cost and more memory traffic

## KokkosBlas namespace

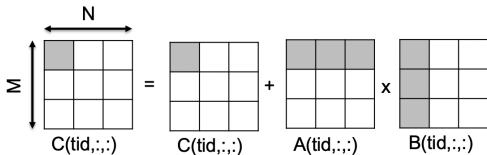
- ▶ **KokkosBlas:** device-level functions with optional TPL support
  - ▶ *Intended Use Case:*
    - ▶ Caller uses the entire device execution space for solving a single dense problem
    - ▶ For performance, the problem should be large enough to exploit the entire device
  - ▶ *Blocking behavior:*
    - ▶ On GPUs, non-blocking by default with some exceptions of norms where the result is requested from host

## KokkosBatched namespace

- ▶ **KokkosBatched:** functor level functions
  - ▶ *Intended Use Case:*
    - ▶ Caller is within parallel kernel body with a batch of input vectors
  - ▶ *Multiple Interfaces: Serial, Team, TeamVector*
    - ▶ Serial: no nested parallelism is used internally
    - ▶ Team: one-level nested parallelism is used with *TeamThreadRange*
    - ▶ TeamVector: two-level nested parallelism is used with *TeamThreadRange* and *TeamVectorRange*

# Exercise: TeamGemm

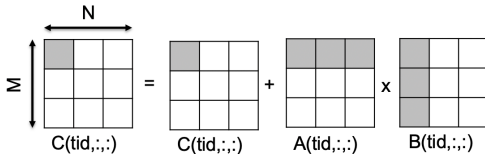
- ▶ Recall Kokkos nested parallelism
- ▶ Exercise:  $C = \beta * C + \alpha * A * B$ 
  - ▶  $C$  is  $P \times M \times N$
  - ▶  $A$  is  $P \times M \times K$
  - ▶  $B$  is  $P \times K \times N$
  - ▶  $\beta$  and  $\alpha$  are scalars



```

Kokkos::parallel_for("teamGemmOuter",
  Kokkos::TeamPolicy<ExecutionSpace>(nTeam, teamSize),
  KOKKOS_LAMBDA (const member_type &member) {
    const int tid = member.league_rank();
    // Each team performs a single TeamGemm
    Kokkos::parallel_for("teamGemmInner",
      Kokkos::TeamThreadRange(member, thisTeamsRangeSize),
      [=] (const unsigned int ij) {
        const int i = ij/N, j = ij%N;
        // each thread computes C(tid,i,j)
      });
  });
};

```

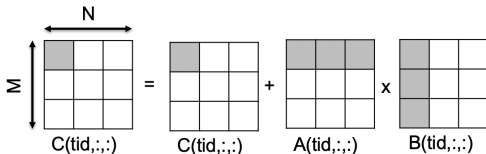


## This can be naturally expressed using the TeamGemm interface

```

Kokkos::parallel_for("teamGemmOuter",
  Kokkos::TeamPolicy<ExecutionSpace>(nTeams, teamSize),
  KOKKOS_LAMBDA (const member_type &member) {
    const int tid = member.league_rank();
    auto a = Kokkos::subview(A, tid, ALL(), ALL());
    auto b = Kokkos::subview(B, tid, ALL(), ALL());
    auto c = Kokkos::subview(C, tid, ALL(), ALL());
    KokkosBatched::TeamGemm(member,  $\alpha$ , a, b,  $\beta$ , c);
  });

```



Related exercise available at: [Exercises/kokkoskernels/TeamGemm](https://github.com/kokkos/exercises/blob/master/kokkoskernels/TeamGemm)

# Exercise: BlockJacobi

- ▶ Objective:
  - ▶ Compose a batched LU factorization of diagonal blocks and compute inverse of the blocks
  - ▶ Compare a non-fused batched functions against the fused batch function using functor level interface
- ▶ Exercise:  
<https://github.com/kokkos/kokkos-tutorials/tree/main/Exercises/kokkoskernels/BlockJacobi/Begin>
- ▶ On GPUs,
  - ▶ Test the code with different team size  
`run-different-teamsize.sh`
  - ▶ Profile the code using `nvprof` `run-nvprof.sh`

▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-different-teamsize.sh](#)

▶ This inverts 16,384 instances of 5x5 block matrices

TeamSize	# of inversion per sec	
	Non-fused	Fused
AUTO	3,385	5,054
32	4,603	<b>8,766</b>
64	4,199	6,488
96	3,581	5,017

▶ Why 32 TeamSize is the best ?

- ▶ For simplicity, assuming 25 entries of a block matrix are updated independently, 25 is the maximum team size
- ▶ By fusing multiple operations, temporal locality is exploited
- ▶ Need to check this using a profiler, nvprof

▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-nvprof.sh](#)

- ▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-different-teamsize.sh](#)

- ▶ This inverts 16,384 instances of 5x5 block matrices

TeamSize	# of inversion per sec	
	Non-fused	Fused
AUTO	3,385	<b>5,054</b>
32	4,603	<b>8,766</b>
64	4,199	6,488
96	3,581	5,017

- ▶ Why 32 TeamSize is the best ?
  - ▶ For simplicity, assuming 25 entries of a block matrix are updated independently, 25 is the maximum team size
  - ▶ By fusing multiple operations, temporal locality is exploited
  - ▶ Need to check this using a profiler, nvprof
- ▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-nvprof.sh](#)
  - ▶ Comparison 1, AUTO vs 32

▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-different-teamsize.sh](#)

▶ This inverts 16,384 instances of 5x5 block matrices

TeamSize	# of inversion per sec	
	Non-fused	Fused
AUTO	3,385	5,054
32	<b>4,603</b>	<b>8,766</b>
64	4,199	6,488
96	3,581	5,017

▶ Why 32 TeamSize is the best ?

- ▶ For simplicity, assuming 25 entries of a block matrix are updated independently, 25 is the maximum team size
- ▶ By fusing multiple operations, temporal locality is exploited
- ▶ Need to check this using a profiler, nvprof

▶ [Exercises/kokkoskernels/BlockJacobi/Solution/run-nvprof.sh](#)

- ▶ Comparison 2, non-fused vs fused

## Comparison 1: the same code with different team size

- ▶ AUTO (set TeamSize = 96) shows higher occupancy

```

Achieved Occupancy      0.537612
Multiprocessor Activity  96.95%
Warp Execution Efficiency 65.83%
L2 Cache Utilization    Low (1)
Global Load Transactions 2186086
Global Store Transactions 1622016
Global Load Throughput   350.22GB/s
Global Store Throughput  259.86GB/s
L2 Throughput (Reads)    38.001GB/s
L2 Throughput (Writes)  261.43GB/s
Global Memory Load Efficiency 52.28%
Global Memory Store Efficiency 54.29%

```

- ▶ TeamSize = 32 leads higher global load/store throughput, resulting 1.7x speedup

```

Achieved Occupancy      0.428055
Multiprocessor Activity  90.10%
Warp Execution Efficiency 44.88%
L2 Cache Utilization    Low (1)
Global Load Transactions 765594
Global Store Transactions 417792
Global Load Throughput   493.55GB/s
Global Store Throughput  269.34GB/s
L2 Throughput (Reads)    161.18GB/s
L2 Throughput (Writes)  269.52GB/s
Global Memory Load Efficiency 58.85%
Global Memory Store Efficiency 73.53%

```

## Comparison 2: the same code with non-fused vs fused version

- ▶ For non-fused version, we show one best performing kernel of four kernels

```

Achieved Occupancy      0.457975
Multiprocessor Activity  95.12%
Warp Execution Efficiency 46.64%
  L2 Cache Utilization  Low (2)
Global Load Transactions 2184714
Global Store Transactions 1622016
  Global Load Throughput 643.68GB/s
  Global Store Throughput 477.89GB/s
  L2 Throughput (Reads)  83.494GB/s
  L2 Throughput (Writes) 486.68GB/s
Global Memory Load Efficiency 52.31%
Global Memory Store Efficiency 54.29%

```

- ▶ Fused version performs 1.9x faster than non-fused version

```

Achieved Occupancy      0.428055
Multiprocessor Activity  90.10%
Warp Execution Efficiency 44.88%
  L2 Cache Utilization  Low (1)
Global Load Transactions 765594
Global Store Transactions 417792
  Global Load Throughput 493.55GB/s
  Global Store Throughput 269.34GB/s
  L2 Throughput (Reads)  161.18GB/s
  L2 Throughput (Writes) 269.52GB/s
Global Memory Load Efficiency 58.85%
Global Memory Store Efficiency 73.53%

```

- ▶ Note that non-fused interface can be optimized much better for each kernel and specific problem size

## Summary: Batched BLAS/LAPACK

- ▶ User composable batched interface: parallel execution policy + functor-level interface
- ▶ Performance on GPUs is tunable:
  - ▶ Launching light-weight kernels multiple times can cause overhead
  - ▶ Fusing too many functor-level BLAS/LAPACK operations is difficult to perform optimal with a single team size

# Sparse Linear Algebra

Sparse linear algebra data structures and functions.

