Kokkos Port of CoMD Mini-App

DOE COE Performance Portability Meeting 2017
Denver, Colorado

David Gunter
Toks Adedoyin
22 August 2017
Outline

• Background Motivation
• Molecular Dynamics / CoMD proxy application
• Kokkos Library
• Porting Effort
• Performance Results
• Lessons Learned
Motivation
LANL CCS-7 Application Performance Team FY18 Goal

- Explore programming models
- Extensively test on proxy apps and other relevant code
- Educate IC user community
## Background

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Portable CPU/GPU</th>
<th>Data layout</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAJA</td>
<td>Yes</td>
<td>Yes</td>
<td>Library</td>
</tr>
<tr>
<td>C++ AMP</td>
<td>Yes</td>
<td>No</td>
<td>Language</td>
</tr>
<tr>
<td>Thrust</td>
<td>Yes</td>
<td>No</td>
<td>Library</td>
</tr>
<tr>
<td>SGPU2</td>
<td>Yes</td>
<td>No</td>
<td>Library</td>
</tr>
<tr>
<td>XKAPPI</td>
<td>Yes</td>
<td>No</td>
<td>Library</td>
</tr>
<tr>
<td>OpenACC</td>
<td>Yes</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>OpenHMPP</td>
<td>Yes</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>StarSs</td>
<td>Yes</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>OmpSs</td>
<td>Yes</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>HOMPI</td>
<td>Yes</td>
<td>No</td>
<td>Translator</td>
</tr>
<tr>
<td>PEPPHER</td>
<td>Yes</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Yes</td>
<td>No</td>
<td>Language</td>
</tr>
<tr>
<td>StarPU</td>
<td>Yes</td>
<td>No</td>
<td>Language</td>
</tr>
<tr>
<td>Loci</td>
<td>No</td>
<td>Yes</td>
<td>Library</td>
</tr>
<tr>
<td>Cilk Plus</td>
<td>No</td>
<td>Yes</td>
<td>Language</td>
</tr>
<tr>
<td>TBB</td>
<td>No</td>
<td>No</td>
<td>Library</td>
</tr>
<tr>
<td>Charm++</td>
<td>No</td>
<td>No</td>
<td>Library</td>
</tr>
<tr>
<td>OpenMP</td>
<td>Almost*</td>
<td>No</td>
<td>Directives</td>
</tr>
<tr>
<td>CUDA†</td>
<td>No</td>
<td>No</td>
<td>Language</td>
</tr>
</tbody>
</table>
CoMD Proxy App
Based on SPaSM, a Gordon Bell Prize winning MD code important to several projects in materials science

- Essential feature is the emphasis on short-range interactions
- SPaSM, the go-to code for shaking out new architectures such as Roadrunner, Cielo, and Trinity, among other notable HPC platforms

Array-of-Structures (AoS) data layout

\[
\mathbf{r} = \{x_i, y_i, z_i\}, \quad i=1..\# \text{ atoms}
\]

\[
\mathbf{f} = \{f_{xi}, f_{yi}, f_{zi}, e_i\}, \quad i=1..\# \text{ atoms}
\]

Force calculations follow serialized (per rank), nested loop design

for all local link cells
  for each atom in local link cell
    for all neighbor link cells
      for each atom in neighbor link cell
CoMD: Molecular Dynamics Proxy Application

• Created and maintained by the Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx) group
  
  http://www.exmatex.org/comd.html
  
  https://github.com/exmatex/CoMD

• Written entirely in C using a collection of object-oriented
• Makes adding C++ code very simple (Kokkos is C++)
• CoMD exists as several ports to other programming models
  – pure MPI
  – OpenCL
  – CUDA
  – OpenMP
  – tuned vector + OpenMP (Toks)
Kokkos Programming Model
Kokkos: Many-core device performance library

- C++ library, not a new language or language extension
- Supports clear, concise, thread-scalable parallel patterns
- Abstracts away commonalities between languages and libraries that run on multiple architectures (CUDA, OpenACC, OpenCL, OpenMP, Coarray Fortran/Fortran 2008, …)
- Supports "write once, run everywhere" code development
- Minimizes the amount of architecture-specific implementation details users must know to leverage mix of simple & complex cores
- Solves the data layout problem through multi-dimensional arrays with architecture-dependent layouts to handle heterogeneous memory
Fundamental Abstractions

- Devices have *Execution Spaces* and *Memory Spaces*
  - **Execution spaces**: Subset of CPU cores, CPU, PIM, ...
  - **Memory spaces**: host memory, host pinned memory, high-bandwidth memory, GPU global memory, GPU shared memory, GPU UVM memory, ...

