The Kokkos C++ Performance Portability EcoSystem

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

Unclassified Unlimited Release
D. Sunderland, N. Ellingwood, D. Ibanez, S. Bova, J. Miles, D. Hollman, V. Dang

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.
Goals For Performance Portability

- One coherent approach to low level HPC performance portability needs
  - Parallel Execution
  - Data Structures and Management
  - Math Kernels
  - Tools

- Limit cognitive overload
  - Orthogonalization of concerns
  - Most of the time no explicit reference to backends (e.g. CUDA, or OpenMP)

- Off ramp via standards integration to limit scope
  - Invest into C++ standards work to make Kokkos a “sliding window” of advanced capabilities
Kokkos EcoSystem

Science and Engineering Applications

Kokkos Tools
- Debugging
- Profiling
- Tuning

Kokkos Kernels
- Linear Algebra Kernels
- Graph Kernels

Kokkos Core
- Parallel Execution
- Parallel Data Structures

Kokkos Remote Spaces
- PGAS
- IO

Kokkos Support
- Documentation
- Tutorials
- Bootcamps
- App support

Multi-Core
Many-Core
APU
CPU + GPU
Kokkos Development Team

- Dedicated team with a number of staff working most of their time on Kokkos
  - Main development team at Sandia in CCR – Sandia Apps are customers

Kokkos Core:  
**C.R. Trott**, D. Sunderland, N. Ellingwood, D. Ibanez, S. Bova, J. Miles, D. Hollman, V. Dang,  
soon: H. Finkel, N. Liber, D. Lebrun-Grandie, A. Prokopenko  
former: **H.C. Edwards**, D. Labreche, G. Mackey

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

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
Patterns and Policy

- Reduce cognitive overload by reusing the same code structure
  - **Parallel_Pattern**(*ExecutionPolicy*, *FunctionObject*, *ReductionArgs*)
    
    // Basic parallel for:
    parallel_for(N, Lambda);
    // Parallel for with dynamic scheduling:
    parallel_for(RangePolicy<Schedule<Dynamic>>(0,N), Lambda);
    // Parallel Reduce with teams:
    parallel_reduce(TeamPolicy<>)(N,AUTO), Lambda, Reducer);
    // Parallel Scan with a nested policy
    parallel_scan(ThreadVectorRange(team_handle,N), Lambda);
    // Restriction pattern equivalent to #pragma omp single
    single(PerTeam(team_handle), Lambda);
    // Task Spawn
    task_spawn(TeamTask(scheduler, dependency), Task);

- Orthogonalize further via “require” mechanism to customize exec policy
  
  auto exec_policy_low_latency = require(exec_policy,KernelProperty::HintLightWeight);
### 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), ROCm (experimental)` |
More Kokkos Capabilities

- MemoryPool
- ParallelScan
- DualView
- ScatterView
- OffsetView
- StaticWorkGraph
- UnorderedMap
- RandomPool
- Reducers
- Sort
- KokkosMalloc
- KokkosFree
- LayoutRight
- LayoutLeft
- LayoutStrided
- Vector
- Bitset
- ScratchSpace
- ProfilingHooks
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

- View have all their size and stride information => Interface is simpler

```c
// 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);
```

```c
// 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)

```c
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 & Debugging

- Performance tuning requires insight, but tools are different on each platform
- Insight into
- KokkosTools: Provide common set of basic tools + hooks for 3rd party tools
- One 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
Improved Fine Grained Tasking

- Generalization of TaskScheduler abstraction to allow user to be generic with respect to scheduling strategy and queue
- Implementation of new queues and scheduling strategies:
  - Single shared LIFO Queue (this was the old implementation)
  - Multiple shared LIFO Queues with LIFO work stealing
  - Chase-Lev minimal contention LIFO with tail (FIFO) stealing
  - Potentially more
- Reorganization of Task, Future, TaskQueue data structures to accommodate flexible requirements from the TaskScheduler
  - For instance, some scheduling strategies require additional storage in the Task

Questions: David Hollman
Kokkos Remote Spaces: PGAS Support

- PGAS Models may become more viable for HPC with both changes in network architectures and the emergence of “super-node” architectures
  - Example DGX2
  - First “super-node”
  - 300GB/s per GPU link