## Learning objectives:

- ▶ Key characteristics algorithms
- ▶ Containers: CrsMatrix, StaticCrsGraph, Vector
- ▶ SpMV
- ▶ SpADD
- ▶ SpGEMM

## Support for important class of applications

- ▶ Representation of choices for discrete PDE systems (FEM, FD, CVFEM, ...)
- ▶ Natural use for network representation
  - ▶ Electrical grid, electronic circuit
  - ▶ Social network

## Unique format supported: Compressed row sparse

Sparse matrices can be stored in various format, currently only Crs format is fully supported, BlockCrs is partially supported

## Constraints from Crs format

- ▶ hard to optimize memory access patterns
- ▶ often multi-pass algorithms required
  1. compute storage
  2. compute column index and actual values
- ▶ typically algorithms can be split in symbolic and numeric phases

## Symbolic/Numeric split

While extremely useful for reuse it is potentially slower for single use case depending on implementation

## Handle: hiding important details!

- ▶ What the handles does for you:
  - ▶ stores user parameters
  - ▶ keeps temporary data needed in numeric of solve/apply phases
  - ▶ cleans up temporary data at destruction
  - ▶ contains kernel specific "sub-handle"
  - ▶ specifies required data types
- ▶ Usage: `KokkosKernels::Experimental::KokkosKernelsHandle<size_type, index_type, scalar_type, ExecutionSpace, TempMemSpace, PermMemSpace>()`

**One dense structure:**

- ▶ View (of rank 1): represents a "vector"
- ▶ View (of rank 2): represents a "multi-vector"

**Two sparse structures:**

- ▶ StaticCrsGraph: encodes the sparsity pattern in `row_map` and `entries`
- ▶ CrsMatrix: contains a `StaticCrsGraph` and values

**Example:**

`example/wiki/sparse/KokkosSparse_wiki_crsmatrix.cpp`

## Two interfaces for one kernel?

### 1. Simplified interface

- ▶ uses high level containers
- ▶ reduced number of parameters and templates
- ▶ allocates memory

### 2. Expert interface

- ▶ uses low level container (i.e. views)
- ▶ allows for finer memory management

## Simplified/Expert interface

For clarity we will focus on the simplified interface in the rest of the lecture

## SpMV: a mixed sparse/dense kernel

$$0.5 * \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + 1.0 \begin{bmatrix} 1 & & 2 \\ & 3 & \\ 4 & & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 8.5 \\ 22 \end{bmatrix}$$

- ▶ Computes:  $y = \beta * y + \alpha * A * x$
- ▶ Output is a dense vector
  - ▶ single pass algorithm since no CrsGraph needs to be computed
  - ▶ good amount of parallelism exploitable
- ▶ Usage:
 

```
KokkosSparse::spmv(mode, alpha, A, x, beta, y);
```
- ▶ Example:
 

```
example/wiki/sparse/KokkosSparse_wiki_spmv.cpp
```

## SpADD: Sparse Matrix Addition

$$2.0 \begin{bmatrix} 1 & & 2 \\ & 3 & 4 \\ 5 & & \end{bmatrix} + 0.5 \begin{bmatrix} 6 & 7 & \\ & 8 & \\ & & 9 \end{bmatrix} = \begin{bmatrix} 5 & 3.5 & 4 \\ & 10 & 8 \\ 10 & & 4.5 \end{bmatrix}$$

- ▶ Computes:  $C = \alpha A + \beta B$  given  $A$  and  $B$  two CrsMatrices
- ▶ Sorted inputs speeds-up the kernel and reduces memory consumption

- ▶ Usage:

```
KokkosSparse::spadd_symbolic(handle, A, B, C);  
KokkosSparse::spadd_numeric(handle, alpha, A,  
beta, B, C);
```

- ▶ Example:

```
example/wiki/sparse/KokkosSparse_wiki_spadd.cpp
```

## SpGEMM: Sparse General Matrix Matrix Multiply

- ▶ Compute  $A \times B = C$  for given sparse matrices  $A$  and  $B$

$$\begin{bmatrix} 1 & & 2 \\ & 3 & 4 \\ 5 & & \end{bmatrix} \times \begin{bmatrix} 6 & & 7 \\ 8 & 9 & \\ & & 10 & 11 \end{bmatrix} = \begin{bmatrix} 6 & & 27 & 22 \\ 24 & 27 & 40 & 41 \\ 30 & 35 & & \end{bmatrix}$$

- ▶ Sparsity structure of  $C$  is not known in advance!
- ▶ We have a two-phase implementation:
  - ▶ This allows determining the sparsity of  $C$  efficiently

## SpGEMM: Sparse General Matrix Matrix Multiply

### ▶ Symbolic phase:

- ▶ `KokkosSparse::spgemm_symbolic(handle, A, isTrnspsdA, B, isTrnspsdB, C);`
- ▶ determines number of nonzeros in each row of  $C$  and
- ▶ allocates memory for column indices and values of the nonzeros

### ▶ Numeric phase

- ▶ `KokkosSparse::spgemm_numeric(handle, A, isTrnspsdA, B, isTrnspsdB, C);`
- ▶ computes column indices and values of the nonzeros of  $C$

### ▶ Example

`example/wiki/sparse/KokkosSparse_wiki_spgemm.cpp`

## SpGEMM: Sparse General Matrix Matrix Multiply

- ▶ We follow Gustavson's algorithm:
  - for** each row index  $i \leftarrow 0$  **to**  $nrowsA$  **do**
  - for** each column index  $j \in A(i, :)$  **do**
  - //accumulate partial row results
  - $C(i, :) \leftarrow C(i, :) + A(i, j)B(j, :)$
- ▶ Our implementation exploits hierarchical paralelism
  - ▶ Teams are assigned contiguous row chunks in  $A$
  - ▶ Threads are assigned individual rows of  $A$
  - ▶ Vector lanes are assigned the nonzeros of rows of  $B$

## SpGEMM: Sparse General Matrix Matrix Multiply

- ▶ We follow Gustavson's algorithm:
  - for** each row index  $i \leftarrow 0$  **to**  $nrowsA$  **do**
    - for** each column index  $j \in A(i, :)$  **do**
      - //accumulate partial row results
      - $C(i, :) \leftarrow C(i, :) + A(i, j)B(j, :)$
- ▶ Our thread-scalable hashmap accumulator implementation
  - ▶ is used in both symbolic and numeric phases
  - ▶ supports both sparse and dense accumulators
  - ▶ has a two-level structure: Level-1 ( $L_1$ ) and Level-2 ( $L_2$ )
    - ▶  $L_1$  hashmap lives in the fast shared memory
    - ▶  $L_2$  hashmap is created only if  $L_1$  hashmap runs out of memory
    - ▶  $L_2$  hashmap lives in the large global memory
- ▶ For more details see: M. Deveci, C. Trott, S. Rajamanickam, "Multithreaded sparse matrix-matrix multiplication for many-core and GPU architectures", *Parallel Computing* 78, 33-46, 2018.

