Kokkos:
Performance Portability and Productivity for Next Generation HPC

Workshop on Exascale Software Technology

H. Carter Edwards

January 27-28, 2016
Albuquerque, NM
SAND2016-0648 PE
Takeaway

Applications & Libraries

Kokkos
performance portability for C++ applications

LAMMPS
Albany
Drekar

EMPRESS
SPARC

Trilinos

Multi-Core
Many-Core
APU
CPU+GPU

Drekar
Trilinos

SPARC

Albany

LAMMPS
What is Kokkos?

- **ΚΟΚΚΟΣ** (Greek, not an acronym)
  - Translation: “granule” or “grain”; *like grains of sand on a beach*

- **Performance Portable Thread-Parallel Programming Model**
  - E.g., “X” in “MPI+X”; **not** a distributed-memory programming model
  - Application identifies its parallelizable grains of *computations and data*
  - Kokkos maps those computations onto cores *and* that data onto memory

- **Fully Performance Portable C++11 Library Implementation**
  - **Production** – open source at [https://github.com/kokkos/kokkos](https://github.com/kokkos/kokkos)
  - ✔ **Multicore CPU** - including NUMA architectural concerns
  - ✔ **Intel Xeon Phi (KNC)** – testbed prototype toward Trinity / ATS-1
  - ✔ **NVIDIA GPU (Kepler)** – testbed prototype toward Sierra / ATS-2
  - ✦ **IBM Power 8** – testbed prototype toward Sierra / ATS-2
  - ✦ **AMD Fusion** – via collaboration with AMD

  ✔ Regularly and extensively tested
  ✦ Ramping up testing
Some Collaborations

- Sandia: ASC / ATDM, IC, CSSE, and PEM
  - Integral for performance portability to next generation platforms (NGPs)
- LANL: ASC/ATDM exploring Legion/Kokkos integration
- ORNL: Exploring for SHIFT using Kokkos
- LLNL: programming model discussions
- Universities and other HPC research labs (US Army, Swiss, ...)
- Vendors: DOE FastForward & DesignForward
  - NVIDIA – evaluating and influencing new CUDA C++ features
  - PGI – consulting to improve OpenACC/C++ integration
  - IBM – target new generation xlc compiler
  - AMD – target for HCC compiler
- ISO/C++ Standards Committee
Abstractions: Patterns, Policies, and Spaces

- **Parallel Pattern** of user’s computations
  - parallel_for, parallel_reduce, parallel_scan, task-graph, ... *(extensible)*

- **Execution Policy** tells *how* user computation will be executed
  - Static scheduling, dynamic scheduling, thread-teams, ... *(extensible)*

- **Execution Space** tells *where* user computations will execute
  - Which cores, numa region, GPU, ... *(extensible)*

- **Memory Space** tells *where* user data resides
  - Host memory, GPU memory, high bandwidth memory, ... *(extensible)*

- **Layout** (policy) tells *how* user data is laid out in memory
  - Row-major, column-major, array-of-struct, struct-of-array ... *(extensible)*

- **Differentiating**: Layout and Memory Space
  - Versus other programming models (OpenMP, OpenACC, ...)
  - Critical for performance portability ...
Layout Abstraction: Multidimensional Array

- Classical (50 years!) data pattern for science & engineering codes
  - Computer languages hard-wire multidimensional array layout mapping
  - Problem: different architectures require different layouts
    - Leads to architecture-specific versions of code to obtain performance
  - E.g., “Array of Structure” ↔ “Structure of Array” redesigns

- Kokkos separates layout from user’s computational code
  - Choose layout for architecture-specific memory access pattern
    - Without modifying user’s computational code
  - Polymorphic layout via C++ template meta-programming (extensible)
    - e.g., Hierarchical Tiling layout

- Bonus: easy/transparent use of special data access hardware
  - Atomic operations, GPU texture cache, ... (extensible)
Performance Impact of Data Layout

- Molecular dynamics computational kernel in miniMD
- Simple Lennard Jones force model:
  \[ F_i = \sum_{j, r_{ij} < r_{cut}} 6 \epsilon \left( \left( \frac{\varsigma}{r_{ij}} \right)^7 - 2 \left( \frac{\varsigma}{r_{ij}} \right)^{13} \right) \]
- Atom neighbor list to avoid \( N^2 \) computations

```javascript
pos_i = pos(i);
for (jj = 0; jj < num_neighbors(i); jj++) {
    j = neighbors(i, jj);
    r_ij = pos(i,0..2) - pos(j,0..2); // random read 3 floats
    if (|r_ij| < r_cut) f_i += 6*e*((s/r_ij)^7 – 2*(s/r_ij)^13);
}
f(i) = f_i;
```

- Test Problem
  - 864k atoms, ~77 neighbors
  - 2D neighbor array
  - Different layouts CPU vs GPU
  - Random read ‘pos’ through GPU texture cache
- Large performance loss with wrong data layout
Performance Portability & Future Proofing

Integrated mapping of users’ parallel computations *and* data through abstractions of patterns, policies, spaces, *and* layout.