- Idea: Add new memory spaces which return data handles with shmem semantics to Kokkos View
  - `View<double**[3], LayoutLeft, NVShmemSpace> a(“A”,N,M);`
  - Operator `a(i,j,k)` returns:

```
template<>
struct NVShmemElement<double> {
  NVShmemElement(int pe_, double* ptr_):pe(pe_),ptr(ptr_){}
  int pe; double* ptr;
  void operator = (double val) { shmem_double_p(ptr,val,pe); }
};
```
PGAS Performance Evaluation: miniFE

- Test Problem: CG-Solve
  - Using the miniFE problem $N^3$
  - Compare to optimized CUDA
  - MPI version is using overlapping
  - DGX2 4 GPU workstation
  - Dominated by SpMV (Sparse Matrix Vector Multiply)
  - Make Vector distributed, and store global indicies in Matrix

- 3 Variants
  - Full use of SHMEM
  - Inline functions by ptr mapping
    - Store 16 pointers in the View
  - Explicit by-rank indexing
    - Make vector 2D
    - Encode rank in column index

Warning: I don’t think this is a viable thing in the next couple years for most of our apps!!
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 30
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around 50
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate 80-120 packages.
Kokkos Users

Pacific Northwest
NATIONAL LABORATORY

Los Alamos
NATIONAL LABORATORIES

Sandia
National Laboratories

ARL

Michigan State

NREL

Oak Ridge
National Laboratory

Argonne
NATIONAL LABORATORY

U.S. NAVAL RESEARCH
LABORATORY

THE UNIVERSITY OF UTAH

Max-Planck-Institut
für Plasmaphysik

IPP

CEA

Jülich
Forschungszentrum

Rensselaer

Berkeley Lab

TUM

Technische
UNIVERSITAT
MÜNCHEN
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 Sunderlan
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

Questions: Stan Moore
Alexa

- Portably performant shock hydrodynamics application
- Solving multi-material problems for internal Sandia users
- Uses tetrahedral mesh adaptation

Questions: Dan Ibanez

- All operations are Kokkos-parallel
- Test case: metal foil expanding due to resistive heating from electrical current.
- Goal: solve aerodynamics problems for Sandia (transonic and hypersonic) on ‘leadership’ class supercomputers
- Solves compressible Navier-Stokes equations
- Perfect and reacting gas models
- Laminar and RANS turbulence models -> hybrid RANS-LES
- Primary discretization is cell-centered finite volume
- Research on high-order finite difference and discontinuous Galerkin discretizations
- Structured and unstructured grids

4 Sierra nodes (16x V100) equivalent to ~40 Trinity nodes (80x Haswell 16c CPU)
Aligning Kokkos with the C++ Standard

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

Propose for C++

Back port to compilers we got

Move accepted features to legacy support

Implemented legacy capabilities in terms of new C++ features
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)

- Also C++23: Executors and **Basic Linear Algebra** (just began design work)
Towards C++23 Executors

- C++ standard is moving towards more asynchronicity with Executors
  - Dispatch of parallel work consumes and returns new kind of future

- Aligning Kokkos with this development means:
  - Introduction of Execution space instances (CUDA streams work already)
    ```
    DefaultExecutionSpace spaces[2];
    partition( DefaultExecutionSpace(), 2, spaces);
    // f1 and f2 are executed simultaneously
    parallel_for( RangePolicy<>)(spaces[0], 0, N, f1);
    parallel_for( RangePolicy<>)(spaces[1], 0, N, f2);
    // wait for all work to finish
    fence();
    ```

  - Patterns return futures and Execution Policies consume them
    ```
    auto fut_1 = parallel_for( RangePolicy<>("Funct1", 0, N), f1 );
    auto fut_2a = parallel_for( RangePolicy<>("Funct2a", fut_1, 0, N), f2a);
    auto fut_2b = parallel_for( RangePolicy<>("Funct2b", fut_1, 0, N), f2b);
    auto fut_3 = parallel_for( RangePolicy<>("Funct3", all(fut_2a,fut2_b),0, N), f3);
    fence(fut_3);
    ```