## Summary: Sparse Linear Algebra

- ▶ Main difficulties: finding sparsity patterns and memory access
- ▶ Containers: `View`, `StaticCrsGraph` and `CrsMatrix`
- ▶ Kernels: `SpMV`, `SpADD` and `SpGEMM`

# Graph Kernels

Kokkos Kernels functionality for graph computations.

## Learning objectives:

- ▶ Distance-1 Graph Coloring
- ▶ Distance-2 Graph Coloring
- ▶ Bipartite Graph Partial Coloring

## Distance-1 Graph Coloring

- ▶ Given a graph, assign a color to each vertex so that no two adjacent vertices have the same color
- ▶ Minimizing the number of unique colors is NP-hard
- ▶ Approximate solution (with a few more colors than optimal) is still useful
- ▶ KokkosKernels has two main algorithms for this: vertex-based and edge-based

## Vertex-Based (VB) Coloring

Initialize worklist containing every vertex.

- ▶ In parallel, for each vertex  $v$  in worklist:
  - ▶ Assign smallest color to  $v$  which isn't found on any neighbor
- ▶ In parallel, for each vertex  $v$  in worklist:
  - ▶ If  $v$ 's color matches with a neighbor, uncolor  $v$  and add it to next worklist

These steps are repeated until the worklist is empty (all vertices have been colored).

## Edge-Based (EB) Coloring

Initialize worklist containing every edge.

- ▶ In parallel, for each edge  $e$  in worklist:
  - ▶ If both endpoints of  $e$  have the same color, uncolor the one with a higher ID
  - ▶ If at least one endpoint of  $e$  is uncolored, add  $e$  to the next worklist.
- ▶ In parallel, for each edge  $e$  in worklist:
  - ▶ If exactly one endpoint is colored, add that color to forbidden set for other endpoint
- ▶ In parallel, for each uncolored vertex  $v$ :
  - ▶ Color  $v$  with smallest non-forbidden color

These steps are repeated until the edge worklist is empty, meaning both endpoints of every edge have been colored.

## Algorithm Summary

- ▶ EB pseudocode was simplified, did not include tentative coloring (technique for faster convergence)
- ▶ In VB, work per thread requires loop over neighbors of a vertex
- ▶ In EB, work per thread is constant time, but the worklists are longer
- ▶ EB is significantly faster on GPUs when the maximum degree is high (generally,  $> 3000$ )
- ▶ Otherwise, VBBIT (VB with bitwise operations to track forbidden colors) is usually the fastest.
- ▶ Use enum values `KokkosGraph::COLORING_VBBIT` and `KokkosGraph::COLORING_EB`

## Using Distance-1 Coloring

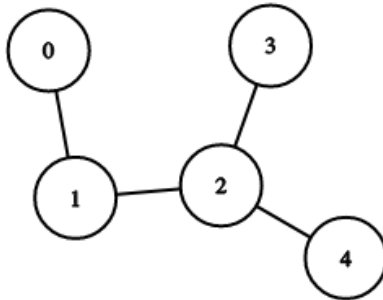
```
#include "KokkosGraph_Distance1Color.hpp"
KokkosKernels::KokkosKernelsHandle<...> handle;
// Choose algorithm and set up
handle.create_graph_coloring_handle(KokkosGraph::COLORING_VB);
// Compute the coloring
KokkosGraph::Experimental::graph_color(&handle,
    numVertices, numVertices, rowmap, entries);
// Get the subhandle for coloring
auto colorHandle = handle.get_graph_coloring_handle();
// Get the number of colors used, and color labels
auto numColors = colorHandle->get_num_colors();
auto colors = colorHandle->get_vertex_colors();
// Clean up
handle.destroy_graph_coloring_handle();
```

## Distance-2 Coloring Problem

- ▶ Each vertex must have a different color than all vertices within 2 hops of it
- ▶ If  $G$  is represented by adjacency matrix, this is equivalent to computing distance-1 coloring on  $G^2$
- ▶ Graph must be undirected (symmetric adjacency matrix)

## Distance-2 Coloring Problem

In this graph, 0 couldn't have the same color as 1 or 2, but it could have the same as 3 or 4.

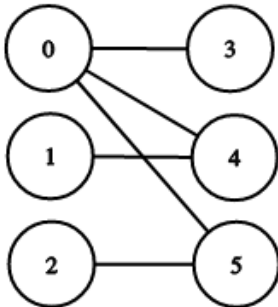


## Bipartite Graph Partial Coloring

- ▶ Closely related to distance-2 coloring
- ▶ Color either left or right side of a bipartite graph so that any vertices 2 hops apart have different colors
- ▶ Left-side BGPC equivalent to distance-1 coloring on  $GG^T$
- ▶ Right-side BGPC equivalent to distance-1 coloring on  $G^T G$

## Bipartite Graph Partial Coloring

- ▶ For left-sided coloring of this graph, 1 couldn't have the same color as 0, but could have the same as 2.
- ▶ For right-sided coloring of this graph, vertices 3, 4 and 5 must all have different colors.



## D2/BGPC Algorithms

- ▶ VB (`KokkosGraph::COLORING_D2_VB_BIT`): Just like distance-1 VB, but coloring and conflict resolution loop over neighbors-of-neighbors, not just neighbors
- ▶ NB (`KokkosGraph::COLORING_D2_NB_BIT`) Net-based coloring from “Greed is Good: Parallel Algorithms for BGPC” by Taş et al. Is asymptotically faster than VB by avoiding neighbors-of-neighbors loops, and is faster in practice.

## Using Distance-2 Coloring

```
#include "KokkosGraph_Distance2Color.hpp"
KokkosKernels::KokkosKernelsHandle<...> handle;
// Set up for coloring, and choose algorithm
handle.create_distance2_graph_coloring_handle(
    KokkosGraph::COLORING_D2_NB_BIT);
// Compute the coloring
KokkosGraph::Experimental::graph_color_distance2(
    &handle, numVertices, rowmap, entries);
// Get the subhandle for D2 coloring
auto colorHandle =
    handle.get_distance2_graph_coloring_handle();
auto numColors = colorHandle->get_num_colors();
auto colors = colorHandle->get_vertex_colors();
handle.destroy_distance2_graph_coloring_handle();
```

## Using BGPC

Same handle and algorithm choices as D2, but use:

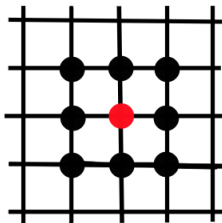
```
KokkosGraph::Experimental::bipartite_color_rows(  
    &handle, numRows, numColumns, rowmap, entries);
```

and:

```
KokkosGraph::Experimental::bipartite_color_columns(  
    &handle, numRows, numColumns, rowmap, entries);
```

## Coloring Exercise

- ▶ `Intro-Full/Exercises/kokkoskernels/GraphColoring`
- ▶ Compute both D1 and D2 colorings of a graph
- ▶ The graph is generated as a 9-point stencil on a small 2D grid
- ▶ The colors will be printed out in the layout of the grid



**Figure:** A 9-point stencil. The black points are adjacent to the red point.

## Summary: Graph Algorithms

- ▶ Distance-1 Coloring
  - ▶ vertex-based (VB) and edge-based (EB) based algorithms
  - ▶ Use COLORING\_VBBIT, unless maximum degree  $> 3000$  - then use COLORING\_EB
- ▶ Distance-2 and Bipartite Graph Partial Coloring
  - ▶ vertex-based (VB) and net-based (NB) algorithms
  - ▶ Use COLORING\_D2\_NB\_BIT for best performance

# Sparse Solvers

## Gauss-Seidel Preconditioners

### **Learning objectives:**

- ▶ Multicolor Gauss-Seidel
- ▶ Cluster Gauss-Seidel
- ▶ Two-Stage Gauss-Seidel

## Multicolor Gauss-Seidel

Gauss-Seidel (GS) method for solving  $A\mathbf{x} = \mathbf{b}$  updates one entry of the unknown at a time:

For  $i = 1..M$ :

$$\mathbf{x}_i := (\mathbf{b}_i - \sum_{j=1}^N A_{ij}\mathbf{x}_j) / A_{ii}$$

- ▶ Standard GS is sequential: updates to  $\mathbf{x}_i$  are affected by previous updates to  $\mathbf{x}_j$  in the same iteration (where  $j < i$ )
- ▶ Treating  $A$  as a graph's adjacency matrix,  $A_{ij} \neq 0$  if vertices  $i$  and  $j$  are adjacent
- ▶ Suppose a coloring is computed for this graph, and  $Color(i) = Color(j)$ .
- ▶ then  $\mathbf{x}_j$  does not directly affect the updated value of  $\mathbf{x}_i$

## Using KokkosKernels Multicolor GS

KokkosKernels supports preconditioning with multicolor GS. Rows with the same color are updated in parallel.

```
#include "KokkosSparse_gauss_seidel.hpp"
// Handle creation
KokkosKernels::KokkosKernelsHandle<...> handle;
handle.create_gs_handle(KokkosSparse::GS_POINT);
// Symbolic setup
KokkosSparse::Experimental::gauss_seidel_symbolic(
    &handle, numRows, numCols,
    A.graph.row_map, A.graph.entries, graphIsSymmetric);
// Numeric setup
KokkosSparse::Experimental::gauss_seidel_numeric(
    &handle, numRows, numCols,
    A.graph.row_map, A.graph.entries, A.values,
    graphIsSymmetric);
```

## Using KokkosKernels Multicolor GS, continued

KokkosKernels supports parallel preconditioning with multicolor GS.

```
KokkosSparse::Experimental::forward_sweep_gauss_seidel_apply(  
    &handle, numRows, numCols,  
    A.graph.row_map, A.graph.entries, A.values,  
    x, b, initZeroX, updateCachedB, omega, numSweeps);  
// --- or ---  
KokkosSparse::Experimental::backward_sweep_gauss_seidel_apply(  
    &handle, numRows, numCols,  
    A.graph.row_map, A.graph.entries, A.values,  
    x, b, initZeroX, updateCachedB, omega, numSweeps);  
// --- or ---  
KokkosSparse::Experimental::symmetric_gauss_seidel_apply(  
    &handle, numRows, numCols,  
    A.graph.row_map, A.graph.entries, A.values,  
    x, b, initZeroX, updateCachedB, omega, numSweeps);  
// Clean up  
handle.destroy_gs_handle();
```

## Using KokkosKernels Multicolor GS

- ▶ Algorithm called POINT because individual rows of the matrix are colored, as opposed to blocks/clusters
- ▶ `graphIsSymmetric`: whether the matrix is structurally symmetric. If false, must symmetrize before coloring.
- ▶ `initZeroX`: whether to zero out  $\mathbf{x}$  before starting
- ▶ `updateCachedB`: whether on the first apply, or  $\mathbf{b}$  has changed since the last apply
- ▶ `omega`: damping factor for successive over-relaxation (default is 1.0)
- ▶ `numSweeps`: how many applications to perform. For symmetric apply, forward+back counts as 1 application.

## Cluster GS

- ▶ In Multicolor GS, an independent row  $j$  does not *directly* affect the updated value of  $\mathbf{x}_i$ , but it can affect it *indirectly*.
- ▶ For example, if  $i$  and  $j$  have the same color and are separated by  $k$ , then information is not transferred from  $\mathbf{x}_i$  to  $\mathbf{x}_j$  through  $\mathbf{x}_k$  within a sweep.
- ▶ This is why multicolor GS usually gives a slightly worse answer than sequential GS.
- ▶ To help with this, cluster GS coarsens the graph and applies GS sequentially within a cluster.

## Cluster GS Example:

```
handle.create_gs_handle(  
    KokkosSparse::CLUSTER_BALLOON, clusterSize);
```

- ▶ “Balloon” is the coarsening algorithm (others may be added in the future)
- ▶ `clusterSize` is the coarsening factor (an integer larger than 1, but should be small compared to the number of rows)
- ▶ The symbolic, numeric and apply interface is the same as multicolor (POINT)

## Two-Stage GS

- ▶ Hybrid of the Jacobi and Gauss-Seidel methods
- ▶ Formulates Gauss-Seidel as a lower or upper triangular solve (for forward and backward sweeps, respectively), and uses some number of Jacobi sweeps as an approximation for this solve.

Usage:

```
handle.create_gs_handle(KokkosSparse::TWO_STAGE);
```

## GS: Exercise

- ▶ `Intro-Full/Exercises/kokkoskernels/GaussSeidel`
- ▶ Generates a small, diagonally dominant system
- ▶ Fill in the necessary calls to set up and use one of the GS algorithms as an iterative solver

## Summary: Gauss-Seidel

- ▶ Multicolor Gauss-Seidel
  - ▶ Uses coloring to find independent rows
- ▶ Cluster Gauss-Seidel
  - ▶ Like multicolor but coarsens graph first
- ▶ Two-Stage Gauss-Seidel
  - ▶ Hybrid Gauss-Seidel/Jacobi-Richardson

The best choice depends on the problem.

# Sparse Solvers 2

Sparse factorization and triangular solver.

## Learning objectives:

- ▶ Sparse incomplete LU factorization
- ▶ Sparse triangular solvers

## SPARSE SPILUK and SPTRSV

KokkosKernels supports preconditioning with sparse incomplete LU factorization coupled with sparse triangular solvers.

- ▶ **SPILUK**: Sparse k-level incomplete LU factorization
  - ▶ Computes sparse lower triangular matrix  $L$  and upper triangular matrix  $U$  such that  $M = LU$  is "similar" to  $A$
  - ▶  $k = 0$ : No additional fill-in.  $G(L + U) = G(A)$
  - ▶  $k > 0$ : Increased fill level improves accuracy
- ▶ **SPTRSV**: Sparse triangular solver
  - ▶ Apply ILU:  $\mathbf{z} = M^{-1}\mathbf{r} \Leftrightarrow \mathbf{z} = (LU)^{-1}\mathbf{r} \Leftrightarrow \mathbf{z} = U^{-1}(L^{-1}\mathbf{r})$
  - ▶ L,U reused by triangular solver to apply preconditioning during linear solver iterations

## SPILUK usage

- ▶ ILU(k): requires matrices in "Crs" format
- ▶ Symbolic phase on host (serial):
  - ▶ Construct nonzero patterns of L and U
  - ▶ Perform level-scheduling to group independent rows into levels based on L's sparsity pattern. Level-scheduling results stored within a handle for reuse
- ▶ Numeric phase (parallel) fill data to the nonzero patterns based on level-scheduling results found in the symbolic phase
- ▶ Algorithm options:
  - ▶ SEQLVLSCHD\_RP: using range policy parallelism for numeric phase
  - ▶ SEQLVLSCHD\_TP1: using team policy parallelism for numeric phase

## SPILUK: Interface

- ▶  $\{A,L,U\}$ \_rowmap: Arrays storing row pointer offset, as a 1-D Kokkos::View
- ▶  $\{A,L,U\}$ \_entries: Arrays storing column indices, as a 1-D Kokkos::View
- ▶  $\{A,L,U\}$ \_values: Arrays storing corresponding matrix values, as a 1-D Kokkos::View
- ▶ Handle: Stores internal data structures from symbolic phase
  - ▶ Input: SPILUKAlgorithm, number of rows, est. number of nonzeros L, est. number of nonzeros of U
  - ▶ Templated on rowmap data type (size\_type), entries ordinal type (lno\_t), values scalar type (scalar\_t), execution space, "persistent" memory space, "temporary" memory space (unused here)

## SPILUK: Interface

## SPILUK: Interface

- ▶ Include header file

```
#include "KokkosSparse_spiluk.hpp"  
//SPILUK in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

## SPILUK: Interface

- ▶ Include header file

```
#include "KokkosSparse_spiluk.hpp"  
//SPILUK in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_type, lno_t, scalar_t, exec_space, mem_space, mem_space> kh;
```

## SPILUK: Interface

- ▶ Include header file

