The Kokkos Ecosystem

Christian 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.
Cost of Porting Code

10 LOC / hour ~ 20k LOC / year

- Optimistic estimate: 10% of an application is modified to adopt an on-node Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - 500k x 10% => Typical App Port 2.5 Man-Years
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Man-Years
  - Trilinos: 4,000k Lines x 10% => 20 Man-Years
What is Kokkos?

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, HPX, ...
  - Aims to be descriptive not prescriptive
  - Aligns with developments in the C++ standard
- Expanding solution for common needs of modern science/engineering codes
  - Math libraries based on Kokkos
  - Tools which allow inside into Kokkos
- It is Open Source
  - Maintained and developed at https://github.com/kokkos
- It has many users at wide range of institutions.
Kokkos Development Team

Kokkos Core:  
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:  
S. Hammond, C.R. Trott, D. Ibanez, S. Moore, L. Cannada, D. Poliakoff

Kokkos Support:  
C.R. Trott, G. Shipman, G. Lopez, G. Womeldorff, 
former: H.C. Edwards, D. Labreche, Fernanda Foertter
Kokkos Core Abstractions

Data Structures
- Memory Spaces ("Where")
  - HBM, DDR, Non-Volatile, Scratch
- Memory Layouts
  - Row/Column-Major, Tiled, Strided
- Memory Traits ("How")
  - Streaming, Atomic, Restrict

Parallel Execution
- Execution Spaces ("Where")
  - CPU, GPU, Executor Mechanism
- Execution Patterns
  - parallel_for/reduce/scan, task-spawn
- Execution Policies ("How")
  - Range, Team, Task-Graph
Kokkos Core Capabilities

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Loops</td>
<td><code>parallel_for( N, KOKKOS_LAMBDA (int i) { ...BODY... });</code></td>
</tr>
</tbody>
</table>
| Parallel Reduction    | `parallel_reduce( RangePolicy<ExecSpace>(0,N), KOKKOS_LAMBDA (int i, double& upd) { ...
|                       |  ...BODY...
|                       |  upd += ...
|                       | }, Sum<>(result));                                                      |
| Tightly Nested Loops  | `parallel_for(MDRangePolicy<Rank<3> > {{0,0,0},{N1,N2,N3},{T1,T2,T3},
|                       |  KOKKOS_LAMBDA (int i, int j, int k) {...BODY...});`                   |
| Non-Tightly Nested Loops | `parallel_for( TeamPolicy<Schedule<Dynamic>>( N, TS ), KOKKOS_LAMBDA (Team team) {
|                       |  … COMMON CODE 1 …
|                       |  parallel_for(TeamThreadRange( team, M(N)), [&] (int j) { ... INNER BODY... });
|                       |  … COMMON CODE 2 …
|                       | });`                                                                    |
| Task Dag              | `task_spawn( TaskTeam( scheduler , priority), KOKKOS_LAMBDA (Team team) { … BODY });` |
| Data Allocation       | `View<double**, Layout, MemSpace> a("A",N,M);`                         |
| Data Transfer         | `deep_copy(a,b);`                                                      |
| Atomics               | `atomic_add(&a[i],5.0); View<double*,MemoryTraits<AtomicAccess>> a(); a(i)+=5.0;` |
| Exec Spaces           | `Serial, Threads, OpenMP, Cuda, HPX (experimental), HIP (experimental), OpenMPTarget (experimental)` |
More Kokkos Capabilities

- MemoryPool
- parallel_scan
- DualView
- ScatterView
- OffsetView
- StaticWorkGraph
- UnorderedMap
- Reducers
- RandomPool
- sort
- StaticWorkGraph
- kokkos_malloc
- kokkos_free
- LayoutRight
- Vector
- Bitset
- LayoutLeft
- UniqueToken
- ScratchSpace
- LayoutStrided
- ProfilingHooks
Example: Conjugent Gradient Solver

- Simple Iterative Linear Solver
- For example used in MiniFE
- Uses only three math operations:
  - Vector addition (AXPBY)
  - Dot product (DOT)
  - Sparse Matrix Vector multiply (SPMV)
- Data management with Kokkos Views:

```cpp
View<double*,HostSpace,MemoryTraits<Unmanaged>> > h_x(x_in, nrows);
View<double*> x("x",nrows);
depth_copy(x,h_x);
```
CG Solve: The AXPBY

- Simple data parallel loop: Kokkos::parallel_for
- Easy to express in most programming models
- Bandwidth bound
- Serial Implementation:

```c
void axpby(int n, double* z, double alpha, const double* x,
          double beta, const double* y) {
    for(int i=0; i<n; i++)
        z[i] = alpha*x[i] + beta*y[i];
}
```

- Kokkos Implementation:

```c
void axpby(int n, View<double*> z, double alpha, const double* x,
           double beta, const double* y) {
    parallel_for("AXpBY", n, KOKKOS_LAMBDA (const int i) {
        z(i) = alpha*x(i) + beta*y(i);
    });
}
```
CG Solve: The Dot Product

