Kokkos Status 2019

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.
Kokkos EcoSystem
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, J. Miles, D. Hollman, V. Dang, Mikael Simberg  
soon: H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin  
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, J. Wilke, W. McLendon

**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
Some Kokkos Stats Since 2015

- 17 Releases Since 2016
  - Only 4 since December 2017
- 50 Contributors
  - 17 with more than 10 commits
  - 11 with more than 10k lines touched
- 1345 Issues of which 1134 were resolved
  - 305 bug reports
  - 381 enhancement requests
  - 129 Feature Requests
- 766 pull requests
- 15k messages on kokkosteam.slack.com (Started in 2017)
<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>
<tr>
<td>Parallel Reduction</td>
<td><code>parallel_reduce( RangePolicy&lt;ExecSpace&gt;(0,N), KOKKOS_LAMBDA (int i, double&amp; upd) { ...BODY... upd += ... }, Sum&lt;&gt;(result));</code></td>
</tr>
<tr>
<td>Tightly Nested Loops</td>
<td><code>parallel_for(MDRangePolicy&lt;Rank&lt;3&gt; &gt; (0,0,0),N1,N2,N3),T1,T2,T3, KOKKOS_LAMBDA (int i, int j, int k) { ...BODY... });</code></td>
</tr>
<tr>
<td>Non-Tightly Nested Loops</td>
<td><code>parallel_for( TeamPolicy&lt;Schedule&lt;Dynamic&gt;&gt;( N, TS ), KOKKOS_LAMBDA (Team team) { ... COMMON CODE 1 ... parallel_for(TeamThreadRange(team, M(N)), [&amp;] (int j) { ... INNER BODY... }); ... COMMON CODE 2 ... });</code></td>
</tr>
<tr>
<td>Task Dag</td>
<td><code>task_spawn( TaskTeam( scheduler , priority), KOKKOS_LAMBDA (Team team) { ... BODY });</code></td>
</tr>
<tr>
<td>Data Allocation</td>
<td><code>View&lt;double**, Layout, MemSpace&gt; a(&quot;A&quot;,N,M);</code></td>
</tr>
<tr>
<td>Data Transfer</td>
<td><code>deep_copy(a,b);</code></td>
</tr>
<tr>
<td>Atomics</td>
<td><code>atomic_add(&amp;a[i],5.0); View&lt;double*,MemoryTraits&lt;AtomicAccess&gt;&gt; a(); a(i) += 5.0;</code></td>
</tr>
<tr>
<td>Exec Spaces</td>
<td><code>Serial, Threads, OpenMP, Cuda, HPX (experimental), ROCm (experimental)</code></td>
</tr>
</tbody>
</table>
TeamVectorRange

- Fix situations with mix of 2-level and 3-level hierarchical parallelism
  - Now in develop!

```cpp
parallel_for("BigKernel", TeamPolicy<>\((N,AUTO,8) KOKKOS_LAMBDA (const team_t& team) {
    parallel_for(TeamVectorRange(team,M), [&] (const int j) {
        // Fill Buffer
    });
    //...
    parallel_for(TeamThreadRange(team,M), [&](const int j) {
        //...
        parallel_for(ThreadVectorRange(team,K), [&] (const int k) {
            //...
        });
    });
    //...
});
```
HPX Backend

- HPX (LSU/CSCS implementation) is a task based programming model in C++
  - Completely Asynchronous
  - Tries to align with C++ standard interface wise
- Goal: production use by end of FY19
  - CSCS will maintain this
- Benefits for general Kokkos users:
  - First asynchronous Host backend
    - Find synchronization issues in your code
  - Much easier to align with future directions of Kokkos
Configuration / Runtime Management

- Environment Variables: KOKKOS_NUM_THREADS=int, KOKKOS_NUMA=int, KOKKOS_DEVICE_ID=int, KOKKOS_NUM_DEVICES=int, KOKKOS_SKIP_DEVICE=int, KOKKOS_DISABLE_WARNINGS=bool

- hpcbind: command line tool to partition node, set environment variables, visible gpus, and control stdout and mpi output files (see hpcbind --help)
  - Example: launch 16 jobs over 4 nodes with 4 jobs per and save output
    mpiexec -N 16 -nperrnode 4 hpcbind --whole-system
    --distribute=4 --output-prefix=out -- executable [args]

- C++14/17/2a support
  - Backend support is compiler dependent (for example Cuda does not support C++17/2a)
Reducers

- Common Reduction types are now provided by Kokkos:
  Sum, Prod, Min, Max, Land, Lor, Band, Bor, VallocScalar, MinLoc, MaxLoc, MinMaxScalar, MinMax, MinMaxLocScalar, MinMaxLoc

