Kokkos – A Brief Overview

Christian R. Trott, - Center for Computing Research
Sandia National Laboratories/NM
What Is Kokkos And Why Does It Matter?
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 enable insight into Kokkos
- It is Open Source
  - Maintained and developed at https://github.com/kokkos
- It has many users at wide range of institutions.
Kokkos Uptake

**Community Project**

- >750 registered users
- 90 Institutions
- Every continent
  - (-Antarctica)

**Kokkos Slack Channel**

- 25 Developers
  - 15 for Core
- Regular contributions from HPC vendors

**Applications and Libraries**

- Estimated 150-250 HPC projects using Kokkos
- On the order of two-dozen apps run science and engineering production runs with Kokkos
- Similar distribution as the Slack User
- In ECP: Almost as many projects require Kokkos as projects require Fortran (18-21)
How Does Kokkos Work?
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(MDRRangePolicy<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
- UnorderedMap
- kokkos_malloc
- kokkos_free
- Vector
- Bitset
- LayoutLeft
- LayoutRight
- LayoutStrided
- UniqueToken
- ScratchSpace
- 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);
```
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, View<const double*> x,
            double beta, View<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;
}
```
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_row_offsets);

    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;
        });
    });
}
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
CG Solve Performance Today

- CG-Solve as discussed above
- Also try replacing SPMV with TPL
- Running 100x100x100 heat conduction problem
  - "MiniFE" Proxyapp setup
- Measure effective Bandwidth
  - Algorithmical memory ops per time
- Why is this beating vendor libs?
  - Its complicated, but a real effect
The EcoSystem
Kokkos Support

- The Kokkos Lectures
  - 8 lectures covering most aspects of Kokkos
  - 17 hours of recordings
  - > 500 slides
  - >20 exercises
- Extensive Wiki
  - API Reference
  - Programming Guide
- Slack as primary direct support

https://kokkos.link/the-lectures

- Module 1: Introduction
  - Introduction, Basic Parallelism, Build System
- Module 2: Views and Spaces
  - Execution and Memory Spaces, Data Layout
- Module 3: Data Structures and MDRangePolicy
  - Tightly Nested Loops, Subviews, ScatterView,…
- Module 4: Hierarchical Parallelism
  - Nested Parallelism, Scratch Pads, Unique Token
- Module 5: Advanced Optimizations
  - Streams, Tasking and SIMD
- Module 6: Language Interoperability
  - Fortran, Python, MPI and PGAS
- Module 7: Tools
  - Profiling, Tuning, Debugging, Static Analysis
- Module 8: Kokkos Kernels
  - Dense LA, Sparse LA, Solvers, Graph Kernels
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

```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);
// 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
  - 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
More Information

- [https://kokkos.link/the-lectures](https://kokkos.link/the-lectures)
  - ~17 hours of recorded lectures with hands-on exercise homework
- [https://github.com/kokkos](https://github.com/kokkos)
  - The Kokkos EcoSystem Organization with all the repos and documentation
- [https://kokkosteam.slack.com](https://kokkosteam.slack.com)
  - Slack channel for the Kokkos community with over 750 members

Papers:

The Kokkos EcoSystem: Comprehensive Performance Portability For High Performance Computing
C.R. Trott et al., Computing in Science & Engineering, 2021

Kokkos 3: Programming Model Extensions for the Exascale Era
C.R. Trott et al., IEEE Transactions on Parallel and Distributed Systems, 2021

Kokkos: Enabling manycore performance portability through polymorphic memory access patterns
H.C. Edwards et al., Journal of Parallel and Distributed Computing, 2014
Backup Slides
Cost of Porting Code

- 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

10 LOC / hour ~ 20k LOC / year
Transitioning To Community Project

- **Kokkos Core**: 15 Developers (8 SNL)
- More code contributions from non-SNL
  - >50% of commits from non-Sandians
- Sandia leads API design
- Other labs lead backend implementations
- Other subprojects largely by Sandia so far

Papers:

The Kokkos EcoSystem: Comprehensive Performance Portability For High Performance Computing
C.R. Trott et al., Computing in Science & Engineering, 2021

Kokkos 3: Programming Model Extensions for the Exascale Era
C.R. Trott et al., IEEE Transactions on Parallel and Distributed Systems, 2021

Kokkos: Enabling manycore performance portability through polymorphic memory access patterns
H.C. Edwards et al., Journal of Parallel and Distributed Computing, 2014
Kokkos Uptake

ECP Critical Dependencies

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>60</td>
</tr>
<tr>
<td>LLVM</td>
<td>57</td>
</tr>
<tr>
<td>C++</td>
<td>41</td>
</tr>
<tr>
<td>OpenMP</td>
<td>34</td>
</tr>
<tr>
<td>LAPACK</td>
<td>24</td>
</tr>
<tr>
<td>CUDA</td>
<td>22</td>
</tr>
<tr>
<td>Fortran</td>
<td>21</td>
</tr>
<tr>
<td>HDF5</td>
<td>21</td>
</tr>
<tr>
<td>BLAS</td>
<td>21</td>
</tr>
<tr>
<td>Kokkos</td>
<td>18</td>
</tr>
<tr>
<td>C</td>
<td>14</td>
</tr>
<tr>
<td>ALPINE</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>hypre</td>
<td>11</td>
</tr>
<tr>
<td>DAV-SDK</td>
<td>11</td>
</tr>
<tr>
<td>VTK-m</td>
<td>11</td>
</tr>
<tr>
<td>Trilinos</td>
<td>10</td>
</tr>
<tr>
<td>ADIOS</td>
<td>8</td>
</tr>
<tr>
<td>SPACK</td>
<td>8</td>
</tr>
<tr>
<td>SCALAPACK</td>
<td>8</td>
</tr>
<tr>
<td>FFT</td>
<td>7</td>
</tr>
<tr>
<td>OpenACC</td>
<td>7</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>6</td>
</tr>
<tr>
<td>PnetCDF</td>
<td>6</td>
</tr>
<tr>
<td>Tau</td>
<td>6</td>
</tr>
</tbody>
</table>

Kokkos Slack Users

- 660 registered users
- 90 Institutions
- Every continent
  - (-Antarctica)

Other

SNL

DOE (not SNL)

Universities

Total membership

Weekly active members
Tracking New Capabilities: Graphs

- Build static graphs of kernels
  - Can use CUDAGraphs as backend
  - Allows repeated dispatch
- Helps with Latency Limited codes
  - Cuts down on launch latency
  - Can leverage streams to overlap work
  - Infers overlapping from dependencies
- Prototype release part of Kokkos 3.3