  *Dispatch computation to execution space accessing data in memory spaces*

- Multidimensional Arrays on steroids
  - Map multi-index (i,j,k,...) ↔ memory location in a *memory space*
  - Map derived from an array *layout*
  - Choose layout for device-specific memory access pattern
  - Make layout changes transparent to user code
  - Users only need to follow the simple API: a(i,j,k) (*Fortran-like*)

  *Separates user index space from memory layout*
Kokkos: Many-core device performance library

Kokkos::parallel_for(N, KOKKOS_LAMBDA (const int& i) {
    y[i] = a*x[i] + y[i];
});

- Kokkos defines a library of known pattern types.
- Computational bodies are passed to the libraries as C++ lambda functors.
- KOKKOS_LAMDA is a macro to hide much of the details from users.
- The above line is a short-hand way to tell Kokkos we want to do a parallel-for on the computational body we have given over the declared index i.
CoMD-Kokkos Porting Effort
CoMD-Kokkos

- Tuned version of CoMD contained ~6,000 lines of code
- Majority of time is spent calculating inter-atomic forces and time-stepping
  - Kokkos routines added to *ljForce* and *EAM* routines
  - Kokkos used to update velocities, positions, energy, and to redistribute atoms at the end of a time-step
  - Kokkos also used for initialization of data structures
- In all, 15 loops instrumented with Kokkos
  - 10 parallel-for
  - 5 parallel-reduce
- New data structures created to handle multiple-variable reductions
- Minor changes to some data structures to compile under C++11
L-J Force Calculation

```c
// loop over local boxes
for (int iBox=0; iBox<boxes->nLocalBoxes; iBox++)
{
    int nIBox = s->boxes->nAtoms[iBox];
    // loop over neighbors of iBox
    ...
    for (int jTmp=0; jTmp<nBrBoxes; jTmp++)
    {
        int jBox = s->boxes->nbrBoxes[iBox][jTmp];
        ...
        // loop over atoms in iBox
        for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOMS+nIBox); iOff++)
        {
            // loop over atoms in jBox
            for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
            {
                compute_sum_of_inter_atomic_potential_energy(...);
            } // loop over atoms in jBox
        } // loop over atoms in iBox
    } // loop over neighbor boxes
} // loop over local boxes in system
```
L-J Force Calculation

// loop over local boxes
Kokkos::parallel_reduce( s->boxes->nLocalBoxes, KOKKOS_LAMBDA( const int& iBox, 
          real_t& local_ePot)
{
    int nIBox = s->boxes->nAtoms[iBox];
    // loop over neighbors of iBox
    ...
    for (int jTmp=0; jTmp<nNbrBoxes; jTmp++)
    {
        int jBox = s->boxes->nbrBoxes[iBox][jTmp];
        ...
        // loop over atoms in iBox
        for (int iOff=MAXATOMS*iBox; iOff<(iBox*MAXATOMS+nIBox); iOff++)
        {
            // loop over atoms in jBox
            for (int jOff=jBox*MAXATOMS; jOff<(jBox*MAXATOMS+nJBox); jOff++)
            {
                // compute_sum_of_inter_atomic_potential_energy(...);
                } // loop over atoms in jBox
        } // loop over atoms in iBox
    } // loop over neighbor boxes
}, ePot); // loop over local boxes in system
Implementing multiple variable parallel reduction

```c
real_t v0 = 0.0;
real_t v1 = 0.0;
real_t v2 = 0.0;
real_t v3 = 0.0;

#pragma omp parallel for reduction(+:v0) reduction(+:v1) reduction(+:v2) reduction(+:v3)
for (int iBox=0; iBox<s->boxes->nLocalBoxes; ++iBox) {
    for (int iOff=MAXATOMS*iBox, ii=0; ii<s->boxes->nAtoms[iBox]; ++ii, ++iOff) {
        v0 += s->atoms->p[iOff][0];
        v1 += s->atoms->p[iOff][1];
        v2 += s->atoms->p[iOff][2];
        int iSpecies = s->atoms->iSpecies[iOff];
        v3 += s->species[iSpecies].mass;
    }
}

vcmLocal[0] = v0;
vcmLocal[1] = v1;
vcmLocal[2] = v2;
vcmLocal[3] = v3;
```

Summing momenta and particle mass
// A struct needed to implement a multiple variable reduction in Kokkos

struct vstruct {
    real_t v0;
    real_t v1;
    real_t v2;
    real_t v3;
    // default constructor
    KOKKOS_INLINE_FUNCTION
    vstruct() {
        v0 = 0.0;
        v1 = 0.0;
        v2 = 0.0;
        v3 = 0.0;
    }
    // operator += (volatile)
    KOKKOS_INLINE_FUNCTION
    vstruct & operator+=( const volatile vstruct & src) volatile {
        v0 += src.v0;
        v1 += src.v1;
        v2 += src.v2;
        v3 += src.v3;
    }
};
Implementing multiple variable parallel reduction

Summing momenta and particle mass

```cpp
vstruct Vvals;

Kokkos::parallel_reduce(s->boxes->nLocalBoxes, KOKKOS_LAMBDA( const int& iBox,
    vstruct& local_v) {
    for (int iOff=MAXATOMS*iBox, ii=0; ii<s->boxes->nAtoms[iBox]; ++ii, ++iOff) {
        local_v.v0 += s->atoms->p[iOff][0];
        local_v.v1 += s->atoms->p[iOff][1];
        local_v.v2 += s->atoms->p[iOff][2];
        local_v.int iSpecies = s->atoms->iSpecies[iOff];
        local_v.v3 += s->species[iSpecies].mass;
    }
}, Vvals);

vcmLocal[0] = Vvals.v0;
vcmLocal[1] = Vvals.v1;
vcmLocal[2] = Vvals.v2;
vcmLocal[3] = Vvals.v3;
```
Performance Results
The benchmark problem

