Kokkos implementation of Albany: a performance-portable finite element application

Sandia National Laboratories
Irina Demeshko, H. Carter Edwards, Michael A. Heroux, Eric T. Phipps, Andrew G. Salinger , Roger P. Pawowski
Sandia National Laboratories, New Mexico, 87185

Introduction

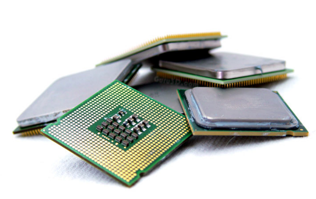
Problem:
Modern HPC applications, Finite Element Assembly codes in particular, need to be run on many different platforms and performance portability has become a critical issue: parallel code needs to be executed correctly and performant despite variation in the architecture, operating system and software libraries.

Approach:
Kokkos programming model from Trilinos: C++ library, which provide performance portability across diverse devices with different memory models.

Funding: ASC Exascale Codesign Project

Kokkos programming model

Kokkos [1] - C++ library from Trilinos [2] to provide scientific and engineering codes with an intuitive manycore performance portable **programming model**.


- Provides portability across manycore devices (Multicore CPU, NVidia GPU, Intel Xeon Phi (potential: AMD Fusion))
- Abstract data layout for non-trivial data structures
- Uses library approach:
  - *Maximize amount of user (application/library) code that can be compiled without modification and run on these architectures*
  - *Minimize amount of architecture-specific knowledge that a user is required to have*
- Performant: Portable user code performs as well as architecture-specific code

Main abstractions:
- Kokkos executes computational kernels in fine-grain data parallel within an *Execution space.*
- Computational kernels operate on multidimensional arrays (*Kokkos::View*) residing in *Memory spaces.*
- Kokkos provides these multidimensional arrays with polymorphic data layout, similar to the Boost.MultiArray flexible storage ordering.

Ice Sheet Simulation Code (FELIX)

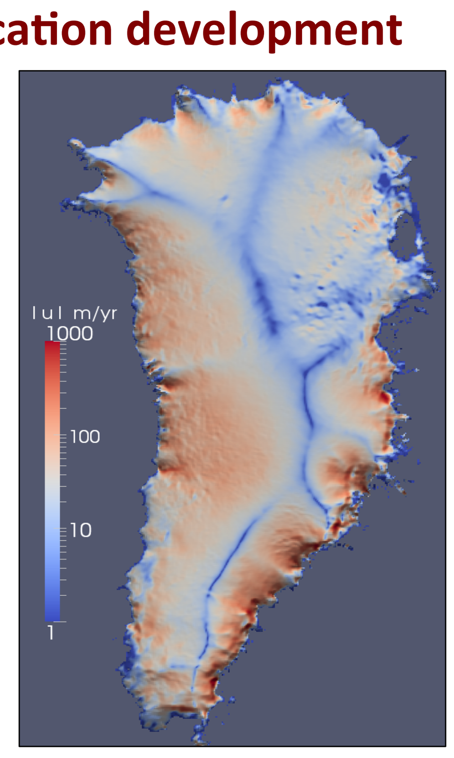
We are developing a performance portable implementation of an Ice Sheet simulation code*, build with the Albany [3] application development environment

Project objectives:
- Develop a robust, scalable, unstructured-grid finite element code for land Ice dynamics.
- Provide sea level rise prediction
- Run on new architecture machines (hybrid systems).

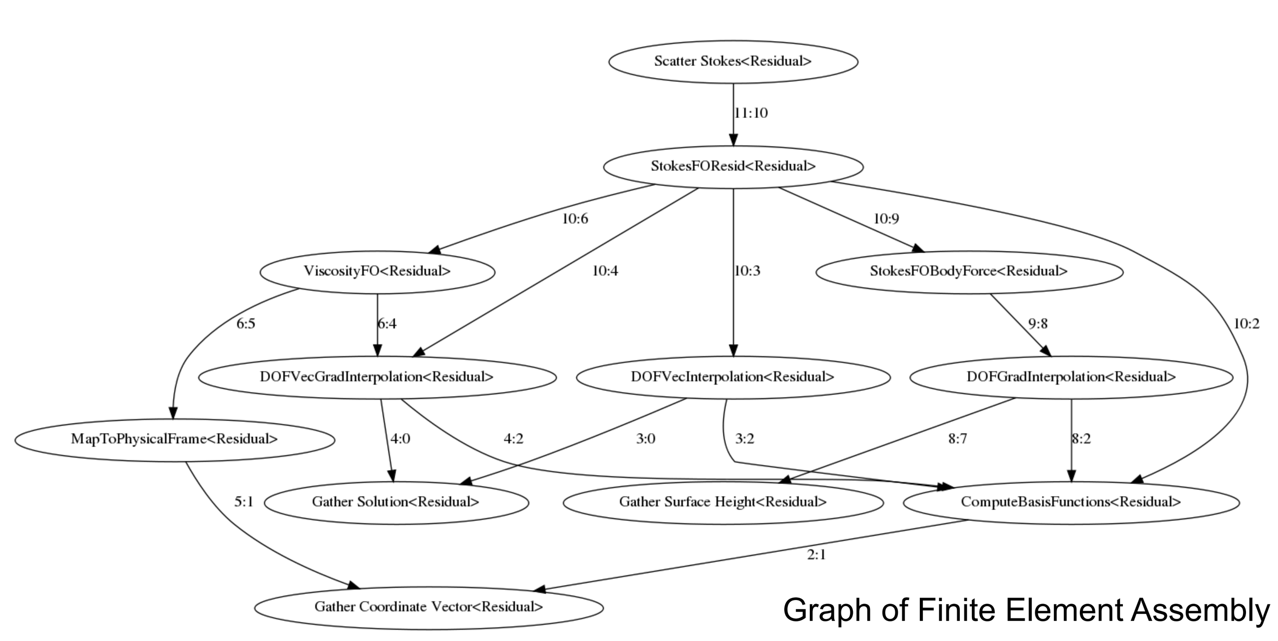
Nonlinear Stokes' Model for Ice Sheet Stresses

