Performance Portability of the Aeras Atmosphere Model and FELIX Land-Ice Model to Next Generation Architectures using Kokkos

Jerry Watkins & Irina Tezaur
Sandia National Laboratories
2017 Albany Users Meeting, January 18th, 2017
Motivation

- Climate models need higher resolutions and more computational power
- High performance computing architectures are becoming increasingly more heterogeneous
Earth System Model (ESM)

- ESM has six modular components:
  1. Atmosphere model
  2. Ocean model
  3. Sea ice model
  4. Land ice model
  5. Land model
  6. Flux coupler

- DOE funded collaboration between several national laboratories
Earth System Model (ESM)

- ESM has six modular components:
  1. Atmosphere model
  2. Ocean model
  3. Sea ice model
  4. Land ice model
  5. Land model
  6. Flux coupler

- DOE funded collaboration between several national laboratories

- This presentation will focus on atmosphere
Aeras

- Next Generation Global Atmosphere Model
  - Uncertainty Quantification and Performance Portability

- Utilizes state-of-the-art C++ libraries
  - Trilinos (discretizations, meshing, coupling, performance portability)
  - Albany (multiphysics application code base)

- Performance portability through Kokkos
  - Parallel performance across a variety of different architectures
Global Atmosphere Model

- 3D Hydrostatic Equations
  - Conservation equations similar to Euler/NS

- Spectral element, explicit time integration
  - Gauss-Lobato points
  - diagonal mass matrix

- Quadrilateral elements on a spherical shell domain

- Finite difference in hybrid vertical coordinate system

- Stabilization through hyperviscosity
Parallelism on modern hardware

- Memory access time has remained the same
- Single core performance has improved but stagnated
- More performance from multicore/manycore processors
- Compute is cheap, memory transfer is expensive!

<table>
<thead>
<tr>
<th>Year</th>
<th>Memory Access Time</th>
<th>Single Core Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980s</td>
<td>~100 ns</td>
<td>~100 ns</td>
</tr>
<tr>
<td>Today</td>
<td>~50-100 ns</td>
<td>~1 ns</td>
</tr>
</tbody>
</table>
Performance Portability

- Problem: Computer architectures rapidly changing
  - Leads to heterogeneous clusters
  - Trends remain the same (i.e. increased computational power through manycore architectures)

- Kokkos programming model
  - C++ library which provides performance across multiple computing architectures
  - Examples: Multicore CPU, GPU, Intel Xeon Phi and more
  - Abstracts data layouts for optimal performance

- Allows researchers to focus on algorithm development for large heterogeneous clusters
  - Write an algorithm once for multiple architectures
Performance Porting using Kokkos

- Aeras finite element assembly is organized into evaluator classes
- Simple Example: Computing Pressure

```cpp
template<
    typename EvalT, 
    typename Traits>

void XZHydrostatic_Pressure<EvalT, Traits>::
    evaluateFields(typename Traits::EvalData workset)
{
    for (int cell=0; cell < workset.numCells; ++cell) {
        for (int node=0; node < numNodes; ++node) {
            for (int level=0; level < numLevels; ++level) {
                Pressure(cell, node, level) = A(level)*P0 + B(level)*Ps(cell, node);
            }
        }
    }
}
```
Performance Porting using Kokkos

- Aeras finite element assembly is organized into evaluator classes
- Simple Example: Computing Pressure

```cpp
template<typename EvalT, typename Traits>
void XZHydrostatic_Pressure<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset)
{
    for (int cell=0; cell < workset.numCells; ++cell) {
        for (int node=0; node < numNodes; ++node) {
            for (int level=0; level < numLevels; ++level) {
                Pressure(cell, node, level) = A(level)*P0 + B(level)*Ps(cell, node);
            }
        }
    }
}
```
Performance Porting using Kokkos

- Loops are parallelized using inline functions
  - Added to class definition:
    ```cpp
typedef Kokkos::View<int***, PHX::Device>::execution_space ExecutionSpace;

struct XZHystrostatic_Pressure_Tag{};
typedef Kokkos::RangePolicy<ExecutionSpace, XZHystrostatic_Pressure_Tag> XZHystrostatic_Pressure_Policy;