```
#include "KokkosSparse_spiluk.hpp"  
//SPILUK in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_type, lno_t, scalar_t, exec_space, mem_space, mem_space> kh;
```

- ▶ Create the spiluk handle - requires estimate for nnz of L, U

```
nnzL = nnzU = EXPAND_FACT*A.nnz()*(fill_lev+1); // EXPAND_FACT set by user  
kh.create_spiluk_handle(SPILUKAlgorithm, nrows, nnzL, nnzU);
```

## SPILUK: Interface

- ▶ Include header file

```
#include "KokkosSparse_spiluk.hpp"
//SPILUK in Experimental namespace — interface may evolve
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle
<size_type, lno_t, scalar_t, exec_space, mem_space, mem_space> kh;
```

- ▶ Create the spiluk handle - requires estimate for nnz of L, U

```
nnzL = nnzU = EXPAND_FACT*A.nnz()*(fill_level+1); // EXPAND_FACT set by user
kh.create_spiluk_handle(SPILUKAlgorithm, nrows, nnzL, nnzU);
```

- ▶ Call symbolic routine

```
spiluk_symbolic(&kh, fill_level, A_rowmap, A_entries,
                L_rowmap, L_entries, U_rowmap, U_entries);
```

## SPILUK: Interface

- ▶ Include header file

```
#include "KokkosSparse_spiluk.hpp"
//SPILUK in Experimental namespace — interface may evolve
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle
<size_type, lno_t, scalar_t, exec_space, mem_space, mem_space> kh;
```

- ▶ Create the spiluk handle - requires estimate for nnz of L, U

```
nnzL = nnzU = EXPAND_FACT*A.nnz()*(fill_level+1); // EXPAND_FACT set by user
kh.create_spiluk_handle(SPILUKAlgorithm, nrows, nnzL, nnzU);
```

- ▶ Call symbolic routine

```
spiluk_symbolic(&kh, fill_level, A_rowmap, A_entries,
               L_rowmap, L_entries, U_rowmap, U_entries);
```

- ▶ Call numeric routine

```
spiluk_numeric(&kh, fill_level, A_rowmap, A_entries, A_values,
              L_rowmap, L_entries, L_values,
              U_rowmap, U_entries, U_values);
```

## SPTRSV usage

- ▶ Sparse triangular solver:  $\{L, U\}x = b$ 
  - ▶ Fallback solver options
  - ▶ Supernode-based solver options
- ▶ Fallback implementation and TPL options:
  - ▶ Symbolic phase analyzes matrix structure
    - ▶ Level-scheduling employed to expose parallelism to solver
    - ▶ All rows within a level can be solved independently in parallel
    - ▶ Symbolic phase results stored within handle for reuse
  - ▶ Solve phase: Uses level-set information from symbolic to execute in parallel
  - ▶ Separate phases allows reuse of symbolic phase / level scheduling information
    - ▶ Use case: As direct solver for preconditioner for iterative solver methods, following factorization

## SPTRSV usage

- ▶ Algorithm options:
  - ▶ SEQLVLSCHD\_TP1: Seq. level scheduling, solver hierarchical parallelism
  - ▶ SEQLVLSCHD\_TP1CHAIN: Seq. level scheduling, solver hierarchical parallelism
    - ▶ "Chaining" of consecutive levels with few rows into single kernel launch
    - ▶ Reduces number of kernel launches for levels bound by launch overhead, e.g. banded matrices
  - ▶ SPTRSV\_CUSPARSE: Wrapper of NVIDIA's CuSPARSE triangular solver

## SPTRSV: Interface

- ▶  $\{L,U\}$ \_rowmap: Arrays storing row pointer offset, as a 1-D Kokkos::View
- ▶  $\{L,U\}$ \_entries: Arrays storing column indices, as a 1-D Kokkos::View
- ▶  $\{L,U\}$ \_values: Arrays storing corresponding matrix values, as a 1-D Kokkos::View
- ▶ Handle: Stores internal data structures from symbolic phase
  - ▶ Input: SPTRSVAlgorithm, number of rows, boolean (is lower triangular)
  - ▶ Templated on rowmap data type (size\_type), entries ordinal type (lno\_t), values scalar type (scalar\_t), execution space, "persistent" memory space, "temporary" memory space (unused here)
- ▶  $\{x,b\}$ : Dense vectors as rank-1 Views

## SPTRSV: Interface

## SPTRSV: Interface

- ▶ Include header file

```
#include "KokkosSparse_sptrsv.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

## SPTRSV: Interface

- ▶ Include header file

```
#include "KokkosSparse_sptrsv.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

## SPTRSV: Interface

- ▶ Include header file

```
#include "KokkosSparse_sptrsv.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

- ▶ Create sptrsv handle - separate handles required for L and U

```
kh.create_sptrsv_handle(SPTRSVAlgorithm, nrows, lower_tri);
```

## SPTRSV: Interface

- ▶ Include header file

```
#include "KokkosSparse_sptrsv.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

- ▶ Create sptrsv handle - separate handles required for L and U

```
kh.create_sptrsv_handle(SPTRSVAlgorithm, nrows, lower_tri);
```

- ▶ Call symbolic analysis

```
sptrsv_symbolic(&kh, rowmap, entries, values);
```

## SPTRSV: Interface

- ▶ Include header file

```
#include "KokkosSparse_sptrsv.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

- ▶ Create sptrsv handle - separate handles required for L and U

```
kh.create_sptrsv_handle(SPTRSVAlgorithm, nrows, lower_tri);
```

- ▶ Call symbolic analysis

```
sptrsv_symbolic(&kh, rowmap, entries, values);
```

- ▶ Call solve

```
sptrsv_solve((&kh, rowmap, entries, values, b, x);
```

## SPTRSV supernode-based usage

- ▶ Users responsible for reordering and factorization to provide supernode block info
- ▶ Metis and SUPERLU software was used for this during development of the algorithm
- ▶ Symbolic phase analyzes matrix structure
  - ▶ Level-set scheduling of supernode blocks
  - ▶ Internal data structures are setup to store blocks
  - ▶ Symbolic phase results stored within handle for reuse
  - ▶ Optional: Merge supernodes with matching sparsity pattern
- ▶ Compute phase
  - ▶ Copy triangular matrix data to internal data structures
  - ▶ Optional: Invert diagonal blocks; apply inverse to off-diagonal blocks
- ▶ Solve phase: Uses level-set information to execute in parallel
- ▶ Separate phases allows reuse of symbolic and compute

## SPTRSV supernode-based usage

- ▶ Algorithm options:
  - ▶ SUPERNODAL\_DAG: Applies batched TRSV/GEMV kernels to supernodes at each level, internally computes DAG for scheduling
  - ▶ SUPERNODAL\_SPMV\_DAG: Applies SPMV at each level and requires inverted diagonal blocks, internally computes DAG for scheduling
  - ▶ SUPERNODAL\_ETREE: Like SUPERNODAL\_DAG, scheduling based on user-provided etree (e.g. from SuperLU)
  - ▶ SUPERNODAL\_SPMV: Like SUPERNODAL\_SMPV\_DAG, scheduling based on user-provided etree
- ▶ For more details see: I. Yamazaki, S. Rajamanicakm, N. Ellingwood “Performance Portable Supernode-based Sparse Triangular Solver for Manycore Architectures”,  
<https://dl.acm.org/doi/fullHtml/10.1145/3404397.3404428>

## SPTRSV: Interface supernode-based

- ▶  $\{L,U\}$ .graph: StaticCrsGraph data structure containing rowmap offsets and column indices
- ▶ nsuper: Number of supernode blocks
- ▶ supercols: Array of supernode block sizes
- ▶ etree: (Optional) Used for level scheduling
- ▶ Handle: Stores internal data structures and matrix blocks from symbolic and compute
  - ▶ Input: SPTRSVAlgorithm, number of rows, boolean (is lower triangular)
  - ▶ Templated on rowmap data type (size\_type), entries ordinal type (Ino\_t), values scalar type (scalar\_t), execution space, "persistent" memory space, "temporary" memory space (unused here)
- ▶  $\{x,b\}$ : Dense vectors as rank-1 Views