- Simple data parallel loop with reduction: Kokkos::parallel_reduce
- Non trivial in CUDA due to lack of built-in reduction support
- Bandwidth bound
- Serial Implementation:

```cpp
double dot(int n, const double* x, const double* y) {
    double sum = 0.0;
    for(int i=0; i<n; i++)
        sum += x[i]*y[i];
    return sum;
}
```

- Kokkos Implementation:

```cpp
double dot(int n, View<const double*> x, View<const double*> y) {
  double x_dot_y = 0.0;
  parallel_reduce("Dot",n, KOKKOS_LAMBDA (const int i, double& sum) {
    sum += x[i]*y[i];
  }, x_dot_y);
  return x_dot_y;
}
```

Parallel Pattern: loop with reduction
Iteration Index + Thread-Local Red. Varible
CG Solve: Sparse Matrix Vector Multiply

- Loop over rows
- Dot product of matrix row with a vector
- Example of Non-Tightly nested loops
- Random access on the vector (Texture fetch on GPUs)

```c
void SPMV(int nrows, const int* A_row_offsets, const int* A_cols,
          const double* A_vals, double* y, const double* x) {
    for(int row=0; row<nrows; ++row) {
        double sum = 0.0;
        int row_start=A_row_offsets[row];
        int row_end=A_row_offsets[row+1];
        for(int i=row_start; i<row_end; ++i) {
            sum += A_vals[i]*x[A_cols[i]];
        }
        y[row] = sum;
    }
}
```
void SPMV(int nrows, View<const int*> A_row_offsets, View<const int*> A_cols, View<const double*> A_vals, View<double*> y, View<const double*, MemoryTraits< RandomAccess>> x) {

  // Performance heuristic to figure out how many rows to give to a team
  int rows_per_team = get_row_chunking(A_rowOffsets);

  parallel_for("SPMV:Hierarchy", TeamPolicy< Schedule< Static >> ((nrows+rows_per_team-1)/rows_per_team,AUTO,8),
      KOKKOS_LAMBDA (const TeamPolicy<>::member_type& team) {
        const int first_row = team.league_rank()*rows_per_team;
        const int last_row = first_row+rows_per_team<nrows? first_row+rows_per_team : nrows;

        parallel_for(TeamThreadRange(team,first_row,last_row),[&] (const int row) {
            const int row_start=A_row_offsets[row];
            const int row_length=A_row_offsets[row+1]-row_start;
            double y_row;
            parallel_reduce(ThreadVectorRange(team,row_length),[&] (const int i, double& sum) {
                sum += A_vals(i+row_start)*x(A_cols(i+row_start));
            }, y_row);
            y(row) = y_row;
        });
    });
}
CG Solve: Performance

- Comparison with other Programming Models
- Straight forward implementation of kernels
- OpenMP 4.5 is immature at this point
- Two problem sizes: 100x100x100 and 200x200x200 elements
Kokkos Kernels

- BLAS, Sparse and Graph Kernels on top of Kokkos and its View abstraction
  - Scalar type agnostic, e.g. works for any types with math operators
  - Layout and Memory Space aware
- Can call vendor libraries when available
- Views contain size and stride information => Interface is simpler

```cpp
// BLAS
int M,N,K,LDA,LDB; double alpha, beta; double *A, *B, *C;
dgemm('N','N',M,N,K,alpha,A,LDA,B,LDB,beta,C,LDC);

// Kokkos Kernels
double alpha, beta; View<double**> A,B,C;
gemm('N','N',alpha,A,B,beta,C);
```

- Interface to call Kokkos Kernels at the teams level (e.g. in each CUDA-Block)

```cpp
parallel_for("NestedBLAS", TeamPolicy<>(N,AUTO), KOKKOS_LAMBDA (const team_handle_t& team_handle) {
    // Allocate A, x and y in scratch memory (e.g. CUDA shared memory)
    // Call BLAS using parallelism in this team (e.g. CUDA block)
    gemv(team_handle,'N',alpha,A,x,beta,y)
});
```
Kokkos Tools

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

- Performance tuning requires insight, but tools are different on each platform
- KokkosTools: Provide common set of basic tools + hooks for 3rd party tools
- Common issue: abstraction layers obfuscate profiler output
  - Kokkos hooks for passing names on
  - Provide Kernel, Allocation and Region
- No need to recompile
  - Uses runtime hooks
  - Set via env variable
Kokkos Tools Integration with 3rd Party

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

**TAU Example:**

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

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

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

        Kokkos::parallel_for(TTR(t, 1), [i](int i) {
            Lambda capture modifies reference capture variable 'a' that is a local
            a += 1;
            cv() += 1;
        });
    });

