Kokkos – Performance Portability Today

Christian Trott, Carter Edwards, Nathan Ellingwood, Si Hammond
crtrott@sandia.gov
Center for Computing Research
Sandia National Laboratories, NM
Kokkos: Performance, Portability and Productivity

https://github.com/kokkos
Performance Portability through Abstraction

Separating of Concerns for Future Systems...

Kokkos

Data Structures

Memory Spaces (“Where”)
- Multiple-Levels
- Logical Space (think UVM vs explicit)

Memory Layouts (“How”)
- Architecture dependent index-maps
- Also needed for subviews

Memory Traits
- Access Intent: Stream, Random, …
- Access Behavior: Atomic
- Enables special load paths: i.e. texture

Parallel Execution

Execution Spaces (“Where”)
- N-Level
- Support Heterogeneous Execution

Execution Patterns (“How”)
- parallel_for/reduce/scan, task spawn
- Enable nesting

Execution Policies
- Range, Team, Task-Dag
- Dynamic / Static Scheduling
- Support non-persistent scratch-pads
Going Production

- Robust Compilation and Environment Testing
  - Nightly test of 12 Compilers (GCC, Intel, Clang, NVCC)
  - >100 total configurations
  - Warnings as errors with “–pedantic -Wall –Wshadow .....”

- Documentation and User Training
  - Programming Guide
  - Extensive Tutorials: [https://github.com/kokkos/kokkos-tutorials](https://github.com/kokkos/kokkos-tutorials)
    - > 300 Slides, dozens of hands-on examples with solutions
    - Under discussion: cloud based self-learning labs (used at GTC 2016)

- Profiling and Debugging Tools Integration
  - Talk by Simon Hammond (SNL) later this week

- Production and Next Generation Applications
  - ATDM targeting KNL and GPUs from beginning (talk by Stan Moore)
  - Sierra Mechanics focusing on thread safety/scalability until late 2017
Managing Memory Hierarchies

- Memory Hierarchies At **Low-Level** are both Physical and Logical
  - Example current x86 + NVIDIA GPU with allocations in
    - UVM Space: let the CUDA runtime handle data transfer
    - CUDA Space: I know what I am doing, leave the runtime out of the way
    - HostPinned Space: I want to use asynchronous mem-copy with DMA engine

- Kokkos gives tools to do **Low-Level** management

- Applications write/use customizations for higher level management
  - Example LAMMPS:
    - Physics modules provide bit-masks for read/write access of fields
    - Memory management in LAMMPS uses Kokkos API to get data where it needs to be, including asynchronous copies

- Open: can we come up with generic **High-Level** Interfaces
  - Are use cases similar enough?
  - Multi-Lab CoE Talk by Ian Karlin (LLNL)
LAMMPS – Heterogeneous Execution

Reverse Offload

- LAMMPS/example/accelerate/in.phosphate
- Goal overlap Pair and Kspace
  - Requires Asynchronous Deep Copy
- When Overlapping:
  - Comm contains pair time since it fences to wait for pair force
  - 96% of Kspace time reduction

Wall Time

<table>
<thead>
<tr>
<th></th>
<th>Modify</th>
<th>Neigh</th>
<th>Kspace</th>
<th>Comm</th>
<th>Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoOverlap</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Overlap</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.05</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Managing Access Patterns

- Change Data Access Pattern
  - Single typedef per code
- Adapt to Architectures
- Custom Layouts Easy
- Example SIMD friendly storage
  - Support explicit vector types for some kernels
  - `View<float*[3],LayoutRight>`
  - `View<float*[3],LayoutSIMDRight>`
  - `View<float2*[3],LayoutRight>`
- Before: `a[i%V + j*V + (i/V)*V*3]`  Now: `a(i,j)`
Example Atomic Support

- Avoiding write conflicts comes with cost
  - **Coloring:** code complexity (for unstructured), potentially much more memory traffic (only parts of each cache line are used per color), loss of concurrency
  - **Data Replication:** memory footprint / traffic, additional reduction, less cache efficient
  - **Atomics:** serialization, loss of vectorization, potentially loss of L1 caching
  - **Compute Replication:** more flops / iops, more memory traffic

- Kokkos is used with all methods

- For **unstructured** problems atomics are often preferred over other approaches
  - `View<double**, MemoryTraits<Atomic> > a_atomic = a;`
Avoiding write conflicts comes with cost

Coloring:
- code complexity (for unstructured), potentially much more memory traffic (only parts of each cache line are used per color), loss of concurrency

Data Replication:
- memory footprint / traffic, additional reduction, less cache efficient

Atomics:
- serialization, loss of vectorization, potentially loss of L1 caching

Compute Replication:
- more flops / iops, more memory traffic

Kokkos is used with all methods

For unstructured problems atomics are often preferred over other approaches

```
View<
  double**,
  MemoryTraits<
    Atomic
  > > a_atomic
```

```
for(i=0,N) atomic_add( a[i%M], one )
```
Under development: KokkosKernels

- Provide BLAS (1,2,3); Sparse; Graph and Tensor Kernels
- No required dependencies other than Kokkos
- Local kernels (no MPI)
- Hooks in TPLs such as MKL or cuBLAS/cuSparse where applicable
- Provide kernels for all levels of hierarchical parallelism:
  - Global Kernels: use all execution resources available
  - Team Level Kernels: use a subset of threads for execution
  - Thread Level Kernels: utilize vectorization inside the kernel
  - Serial Kernels: provide elemental functions (OpenMP declare SIMD)
- Work started based on customer priorities; expect multi-year effort for broad coverage
- People: Many developers from Trilinos contribute
  - Consolidate node level reusable kernels previously distributed over multiple packages
SPMV Benchmark: MKL vs Kokkos
1S HSW 24 Threads, Matrices sorted by size, Matrices obtained from UF

On GPUs: CuSparse vs Kokkos: All: 1.07  Without Small: 0.84
The Common Problems We Face

- **Interference with Compiler Optimizations**
  - Deducing Independence of Views: `restrict` for pointer as class members?
  - Hoisting loads from inner loops, with that being `parallel_for`
  - Loosing “const” when creating lambdas with capture by reference
  - Generally loosing surrounding information when using Lambdas

- **Deficiencies in C++ Language for threading models**
  - *this capture for Lambdas in member functions
    - Part of C++17
    - Enables asynchronous dispatch
    - Added to Clang 3.9
  - Error handling in threaded environments
  - OpenMP 4 handling of classes
  - What to do with STL objects
The Way Forward

- Stabilize Kokkos Capabilities
  - Support tasking on all platforms
  - Make sure compilers optimize through layers
  - Harden KNL support for High Bandwidth Memory
- Broaden Implementation Coverage for Kokkos Kernels
- Support Production Teams in Adoption
- Develop even more Documentation
- Extend profiling tools to help with transition

www.github.com/kokkos/kokkos: Kokkos Core Repository
www.github.com/kokkos/kokkos-tutorials: Kokkos Tutorial Material
www.github.com/kokkos/kokkos-tools: Kokkos Profiling Tools
www.github.com/trilinos/Trilinos: Trilinos Repository
NALU Assembly

- Uses atomic operations to fill into matrix

simple heat conduction; 262,144 elements; run on Shepard (Haswell, 16C/32T per socket)
SPMV Benchmark: CuSparse vs Kokkos

K40c Cuda 7.5; Matrices sorted by size; Matrices from UF.

- Heuristic (1.378)
- Auto Params (1.067 [0.84])