## SPTRSV: Interface supernode-based

## SPTRSV: Interface supernode-based

- ▶ Include header file

```
#include "KokkosSparse_sptrsv_supernode.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

## SPTRSV: Interface supernode-based

- ▶ Include header file

```
#include "KokkosSparse_sptrsv_supernode.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

## SPTRSV: Interface supernode-based

- ▶ Include header file

```
#include "KokkosSparse_sptrsv_supernode.hpp"  
//SPTRSV in Experimental namespace — interface may evolve  
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle  
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

- ▶ Create sptrsv handle - separate handles for L and U

```
khL.create_sptrsv_handle(SPTRSVAlgorithm::SUPERNODAL_SPMV_DAG, nrows,  
    lower_tri);  
khU.create_sptrsv_handle(SPTRSVAlgorithm::SUPERNODAL_SPMV_DAG, nrows,  
    lower_tri);
```

## SPTRSV: Interface supernode-based

- ▶ Include header file

```
#include "KokkosSparse_sptrsv_supernode.hpp"
//SPTRSV in Experimental namespace — interface may evolve
using namespace KokkosKernels::Experimental;
```

- ▶ Create opaque handle

```
KokkosKernelsHandle
<size_t, lno_t, scalar_t, exec_sp, mem_sp, mem_sp> kh;
```

- ▶ Create sptrsv handle - separate handles for L and U

```
khL.create_sptrsv_handle(SPTRSVAlgorithm::SUPERNODAL_SPMV_DAG, nrows,
    lower_tri);
khU.create_sptrsv_handle(SPTRSVAlgorithm::SUPERNODAL_SPMV_DAG, nrows,
    lower_tri);
```

- ▶ Set options

```
// whether to merge supernodes (false defaults)
khL.set_sptrsv_merge_supernodes (merge);

// invert diagonal blocks
khL.set_sptrsv_invert_diagonal (invert_diag);

// whether to apply diagonal-inversion to off-diagonal blocks
khL.set_sptrsv_invert_offdiagonal (invert_offdiag);
```

## SPTRSV: Interface supernode-based

- ▶ Call symbolic analysis

```
sptrsv_supernodal_symbolic (nsuper, supercols.data (), etree, L.graph, &
                             khL, L.graph, &khU);
```

## SPTRSV: Interface supernode-based

- ▶ Call symbolic analysis

```
sptrsv_supernodal_symbolic (nsuper, supercols.data (), etree, L.graph, &khL, L.graph, &khU);
```

- ▶ Call compute

```
sptrsv_compute (&khL, L);
```

## SPTRSV: Interface supernode-based

- ▶ Call symbolic analysis

```
sptrsv_supernodal_symbolic (nsuper, supercols.data (), etree, L.graph, &khL, L.graph, &khU);
```

- ▶ Call compute

```
sptrsv_compute (&khL, L);
```

- ▶ Call solve

```
sptrsv_solve (&khL, x, b);
```

## Use Case: Preconditioned Conjugate Gradient Solver

Assume A and M are both symmetric and positive-definite

### Conjugate Gradient

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A} * \mathbf{x}_0$$

$$\mathbf{p}_0 = \mathbf{r}_0$$

$$k = 0$$

**while**  $\|\mathbf{r}_k\| > \varepsilon$  and  $k < N$

$$\alpha = \frac{\mathbf{r}_k^T * \mathbf{r}_k}{\mathbf{p}_k^T * \mathbf{A} * \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha * \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha * \mathbf{A} * \mathbf{p}_k$$

$$\beta = \frac{\mathbf{r}_{k+1}^T * \mathbf{r}_{k+1}}{\mathbf{r}_k^T * \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta * \mathbf{p}_k$$

$$k = k + 1$$

### Preconditioned Conjugate Gradient

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A} * \mathbf{x}_0$$

$$\mathbf{z}_0 = \mathbf{M}^{-1} * \mathbf{r}_0$$

$$\mathbf{p}_0 = \mathbf{z}_0$$

$$k = 0$$

**while**  $\|\mathbf{r}_k\| > \varepsilon$  and  $k < N$

$$\alpha = \frac{\mathbf{r}_k^T * \mathbf{z}_k}{\mathbf{p}_k^T * \mathbf{A} * \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha * \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha * \mathbf{A} * \mathbf{p}_k$$

$$\mathbf{z}_{k+1} = \mathbf{M}^{-1} * \mathbf{r}_{k+1}$$

$$\beta = \frac{\mathbf{r}_{k+1}^T * \mathbf{z}_{k+1}}{\mathbf{r}_k^T * \mathbf{z}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta * \mathbf{p}_k$$

$$k = k + 1$$

## Use Case: Preconditioned Conjugate Gradient Solver

Goal: Introduce preconditioning to a CG solver code

- ▶ SPILUK: Yields factored  $M$
- ▶  $M = LU \approx A$
- ▶  $M^{-1} = U^{-1}L^{-1}$
  
- ▶ SPTRSV: Apply twice for  $z$
- ▶  $tmp = L \setminus r$  (Matlab notation)
- ▶  $z = U \setminus tmp$

### Preconditioned Conjugate Gradient

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A} * \mathbf{x}_0$$

$$\mathbf{z}_0 = \mathbf{M}^{-1} * \mathbf{r}_0$$

$$\mathbf{p}_0 = \mathbf{z}_0$$

$$k = 0$$

**while**  $\|\mathbf{r}_k\| > \varepsilon$  and  $k < N$

$$\alpha = \frac{\mathbf{r}_k^T * \mathbf{z}_k}{\mathbf{p}_k^T * \mathbf{A} * \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha * \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha * \mathbf{A} * \mathbf{p}_k$$

$$\mathbf{z}_{k+1} = \mathbf{M}^{-1} * \mathbf{r}_{k+1}$$

$$\beta = \frac{\mathbf{r}_{k+1}^T * \mathbf{z}_{k+1}}{\mathbf{r}_k^T * \mathbf{z}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta * \mathbf{p}_k$$

$$k = k + 1$$

## Preconditioned CG: Exercise

- ▶ Objective:
  - ▶ Introduce `spiluk` and `sptrsv` as preconditioning to a CG solver
- ▶ Exercise and logistics:
  - ▶ `Exercises/kokkoskernels/CGSolve_SpILUKprecond`
  - ▶ For convenience, use the provided script to install KokkosKernels and generate a Makefile for the code