KOKKOS_INLINE_FUNCTION
void operator() (const XZHystrostatic_Pressure_Tag& tag, const int& i) const;
```
  - Modified member function:
    ```cpp
template<typename EvalT, typename Traits>
KOKKOSINLINEFUNCTION
void XZHystrostatic_Pressure<EvalT, Traits>::operator() (const XZHystrostatic_Pressure_Tag& tag, const int& cell) const{
  for (int node=0; node < numNodes; ++node) {
    for (int level=0; level < numLevels; ++level) {
      Pressure(cell,node,level) = A(level)*P0 + B(level)*Ps(cell,node);
    }
  }
}
```

```cpp
template<typename EvalT, typename Traits>
void XZHystrostatic_Pressure<EvalT, Traits>::evaluateFields(typename Traits::EvalData workset)
{
  Kokkos::parallel_for(XZHystrostatic_Pressure_Policy(0,workset.numCells),*this);
}
Performance Porting using Kokkos

- Loops are parallelized using inline functions
  - Added to class definition:
    ```cpp
typedef Kokkos::View<int***, PHX::Device>::execution_space ExecutionSpace;

struct XZHydrostatic_Pressure_Tag{};
typedef Kokkos::RangePolicy<ExecutionSpace, XZHydrostatic_Pressure_Tag> XZHydrostatic_Pressure_Policy;

KOKKOS_INLINE_FUNCTION
void operator() (const XZHydrostatic_Pressure_Tag& tag, const int& i) const;
```

- Modified member function:
  ```cpp
  template<typename EvalT, typename Traits>
  KOKKOS_INLINE_FUNCTION
  void XZHydrostatic_Pressure<EvalT, Traits>:::
  operator() (const XZHydrostatic_Pressure_Tag& tag, const int& cell) const{
    for (int node=0; node < numNodes; ++node) {
      for (int level=0; level < numLevels; ++level) {
        Pressure(cell,node,level) = A(level)*P0 + B(level)*Ps(cell,node);
      }
    }
  }

  template<typename EvalT, typename Traits>
  void XZHydrostatic_Pressure<EvalT, Traits>:::
  evaluateFields(typename Traits::EvalData workset) {
    Kokkos::parallel_for(XZHydrostatic_Pressure_Policy(0, workset.numCells), *this);
  }
  ```
Performance Porting using Kokkos

- Loops are parallelized using inline functions
  - Added to class definition:
    ```cpp
typedef Kokkos::View<int***, PHX::Device>::execution_space ExecutionSpace;

struct XZHydrostatic_Pressure_Tag{};
typedef Kokkos::RangePolicy<ExecutionSpace, XZHydrostatic_Pressure_Tag> XZHydrostatic_Pressure_Policy;

KOKKOS_INLINE_FUNCTION
void operator()(const XZHydrostatic_Pressure_Tag& tag, const int& i) const;
```
  - Modified member function:
    ```cpp
template<typename EvalT, typename Traits>
void XZHydrostatic_Pressure<EvalT, Traits>::
operator()(const XZHydrostatic_Pressure_Tag& tag, const int& cell) const{
    for (int node=0; node < numNodes; ++node) {
        for (int level=0; level < numLevels; ++level) {
            Pressure(cell,node,level) = A(level)*P0 + B(level)*Ps(cell,node);
        }
    }
}
```
```cpp
template<typename EvalT, typename Traits>
void XZHydrostatic_Pressure<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset)
{
    Kokkos::parallel_for(XZHydrostatic_Pressure_Policy(0,workset.numCells),*this);
}
Performance Porting using Kokkos

- Loops are parallelized using inline functions
  - Added to class definition:
    ```cpp
typedef Kokkos::View<int**, PHX::Device>::execution_space ExecutionSpace;

struct XZHydrostatic_Pressure_Tag{};
typedef Kokkos::RangePolicy<ExecutionSpace, XZHydrostatic_Pressure_Tag> XZHydrostatic_Pressure_Policy;

KOKKOS_INLINE_FUNCTION
void operator() (const XZHydrostatic_Pressure_Tag& tag, const int& i) const;
```
  - Modified member function:
    ```cpp
template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void XZHydrostatic_Pressure<EvalT, Traits>::
operator() (const XZHydrostatic_Pressure_Tag& tag, const int& cell) const{
  for (int node=0; node < numNodes; ++node) {
    for (int level=0; level < numLevels; ++level) {
      Pressure(cell,node,level) = A(level)*P0 + B(level)*Ps(cell,node);
    }
  }
}

template<typename EvalT, typename Traits>
void XZHydrostatic_Pressure<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset)
{
  Kokkos::parallel_for(XZHydrostatic_Pressure_Policy(0,workset.numCells),*this);
}
```
Performance Porting using Kokkos

- Loops are parallelized using inline functions
  - Added to class definition:

```cpp
typedef Kokkos::View<int**, PHX::Device>::execution_space ExecutionSpace;
struct XZHydrostatic_Pressure_Tag{};
typedef Kokkos::RangePolicy<ExecutionSpace, XZHydrostatic_Pressure_Tag> XZHydrostatic_Pressure_Policy;
KOKKOS_INLINE_FUNCTION
void operator() (const XZHydrostatic_Pressure_Tag& tag, const int& i) const;
```

- Modified member function:

```cpp
template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void XZHydrostatic_Pressure<EvalT, Traits>::operator() (const XZHydrostatic_Pressure_Taq& taq, const int& cell) const{
    for (int node=0; node < numNodes; ++node) {
        for (int level=0; level < numLevels; ++level) {
            Pressure(cell,node,level) = A(level)*P0 + B(level)*Ps(cell,node);
        }
    }
}
template<typename EvalT, typename Traits>
void XZHydrostatic_Pressure<EvalT, Traits>::evaluateFields(typename Traits::EvalData workset)
{
    Kokkos::parallel_for(XZHydrostatic_Pressure_Policy(0,workset.numCells),*this);
}
Performance Porting using Kokkos

- The same code can be used to run on many “devices”
  - (e.g. OpenMP and CUDA)

- For CUDA/GPUs, “A” and “B” must be Kokkos DynRankViews and declared within the class.

- Data transfer from host to device handled by CUDA UVM

- “Kokkos::parallel_for” is mostly used in the code, “Kokkos::atomic_fetch_add” used for filling the sparse matrix
Problem Specification

- **Baroclinic Instability Test Case:**

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Resolution</th>
<th># Elements</th>
<th>Fixed dt</th>
<th>Hyperviscosity Tau</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform_30</td>
<td>1°</td>
<td>5,400</td>
<td>30</td>
<td>5.0e15</td>
</tr>
<tr>
<td>uniform_60</td>
<td>0.5°</td>
<td>21,600</td>
<td>10</td>
<td>1.09e14</td>
</tr>
<tr>
<td>uniform_120</td>
<td>0.25°</td>
<td>86,400</td>
<td>5</td>
<td>1.18e13</td>
</tr>
</tbody>
</table>

- 100 explicit RK43 iterations, 3\(^{rd}\) order elements, 10 vert levels
Computer Architectures

- Shannon used for testing, performance tests
  - 32 nodes w/ varying types/numbers of GPUs per node (10 nodes w/ one NVIDIA K80 dual-GPU)

- Titan used for full length simulations, performance tests
  - 18,688 nodes w/ one NVIDIA K20X GPU per node
uniform_30 test case is faster on Shannon
Poor scaling
OpenMP Strong Scalability

- Efficiency = 100 × \(\frac{\text{Speedup}}{\text{#Cores}}\)
- Poor scaling
MPI+OpenMP Strong Scalability

- uniform_30 test case
- 16 MPI ranks vs. 2 MPI ranks + 8 OpenMP threads per node
- Pure MPI scales better on Shannon
MPI+OpenMP Strong Scalability

- uniform_30 test case
- 16 MPI ranks vs. 2 MPI ranks + 8 OpenMP threads per node
- Pure MPI scales better on Titan
Workset Size on Shannon

- uniform_30 test case
- GPU memory limit reached when using less than 8 GPUs
- ~4x OpenMP Speedup, ~1x GPU Speedup (675 workset size)
Weak Scalability on Titan

- ~3x OpenMP Speedup
- ~0.5x GPU Speedup
- Workset size too small (168 elements per GPU)
Methods for Improvement

- Reduce excess memory usage
- Parallelize over other indices (e.g. vertical levels)
- Utilize shared memory for interpolation
- Replace CUDA UVM with manual memory transfer
- Profile using ‘tau’ and ‘nvprof’
Conclusions

- A performance portable implementation of the 3D hydrostatic equations was implemented using Kokkos.

- Heterogeneous high performance computing architectures can now be utilized for atmospheric research in Aeras.

- Performance studies show that further optimization is needed to fully utilize all resources.

Future Work (Albany/FELIX)

- Generate performance profiles (Xeon Phi, P100 GPU)
- Identify performance bottle necks on GPU (nvprof/NVTX)
- Finish porting and evaluate next generation solvers for FELIX
- Improve performance of the finite element assembly