```cpp
const auto graph = Kokkos::Experimental::create_graph([=](auto root) {
    auto f1 = root.then_parallel_for(
        Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
    auto f2a = f1.then_parallel_for(
        Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
    auto f2b = f1.then_parallel_for(
        Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
    when_all(f2a, f2b).then_parallel_reduce(
        Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…}
        result);
});

while(result() > threshold {
    graph.submit();
    graph.get_execution_space().fence();
}
```
**Benchmark the Example**

**Solid: Graphs**

**Dashed: Simple Dispatch**

**Can reuse graph:**
- In solver iterations
- Between solves if matrix structure unchanged
- >100 reuses could be realistic

**Throughput Improvement:**
- 50K 78%
- 200k 49%
- 1M 15%

Next: look at reducing graph creation time
**AMD Support Status**

**Frontier/El Capitan: HIP and OpenMP 5**
- Primary development of HIP at ORNL
- Most Capabilities ready
  - Fine grained tasking is missing
- PR testing for Kokkos on AMD GPUs in place
- ArborX, Cabana, LAMMPS working with HIP
- Trilinos (4,000k lines HPC library) works.

We are largely using our own machines (not ECP EAS), with the public software stack from Intel and AMD.

*Kokkos 3.3 (Dec 2020):*
- HIP is largely feature complete

*Kokkos 3.4 (April 2021):*
- SYCL Support Largely Complete

Kokkos Core functionality porting to Frontier nearly complete
Aurora Support Status

Programming Models: DPC++/SYCL + OpenMP 5

- Primary work for DPC++ at ANL and ORNL
  - Shifted ORNL team members from HIP to DPC++ since HIP is in much better shape

- DPC++/SYCL was long blocked by compiler issues
  - Worked with Intel to get those fixed
  - Now primary capabilities are merged to develop branch

- PR testing DPC++/SYCL in place
  - Intel DPC++/SYCL testing is done on NVIDIA GPUs …
  - Leverages clang capability to target different backend

We are largely using our own machines (not ECP EAS), with the public software stack from Intel and AMD.

Kokkos 3.3 (Dec 2020):
- OpenMPTarget and DPC++ have most primary capabilities working

Kokkos 3.4 (April 2021):
- DPC++/SYCL is largely feature complete

Initial Kokkos Core functionality porting to Aurora done.
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>190</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_recv()</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_ParallelRegion parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</td>
<td>0</td>
<td>0.504</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>OpenMP_ParallelRegion parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</td>
<td>0.08</td>
<td>0.968</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>OpenMP_ParallelRegion void Kokkos::parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</td>
<td>0.001</td>
<td>0.594</td>
<td>380</td>
<td>380</td>
</tr>
<tr>
<td>OpenMP_SyncRegion_Barrier parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</td>
<td>0.489</td>
<td>0.489</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>OpenMP_SyncRegion_Barrier parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</td>
<td>0.875</td>
<td>0.875</td>
<td>190</td>
<td>0</td>
</tr>
<tr>
<td>OpenMP_SyncRegion_Barrier void Kokkos::parallel_for&lt;Kokkos::RangePolicy&lt;Comm_MPI::Ta</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
Tooling and Remote Spaces

- Tools Support is growing
  - More Native support e.g. Tau
- Connectors to Timemory etc.
- Nsight Systems does more useful stuff with connectors
  - Rename Kernels

- Remote Spaces beta now released
  - https://github.com/kokkos/kokkos-remote-spaces
- Support for NVSHMEM, MPI, SHMEM
- Working on Caching, aggregation etc.
- Potentially huge productivity benefits
New Capabilities: Auto Tuning

- Part of Kokkos 3.2
- Tuning Interface + Tools
  - Same as other hooks: they are always there, but act as no-ops without a tool
- Multi Input – multi Output tuning
  - Inputs describe problem space
  - OutputTypes describe search space
    - Sets, Ranges, Categorical
    - Logarithmic, linear
- Tuning scopes can include multiple kernels
- Tuning of internal variables in 3.3 or 3.4

**Apollo Tuner for SPMV tuning:**
- Rows per team
- Team Size
- Vector Length
New Capabilities: Static Analysis

- Can we catch violations of Kokkos semantics even if code would compile/run?
  - kokkos-llvm: fork of LLVM with Kokkos aware clang-tidy variant
- Three types of violating patterns:
  - compile with some backends but not others.
  - run correct with some backends but crash on others.
  - run correct with some backends but have wrong results with others!

Example: Missing function markup