```
run_installlibs_cmake.sh
```

- ▶ Change path and build variables as needed based on your setup:
  - ▶ `KOKKOS_PATH`: Point to your Kokkos source directory
  - ▶ `KOKKOSKERNELS_PATH`: KokkosKernels source directory
  - ▶ `KOKKOS_DEVICES`: Enabled execution spaces

## Preconditioned CG: Exercise

- ▶ Instructions:
  - ▶ Search for lines marked "EXERCISE" to apply code changes
  - ▶ Lines marked "EXERCISE hint" give suggestions
  - ▶ Opaque handles already created, SPILUK handle initialized
- ▶ Key steps:
  - ▶ Initialize two SPTRSV handles (L and U)
  - ▶ Call `spiluk_symbolic(...)` for ILU(k)
  - ▶ Call `spiluk_numeric(...)` for ILU(k) factorization
  - ▶ Call `sptrsv_symbolic(...)` to do level scheduling for L and U
  - ▶ Call `sptrsv_solve(...)` to apply preconditioner during the CGSolve
- ▶ Observe the convergence behaviors:
  - ▶ without preconditioner
  - ▶ with preconditioner (as ILU(k) fill-level changes)
  - ▶ `./cgsolve -help` will show command-line options

## Summary: SPILUK and SPTRSV

- ▶ SPILUK
  - ▶ Two phase routine, using level scheduling to expose parallelism for the factorization
- ▶ SPTRSV
  - ▶ Two phase routine, using level scheduling to expose parallelism for the solve phase
  - ▶ CuSPARSE TPL support is available for NVIDIA GPUs

# Building Applications with Kokkos Kernels

## Learning objectives:

- ▶ Using Kokkos Kernels in Your Project
- ▶ Configure, Build, and Install Kokkos Kernels
- ▶ Install with Spack

# Building Applications with Kokkos Kernels

## Learning objectives:

- ▶ Using Kokkos Kernels in Your Project
- ▶ Configure, Build, and Install Kokkos Kernels
- ▶ Install with Spack

## Ignore This For Tutorial Only

The following details on options to integrate Kokkos into your build process are NOT necessary to know if you just want to do the tutorial.

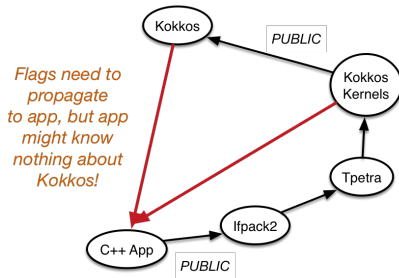
- ▶ **Install via CMake:** For large projects with multiple dependencies installing Kokkos via CMake and then building against it is the best option.
- ▶ **Build inline via CMake:** This is an option suited for applications which have few dependencies (and no one depending on them) and want to build Kokkos inline with their application.
- ▶ **Using Spack:** For projects which largely rely on components provided by the Spack package manager.

- ▶ In the spirit of C++ for *code* performance portability, modern CMake aims for *build system* portability
- ▶ Keep builds simple. Language is always C++ (even if CUDA, HIP, Sycl, ...) and all necessary flags are taken care of for you!
- ▶ Single build system call in your project should configure everything

```
add_library(myLib goTeamVenture.cpp)
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

- ▶ No need to link to Kokkos itself. Kokkos Kernels transitively applies all Kokkos flags.

- ▶ Developing large software tools with Kokkos requires handling transitive dependencies properly - thankfully this is fairly seamless with CMake
- ▶ Example: abstraction layers that hide Kokkos details
- ▶ App will still generate Kokkos code and needs all Kokkos flags



## Basic starting point for apps using kernels

- ▶ Create a CMakeLists.txt file for an executable

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

find_package(KokkosKernels REQUIRED)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myExe PRIVATE
    Kokkos::kokkoskernels)
```

## Basic starting point for apps using kernels

- ▶ Create a CMakeLists.txt file for an executable
- ▶ Declare your C++ project

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

find_package(KokkosKernels REQUIRED)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myExe PRIVATE
    Kokkos::kokkoskernels)
```

## Basic starting point for apps using kernels

- ▶ Create a CMakeLists.txt file for an executable
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project
```

```
find_package(KokkosKernels REQUIRED)
```

```
add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myExe PRIVATE
    Kokkos::kokkoskernels)
```

## Basic starting point for apps using kernels

- ▶ Create a CMakeLists.txt file for an executable
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency
- ▶ Add your program

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

find_package(KokkosKernels REQUIRED)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myExe PRIVATE
    Kokkos::kokkoskernels)
```

## Basic starting point for apps using kernels

- ▶ Create a CMakeLists.txt file for an executable
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency
- ▶ Add your program
- ▶ Link to Kokkos Kernels (PRIVATE, not transitive)

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

find_package(KokkosKernels REQUIRED)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myExe PRIVATE
    Kokkos::kokkoskernels)
```

## Basic starting point for app using kernels as submodule

- ▶ Create CMakeLists.txt for a library with Kokkos built inline

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

add_subdirectory(kokkos)
add_subdirectory(kokkos-kernels)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for app using kernels as submodule

- ▶ Create CMakeLists.txt for a library with Kokkos built inline
- ▶ Declare your C++ project

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

add_subdirectory(kokkos)
add_subdirectory(kokkos-kernels)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for app using kernels as submodule

- ▶ Create CMakeLists.txt for a library with Kokkos built inline
- ▶ Declare your C++ project
- ▶ Add Kokkos as a subdirectory

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project
```

```
add_subdirectory(kokkos)
add_subdirectory(kokkos-kernels)
```

```
add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for app using kernels as submodule

- ▶ Create CMakeLists.txt for a library with Kokkos built inline
- ▶ Declare your C++ project
- ▶ Add Kokkos as a subdirectory
- ▶ Add your program

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

add_subdirectory(kokkos)
add_subdirectory(kokkos-kernels)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for app using kernels as submodule

- ▶ Create CMakeLists.txt for a library with Kokkos built inline
- ▶ Declare your C++ project
- ▶ Add Kokkos as a subdirectory
- ▶ Add your program
- ▶ Link your program to Kokkos Kernels

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX) # C++ needed to build my project

add_subdirectory(kokkos)
add_subdirectory(kokkos-kernels)

add_executable(myExe source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for helper libraries using kernels: PUBLIC dependencies

- ▶ Create a CMakeLists.txt file for your library

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX)

find_package(KokkosKernels REQUIRED)

add_library(myLib source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for helper libraries using kernels: PUBLIC dependencies

- ▶ Create a CMakeLists.txt file for your library
- ▶ Declare your C++ project

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX)

find_package(KokkosKernels REQUIRED)

add_library(myLib source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for helper libraries using kernels: PUBLIC dependencies

- ▶ Create a CMakeLists.txt file for your library
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX)
```

```
find_package(KokkosKernels REQUIRED)
```

```
add_library(myLib source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for helper libraries using kernels: PUBLIC dependencies

- ▶ Create a CMakeLists.txt file for your library
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency
- ▶ Add your library

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX)

find_package(KokkosKernels REQUIRED)

add_library(myLib source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

## Basic starting point for helper libraries using kernels: PUBLIC dependencies

- ▶ Create a CMakeLists.txt file for your library
- ▶ Declare your C++ project
- ▶ Find Kokkos Kernels dependency
- ▶ Add your library
- ▶ Link your library to Kokkos Kernels. Downstream apps will need Kokkos flags so linkage must be PUBLIC (i.e. transitive)

```
cmake_minimum_required(VERSION 3.12)
project(myProject CXX)

find_package(KokkosKernels REQUIRED)

add_library(myLib source.cpp)
# declare dependency on KokkosKernels
target_link_libraries(myLib PUBLIC
    Kokkos::kokkoskernels)
```

```
cmake <ProjectSourceDir> \  
-DCMAKE_CXX_COMPILER=<kokkos dir>/bin/nvcc_wrapper \  
-DKokkosKernels_ROOT=<KokkosInstallPrefix> \  
-DKokkosKernels_<OPTION>:BOOL=ON
```

- ▶ Point to your project source

```
cmake <ProjectSourceDir> \  
-DCMAKE_CXX_COMPILER=<kokkos dir>/bin/nvcc_wrapper \  
-DKokkosKernels_ROOT=<KokkosInstallPrefix> \  
-DKokkosKernels_<OPTION>:BOOL=ON
```

- ▶ Point to your project source
- ▶ Use the same C++ compiler as Kokkos

```
cmake <ProjectSourceDir> \  
-DCMAKE_CXX_COMPILER=<kokkos dir>/bin/nvcc_wrapper \  
-DKokkosKernels_ROOT=<KokkosInstallPrefix> \  
-DKokkosKernels_<OPTION>:BOOL=ON
```

- ▶ Point to your project source
- ▶ Use the same C++ compiler as Kokkos
- ▶ Point to Kokkos Kernels installation

```
cmake <ProjectSourceDir> \  
  -DCMAKE_CXX_COMPILER=<kokkos dir>/bin/nvcc_wrapper \  
  -DKokkosKernels_ROOT=<KokkosInstallPrefix> \  
  -DKokkosKernels_<OPTION>:BOOL=ON
```

- ▶ Point to your project source
- ▶ Use the same C++ compiler as Kokkos
- ▶ Point to Kokkos Kernels installation
- ▶ Pass any Kokkos Kernels options