• Single-node (1 MPI rank)
• 256K & 2,048K atoms
• 100 time steps
• Varied number of threads
• KNL and Haswell nodes of Trinity
• *Still working on GPU compilation*...
• Metrics
  - Walltime
  - Strong Scaling, speedup
  - Time ($\mu$s) per atom over solution
Performance Results

Speedup - ljForce, Haswell, 256K atoms

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

Speedup - ljForce, Haswell, 2,048K atoms

Threads

Speedup

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

Speedup - ljForce, KNL, 2,048K atoms

- Tuned, Kokkos, KNL_quad_flat
- Tuned, Kokkos, KNL_quad_cache
- Tuned, OpenMP, KNL_quad_flat
- Tuned, OpenMP, KNL_quad_cache
Performance Results

Speedup - EAM, Haswell, 2,048K atoms
Performance Results

Speedup - EAM, KNL, 2,048K atoms

- Tuned, Kokkos, KNL_quad_flat
- Tuned, Kokkos, KNL_quad_cache
- Tuned, OpenMP, KNL_quad_flat
- Tuned, OpenMP, KNL_quad_cache
Performance Results

Walltime - ljForce, Haswell, 2,048K atoms, 100 iterations

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

Walltime - ljForce, KNL_quad_flat, 2,048K atoms, 100 iterations

- Tuned, Kokkos
- Tuned, OpenMP

Threads

0 200 400 600 800 1000 1200 1400 1600 1800 2000

1 2 4 8 16 32 64 128 256
Performance Results

Walltime - EAM, Haswell, 2,048K atoms, 100 iterations

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

\[ \mu s/\text{atom} - \text{ljForce, Haswell, 2,048K atoms, 100 iterations} \]

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

\[ \mu s/\text{atom} - \text{ljForce} \]

KNL_quad_flat, 2,048K atoms, 100 iterations

<table>
<thead>
<tr>
<th>Threads</th>
<th>Tuned, Kokkos</th>
<th>Tuned, OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>8.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>32</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>64</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>128</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>256</td>
<td>0.0625</td>
<td>0.0625</td>
</tr>
</tbody>
</table>
Performance Results

\[ \mu s/\text{atom} - \text{ljForce} \]

KNL_quad_flat, 2,048K atoms, 100 iterations

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

μs/atom - EAM, KNL_quad_flat, 2,048K atoms, 100 iterations

- Tuned, Kokkos
- Tuned, OpenMP
Performance Results

Walltime (s) - ljForce
Haswell vs. KNL_quad_flat, 2,048K atoms, 100 iterations

Threads

Tuned, Kokkos, HSW
Tuned, OpenMP, HSW
Tuned, Kokkos, KNL
Tuned, OpenMP, KNL
Performance Results

Walltime (s) - EAM

Haswell vs. KNL_quad_flat, 2,048K atoms, 100 iterations

- Tuned, Kokkos, HSW
- Tuned, OpenMP, HSW
- Tuned, Kokkos, KNL
- Tuned, OpenMP, KNL
Kokkos impairs some vector performance

```c
// Compute Embedding Energy
// loop over all local boxes
Kokkos::parallel_reduce( s->boxes->nLocalBoxes, KOKKOS_LAMBDA( const int& iBox, real_t& local_etot) {
  int nIBox = s->boxes->nAtoms[iBox];
  // loop over atoms in iBox
  for (int iOff=MAXATOMS*iBox; iOff<(MAXATOMS*iBox+nIBox); iOff++) {
    real_t fEmbed, dfEmbed;
    interpolate(table_F, pot->rhobar[iOff], &fEmbed, &dfEmbed);
    pot->dfEmbed[iOff] = dfEmbed;
    s->atoms->U[iOff] += fEmbed;
    local_etot += fEmbed;
  }
}, etot);
```

--- begin vector cost summary ---
scalar cost: 291
vector cost: 38.750
estimated potential speedup: 7.000
--- end vector cost summary ---

--- begin vector cost summary ---
scalar cost: 21
vector cost: 2.250
estimated potential speedup: 8.630
--- end vector cost summary ---

unaligned access patterns
The Take-Away
Lessons Learned

1. **Difficult finding documentation for building Kokkos libs**
   - especially for GPUs
   - Options like `-DKOKKOS_OPT_RANGE_AGGRESSIVE_VECTORIZATION` hard to find

2. **CoMD relatively easy due to "C++"-like structure of the code**
   - But data structures need much work

3. **Special structs required for handling parallel reductions of more than one variable, i.e.**
   #pragma omp parallel for reduction(+:v0) reduction(+:v1)
   reduction(+:v2) reduction(+:v3)

4. **Vectorization performance suffers, unclear why**

5. **Code blocks inside Kokkos lambda functors are not optimized by the compiler to the level of the tuned, OpenMP version**

*We have a long way to go to show the benefit of Kokkos but we think it is there.*