- **Versus other thread parallel programming models (mechanisms)**
  - OpenMP, OpenACC, OpenCL, ... have parallel execution
  - OpenMP 4 finally has execution spaces; when memory spaces ??
  - All of these neglect data layout mapping
    - Requiring significant code refactoring to change data access patterns
    - Cannot provide *performance* portability
  - All require language and compiler changes for extension

- **Kokkos extensibility “future proofing” wrt evolving architectures**
  - Library extensions, not compiler extensions
  - E.g., DOE/ATS-1 high bandwidth memory ← just another memory space
Performance Overhead?

Kokkos is competitive with other programming models

- Regularly performance-test mini-applications on Sandia’s ASC/CSSE test beds
- MiniFE: finite element linear system iterative solver mini-app
  - Compare to versions with architecture-specialized programming models
Simple and Incremental to Adopt

- **Step 1: Replace loops with parallel patterns**
  - Default Execution Space and Memory Space are CPU
  - Default Execution Policy is [0..N)

- **Example sparse matrix-vector multiply:**
  - Original Serial version:
    ```c
    for ( int i = 0 ; i < nrow ; ++i ) {
        for ( int j = irow[i] ; j < irow[i+1] ; ++j )
            y[i] += A[j] * x[ jcol[j] ];
    }
    ```
  - Kokkos parallel version:
    ```c
    parallel_for( nrow , KOKKOS_LAMBDA( int i ) {
        for ( int j = irow[i] ; j < irow[i+1] ; ++j )
            y[i] += A[j] * x[ jcol[j] ];
    });
    ```

- **Challenge: Find and Fix thread-unsafe code**
  - Required to adopt any thread-parallel programming models
  - Inter-thread race conditions: use Kokkos’ atomic operations
  - Serialization performance bottlenecks in algorithm: design new algorithms

- **Step 2: Identify Spaces for execution and data**
Incremental to Portably Optimize

- **Step 3: Introduce Hierarchical Parallelism as needed**
  - When simple [0..N) parallel execution policy is insufficient for performance
  - Optimize those computations with “Thread Team” execution policy

- **Example sparse matrix vector multiply has nested loops**
  - Kokkos simple parallel version:
    ```cpp
    parallel_for( nrow , KOKKOS_LAMBDA( int i ) {
        for ( int j = irow[i] ; j < irow[i+1] ; ++j )
            y[i] += A[j] * x[ jcol[j] ];
    });
    ```
  - Kokkos hierarchical parallel version ( #Teams x #Threads/team )
    ```cpp
    parallel_for( TeamPolicy( nrow ),
                 KOKKOS_LAMBDA( TeamPolicy::member_type const & member ) {
        double result = 0 ;
        const int i = member.league_rank();
        parallel_reduce( TeamThreadRange( member, irow[i], irow[i+1]) ,
                         [&]( int j , double & val ) { val += A[j] * x[jcol[j]];},
                         result );
        if ( member.team_rank() == 0 ) y[i] = result ;
    });
    ```

- **Step 4: Tune multidimensional array data layout as needed**
Key Research, Development, and Support

- **Evolve back-ends for new & changing node architectures**
  - Stable abstractions to access new hardware capabilities (e.g., KNL HBM)
  - R&D, co-design, collaborate to measure and optimize back-ends

- **Extend patterns, policies, spaces, layout**
  - Dynamic scheduling (work stealing) execution policies
  - Multidimensional range policies (parallel “loop collapse”)
  - Tiling and other specialized layout mappings
  - Dynamically resizable arrays - thread-scalable within parallel operations
  - Directed acyclic graph (DAG) of “fine grain” tasks execution pattern/policy
    - Mature and harden internal R&D prototype
  - Remote execution and memory spaces

- **R&D for portable embedded performance instrumentation**

- **Application developer support, is a resource concern...**
  - Tutorials (SC’15, GTC’16), documentation, interactions, feature requests, ...
  - Teaching & consulting for thread-scalable algorithmic patterns & practices
Conclusion

- Integral to SNL / ASC plans for NGP performance portability
- Application developer support is a resource concern
  - ASC program elements, DOE labs, universities, other HPC research labs
- Compared to other programming models
  - They fail to address layout and thus limit performance portability
  - Extensibility (future-proofing) via library extensions vs. compiler extensions
- Strategic collaborations
  - Vendors FastForward, DesignForward, co-design, NGP testbeds
  - PSAAPII Universities
  - ISO/C++ : 2020 standard fully addresses heterogeneous node parallelism
    - Voting block of HPC advocates: SNL, ANL, LANL, LLNL, LBL, ...
- Productivity Assessment: FY15 Co-Design L2 Milestone
  - No harder than OpenMP to adopt; easier to portably optimize performance