```
cmake <ProjectSourceDir> \  
  -DCMAKE_CXX_COMPILER=<kokkos dir>/bin/nvcc_wrapper \  
  -DKokkosKernels_ROOT=<KokkosInstallPrefix> \  
  -DKokkosKernels_<OPTION>:BOOL=ON
```

- ▶ Options almost all fall into one of two categories
  - ▶ ETIs (early template instantiation) options
  - ▶ TPLs (third-party libraries like MKL and cuBLAS)
- ▶ Template instantiation pre-generates kernels for certain types to avoid compiler overheads later
  - ▶ Scalars: float, double, complex float, complex double
  - ▶ Ordinals: int, int64\_t
  - ▶ Offsets: int, size\_t
  - ▶ Spaces: CUDA, OpenMP, Serial
  - ▶ Layouts: left, right
- ▶ Third-party libraries enable using optimized vendor implementations
  - ▶ MKL
  - ▶ cuBLAS
  - ▶ cuSPARSE
  - ▶ SuperLU

- ▶ `-DKokkosKernels_INST_MEMSPACE_CUDAUVMSPACE=ON` says to pre-instantiate kernels with CUDA UVM
- ▶ `-DKokkosKernels_INST_FLOAT=ON` says to pre-instantiate kernels with 32-bit floats
- ▶ `-DKokkosKernels_ENABLE_TPL_MKL=ON` for MKL support
- ▶ `-DKokkosKernels_ENABLE_TPL_SUPERLU=ON,`  
`-DSUPERLU_ROOT=<...>` gives install location for SuperLU

- ▶ `-DKokkosKernels_INST_MEMSPACE_CUDAUVMSPACE=ON` says to pre-instantiate kernels with CUDA UVM
- ▶ `-DKokkosKernels_INST_FLOAT=ON` says to pre-instantiate kernels with 32-bit floats
- ▶ `-DKokkosKernels_ENABLE_TPL_MKL=ON` for MKL support
- ▶ `-DKokkosKernels_ENABLE_TPL_SUPERLU=ON`,  
`-DSUPERLU_ROOT=<...>` gives install location for SuperLU

### Activated options displayed in CMake output

#### KokkosKernels ETI Types

```
Devices:  <OpenMP,HostSpace>
Scalars:  double
Ordinals: int
Offsets:  int;size_t
Layouts:  LayoutLeft
```

#### KokkosKernels TPLs

```
BLAS:      /usr/lib/libblas.dylib
LAPACK:    /usr/lib/liblapack.dylib
```

- ▶ Spack provides a package manager that automatically downloads, configures, and installs package dependencies
- ▶ KokkosKernels itself can be easily installed with specific variants (+) and compilers (%)  

```
spack install kokkos-kernels@develop +openmp %gcc@8.3.0
```
- ▶ Good practice is to define “best variant” for kokkos in your `packages.yaml` directory, e.g. for Volta system

```
packages:  
  kokkos:  
    variants: +cuda +openmp +cuda_lambda +wrapper \  
              ^cuda@10.1 cuda_arch=70  
    compiler: [gcc@7.2.0]
```
- ▶ Build rules in `package.py` automatically map Spack variants to correct CMake options
- ▶ Run `spack info kokkos-kernels` to see full list of variants

- ▶ Build rules created in a `package.py` file
- ▶ Step 1: Declare dependency on specific version of kokkos (3.x, master, or develop)

```
class myLib(CMakePackage):  
    depends_on('kokkos-kernels@3.2')
```

- ▶ Step 2: Add build rule pointing to Spack-installed Kokkos and same C++ compiler Kokkos uses

```
def cmake_args(self):  
    options = []  
    ...  
    options.append('-DCMAKE_CXX_COMPILER={}'.format(  
        self.spec['kokkos'].kokkos_cxx)  
    options.append('-DKokkosKernels_ROOT={}'.format(  
        self.spec['kokkos-kernels'].prefix)  
    return options
```

- ▶ More details can be found in `Spack.md` in Kokkos repo.

- ▶ Kokkos primary build system is CMake.
- ▶ Kokkos options are transitively passed on, including many necessary compiler options.
- ▶ The Spack package manager does support Kokkos.

## Summary

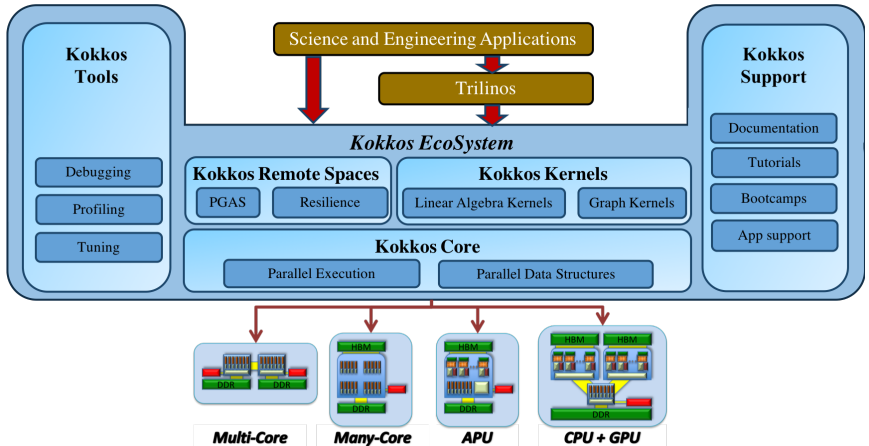
- ▶ Kokkos Kernels provides several performance portable dense, sparse, and graph kernels
- ▶ Working actively with vendors to incorporate algorithms in optimized vendor libraries and provide interface to them
- ▶ Kokkos Core, Kokkos Kernels, and Kokkos tools allows Kokkos ecosystem to be a complete solution for CSE applications
- ▶ Working actively with applications to address unique use cases
- ▶ Use slack/github/e-mail to reach out to us

This concludes:

# **The Kokkos Lectures!**

*8 Lectures – 600 slides – 14 hours of recording.*

# The Kokkos Ecosystem



**What was covered:**

- ▶ Module 1: Introduction, Building and Parallel Dispatch
- ▶ Module 2: Views and Spaces
- ▶ Module 3: Data Structures + MultiDimensional Loops
- ▶ Module 4: Hierarchical Parallelism
- ▶ Module 5: Tasking, Streams and SIMD
- ▶ Module 6: InterOp: Python, Fortran, MPI and PGAS
- ▶ Module 7: Tools: Profiling, Tuning and Debugging
- ▶ Module 8: Kernels: Sparse and Dense Linear Algebra
- ▶ Module 9: Fortran inter-op



- Kokkos Core:** C.R.Trott, J. Ciesko, V. Dang, N. Ellingwood, D.S. Hollman, D. Ibanez, J. Miles, J. Wilke, , H. Finkel, N. Liber, D. Lebrun-Grandie, D. Arndt, B. Turcksin, J. Madsen, R. Gayatri  
former: H.C. Edwards, D. Labreche, G. Mackey, S. Bova, D. Sunderland
- Kokkos Kernels:** S. Rajamanickam, L. Berger, V. Dang, N. Ellingwood, E. Harvey, B. Kelley, K. Kim, C.R. Trott, J. Wilke, S. Acer  
former: M. Deveci, M. Hoemmen, A. Bradley
- Kokkos Tools** D. Poliakoff, C. Lewis, S. Hammond, D. Ibanez, J. Madsen, S. Moore, C.R. Trott
- Kokkos Support** C.R. Trott, G. Shipmann, G. Womeldorff, and all of the above  
former: H.C. Edwards, G. Lopez, F. Foertter, J. Amelang

**H. Carter Edwards:** who started this all!

**Mike Heroux:** for believing in Kokkos and being a champion for the project in its early phase.

**Jeff Amelang:** who developed the original tutorial material 2015.

**Sandia LDRD program:** which did the initial funding of Kokkos.

**DOE Exascale Computing Project:** for funding training efforts and the expansion of the Kokkos team to more institutions.

## 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 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.