- Example:
  View<double*> v("view", N);
  ...
  double sum = 0;
  parallel_reduce(n, [=](int i, double &value) {
    value += v[i];
  }, Sum<double>(sum));
Asynchronicity Semantics

- **ParallelReduce/Scan**

```cpp
double result;
// parallel_for is always Synchronous
parallel_for("AsynchronousFor", N, F);
// parallel_reduce with Scalar as result is Synchronous
parallel_reduce("SynchronousSum", N, F, result);
// parallel_reduce with Reducer constructed from scalar is synchronous
parallel_reduce("SynchronousMax", N, F, Max<double>(result));
// parallel_reduce with any type of View as result is asynchronous
Kokkos::View<double, CudaHostPinnedSpace> result_v("R");
parallel_reduce("AsynchronousSum", N, F, result_v);
// Even with unmanaged view, and wrapped into Reducer
Kokkos::View<double, HostSpace> result_hv(&result);
parallel_reduce("AsynchronousMax", N, F, Max<double>(result_hv));
// Scans without total result argument are asynchronous
parallel_scan("AsynchronousScan", N, Fs);
// Scans with total result argument same rules as parallel_reduce
parallel_scan("SynchronousScanTotal", N, Fs, result);
```

---

![Graph showing 2 Dot Products with N=100k](image)
CUDA Stream Interop

- Initial step to full coarse grained tasking
  - Discuss in more detail in future directions
- For now: make Kokkos dispatch use user CUDA streams
  - Allows for overlapping kernels: best for large work per iteration, low count

```c
// Create two Cuda instances from streams
cudaStream_t stream1,stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
Kokkos::Cuda cuda1(stream1), cuda2(stream2);

// Run two kernels which can overlap
parallel_for("F1",RangePolicy<Kokkos::Cuda>(cuda1,N),F1);
parallel_for("F2",RangePolicy<Kokkos::Cuda>(cuda2,N),F2);
fence();
```
MDRangePolicy

- Multi-index for parallel kernels of tightly nested loops
- Only supported for `parallel_for/parallel_reduce`
  Ex: `parallel_for(MDRangePolicy<Rank<3>,..., (...), [=](i, j, k) {...});`
- Can parallelize over all the dimensions of the loop
- Allows tiled iteration patterns for improved cache/warp memory access
UniqueToken

- Generates a unique ordinal based on the concurrency of the `ExecutionSpace`
  - Can be used to index into resources that are restricted by the amount of concurrency available
- Ordinals can be `local` to a single kernel instance or `global` across all kernels
- Threads first `acquire` a token and then `release` it afterwards
- For the best performance
  - Tokens should be acquired/released in as narrow of scope as possible, and
  - Tokens should be released before calling a `team_barrier` or similar construct
View Improvement

- **LayoutTiled**: Data is contiguous over *tiles*, i.e., multi-dimensional bricks. Tiled dimensions must be powers of two.

- **Anonymous Memory Space**: Allows views to assume that they can always access the memory.
  - The user is responsible for ensuring that the view only accesses data when on a device that can dereference the underlying pointer.
  - Can reduce the number of template parameters needed for a kernel.
  - Can reduce the number of symbols created during compile time.
Kokkos Containers

- **DualView**: Allocate and manage a view on both the host and device. Added non-templated sync functions `sync_host()` and `sync_device()`.
- **OffsetView**: Allows views indices to start at non-zero values.
- **ErrorReporter**: Count number of errors and report the first n messages.
- **StaticCrsGraph**: Compressed row storage data structure. The storage structure is static after construction.
- **UnorderedMap**: Performance portable hash_map/hash_set.
Containers: ScatterView

- Encapsulates common design pattern in reduction algorithms using either data duplication and/or atomics
  - Data duplication is often faster on the host, but too memory expensive on GPUs.
  - Atomics are faster on GPUs, but extremely slow on the host

ScatterView<Datatype[, Layout, ExecSpace, ReduceOp, DupMode, ContribMode]>

ReduceOp: ScatterSum, ScatterProd, ScatterMax, ScatterMin
DupMode: ScatterNonDuplicated, ScatterDuplicated
ContribMode: ScatterNonAtomic, ScatterAtomic
Containers: ScatterView (cont’d)

ScatterView<double, LayoutRight, Cuda, ScatterSum, ...> sv(...);
View<double, LayoutRight, Cuda> v(...);

parallel_for(n, [=](int i){
    auto scatter_access = sv.access();
    int k = foo(i);
    double x = bar(x);
    scatter_access(k) += x;
});

contribute(v, sv);
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:

```cpp
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!!