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

- Production Code Running Real Analysis Today
  - We got about 12 or so.
- Production Code or Library committed to using Kokkos and actively porting
  - Somewhere around 35
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around 65
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate 100-150 packages.
Some Kokkos Users
LAMMPS

- Widely used Molecular Dynamics Simulations package
- Focused on Material Physics
- Over 500 physics modules
- Kokkos covers growing subset of those
- REAX is an important but very complex potential
  - USER-REAXC (Vanilla) more than 10,000 LOC
  - Kokkos version ~6,000 LOC
  - LJ in comparison: 200LOC
  - Used for shock simulations

Architecture Comparison
Example in.reaxc.tatb / 196k atoms / 100 steps
Sparta: Production Simulation at Scale

- **Stochastic PARallel Rarefied-gas Time-accurate Analyzer**
- A direct simulation Monte Carlo code
- Developers: *Steve Plimpton, Stan Moore, Michael Gallis*
- Only code to have run on all of Trinity
  - 3 Trillion particle simulation using both HSW and KNL partition in a single MPI run (~20k nodes, ~1M cores)
- Benchmarked on 16k GPUs on Sierra
  - Production runs now at 5k GPUs
- Co-Designed Kokkos::ScatterView
Uintah

- System wide many task framework from University of Utah led by Martin Berzins
- Multiple applications for combustion/radiation simulation
- Structured AMR Mesh calculations
- Prior code existed for CPUs and GPUs
- Kokkos unifies implementation
- Improved performance due to constraints in Kokkos which encourage better coding practices

Questions: Dan Sunderland
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
- Cray has also announced El Capitan at LLNL (internals still unreleased)
- Have been planning for this eventuality:
  - Kokkos ECP project extended to include ANL, ORNL, and LBNL
  - HIP backend for AMD: main development at ORNL
  - SYCL hybrid for Intel: main development at ANL
  - OpenMPTarget for AMD, Intel and NVIDIA, lead at Sandia
OpenMP-Target Backend

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

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

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

- **Early days**
  - Parallel_for
  - USMMemory space Rank 1
  - Functionality testing on Gen 9 hardware
# Feature Timeline

<table>
<thead>
<tr>
<th>Feature</th>
<th>HIP</th>
<th>DPC++</th>
<th>OpenMP Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemorySpace</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>parallel_for</code> RangePolicy</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>parallel_for</code> MDR range Policy</td>
<td>X</td>
<td>03/20</td>
<td></td>
</tr>
<tr>
<td><code>parallel_reduce</code> RP</td>
<td>X</td>
<td>02/20</td>
<td></td>
</tr>
<tr>
<td><code>parallel_reduce</code> MDRP</td>
<td>05/20</td>
<td>Q4 20</td>
<td>05/20</td>
</tr>
<tr>
<td>Reducers</td>
<td>X</td>
<td>Q4 20</td>
<td></td>
</tr>
<tr>
<td><code>parallel_for</code> TP</td>
<td>03/20</td>
<td></td>
<td>03/20</td>
</tr>
<tr>
<td><code>parallel_reduce</code> TP</td>
<td>06/20</td>
<td></td>
<td>06/20</td>
</tr>
<tr>
<td>atomics</td>
<td>03/20</td>
<td></td>
<td>04/20</td>
</tr>
</tbody>
</table>
Kokkos - C++ Standard integration cycle

- Propose new features for C++ Standard
- Kokkos
- C++ Standard
- C++ Backport
- Kokkos Legacy

- Port accepted features to legacy versions
- Implemented legacy capabilities in terms of new C++ features
- Back port to current compilers
C++ Features in the Works

- First success: `atomic_ref<T>` in C++20
  - Provides atomics with all capabilities of atomics in Kokkos
  - `atomic_ref(a[i])+=5.0;` instead of `atomic_add(&a[i],5.0);`
- Next thing: `Kokkos::View` => `std::mdspan`
  - Provides customization points which allow all things we can do with `Kokkos::View`
  - Better design of internals though! => Easier to write custom layouts.
  - Also: arbitrary rank (until compiler crashes) and mixed compile/runtime ranks
  - We hope will land early in the cycle for C++23 (i.e. early in 2020)
  - Production reference implementation: [https://github.com/kokkos/mdspan](https://github.com/kokkos/mdspan)
- Also C++23: Executors and Basic Linear Algebra: [https://github.com/kokkos/stdblas](https://github.com/kokkos/stdblas)
Features for 3.0 - 3.2

- Full CMake build system with Spack
  
  ```cmake
  find_package(Kokkos REQUIRED)
  add_library(mylibrary ${SOURCES})
  target_link_libraries(mylibrary PUBLIC Kokkos::kokkos)
  ```

- SIMD Types (e.g. Kokkos::simd<double,FixedSize<16>>)
- CUDA Graphs Support
- Improvements in test automation
- Resilience / Remote spaces
- Kokkos/Umpire integration
  - MemoryPool based memory spaces for applications
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