```c
void fooOOPS(int i) { printf("%i\n", i); }

int main(int argc, char **argv) {
    Kokkos::initialize();
    Kokkos::parallel_for(15, KOKKOS_LAMBDA(int i) {
        fooOOPS(i);
    });
    Kokkos::finalize();
}
```

```
> clang-tidy -checks=-*,kokkos-* file.cpp
<main.cpp:7:5> warning: Function 'fooOOPS' called in a lambda was missing KOKKOS_X_FORCE_FUNCTION annotation.
    fooOOPS(i);
^*
<main.cpp:2:1> note: Function 'fooOOPS' was declared here
void fooOOPS(int i) { printf("%i\n", i); }
```
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
**Kokkos - C++ Standard integration cycle**

**Kokkos**
- Propose new features for C++ Standard
- Back port to current compilers

**Kokkos Legacy**
- Port accepted features to legacy versions
- Implemented legacy capabilities in terms of new C++ features

**C++ Standard**

**C++ Backport**
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)
OpenMPTarget Status

- Most capabilities are now working
  - Until earlier in 2020 limited by compiler bugs
- Using primarily main line clang/llvm
  - Are also working with Intel and NVIDIA
  - Started working with AMD and HPE
- Next phase: concentrating on performance
  - C++ performance very fragile
  - We are ramping up collaboration with compiler engineers

Vector Add Performance Illustration

- Simple problem, should clearly be bandwidth limited
- Using clang/llvm 11, CUDA 10.1, NVIDIA V100
- Kokkos/CUDA (kk-c), Kokkos/OMPT (kk-o), Native OMPT (omp), Native OMPT with temporaries (omp-t)

```c
struct Foo {
  int N;
  double *x, *y, *z;
  void axpby() {
    // Need temporaries here for 4x performance gain
    int N_ = N;
    double *xp = x, *yp = y, *zp = z;
    #pragma omp target teams distribute parallel for 
      simd is_device_ptr(xp,yp,zp) data map(to: N_)
    for(int i=0; i<N_; i++) {
      zp[i] = xp[i] + yp[i];
    }
  }
};
```

```c
struct Foo {
  View<double*> x,y,z;
  int N;
  void axpby() {
    parallel_for("axpby", N, 
    KOKKOS_LAMBDA(int i) {
       z(i) = x(i) + y(i);
    });
  }
};
```

OpenMP Vector Add

```c
struct Foo {
  int N;
  double *x, *y, *z;
  void axpby() {
    // Need temporaries here for 4x performance gain
    int N_ = N;
    double *xp = x, *yp = y, *zp = z;
    #pragma omp target teams distribute parallel for 
      simd is_device_ptr(xp,yp,zp) data map(to: N_)
    for(int i=0; i<N_; i++) {
      zp[i] = xp[i] + yp[i];
    }
  }
};
```

Kokkos Vector Add

```c
struct Foo {
  View< double* > x,y,z;
  int N;
  void axpby() {
    parallel_for("axpby", N, 
      KOKKOS_LAMBDA(int i) {
        z(i) = x(i) + y(i);
      });
  }
};
```

Takeaway: Performance is still very fragile!
A more comprehensive Frontend/Compiler comparison

• Comparing simple vector add and dot product
  – Also implemented straight forward native implementation
  – No hoops jumped through to optimize
  – 1M length, not huge, but also not trivial, i.e. latency impact expected but not dominant?
    • If purely bandwidth bound this would be 24us for axpby@1TB/s and 16us for dot
  – clxx denotes clang/llvm version