$$-\nabla \cdot (2\mu\dot{\epsilon}_1) = -\rho g \frac{\partial s}{\partial x}$$
$$-\nabla \cdot (2\mu\dot{\epsilon}_2) = -\rho g \frac{\partial s}{\partial y}$$

Fully Implicit Solution Algorithm:
- Finite Element Assembly
  - ~50% CPU time
- Linear Solve ILU + CG
  - ~50% CPU Time
- Problem Size, up to:
  - 1.12B Unknowns
  - 16384 Cores on Hopper



Computed Surface Velocity [m/yr] for Greenland Ice Sheet



Graph of Finite Element Assembly Kernels

**Funding Source: PISCEES* SciDAC Application (BER & ASCR)
Authors: A. Salinger, I. Kalashnikova, M. Perego, R. Tuminaro [SNL], S. Price[LANL]

Implementation algorithm:

1) Replace array allocations with Kokkos::Views (in Host space)

*Kokkos::View<ScalarType****, Device> jacobian("Jacobian", worksetNumCells, numQPs, numDims, numDims);*

2) Replace array access with Kokkos::Views

B[k][j] = jacobian(i0, i1, k, j);
Kokkos::deep_copy (jacobian, host_jacobian);

3) Replace functions with Functors, run in parallel on Host

Example: Compute Jacobian

Initial code:

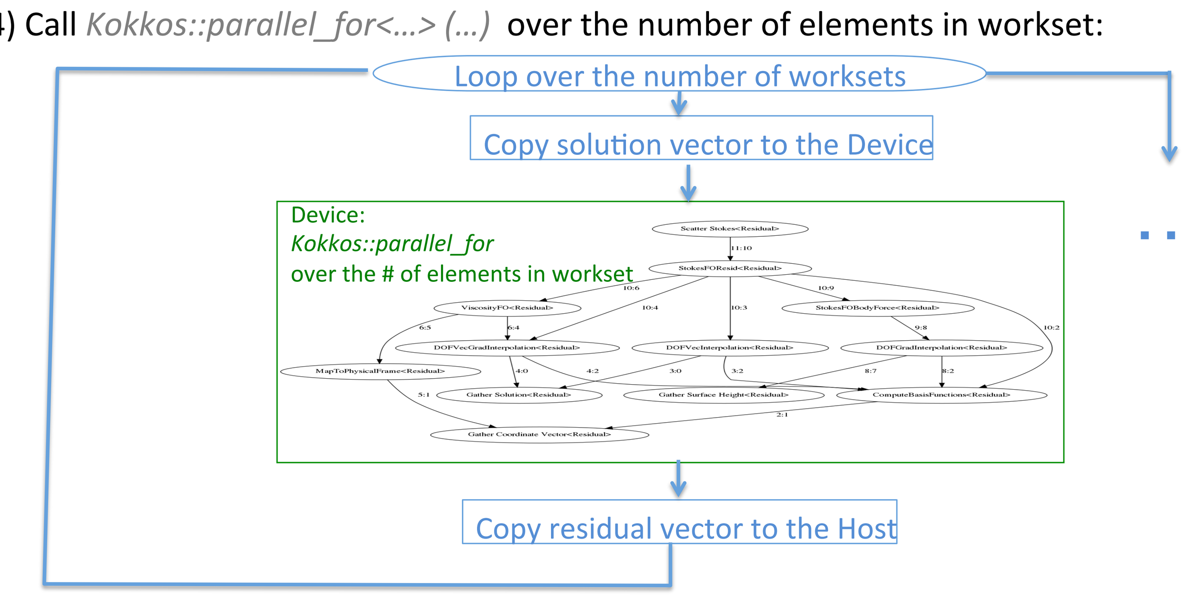
```
for(int cell = 0; cell < worksetNumCells; cell++) {
   for(int qp = 0; qp < numQPs; qp++) {
     for(int row = 0; row < numDims; row++){
       for(int col = 0; col < numDims; col++){
         for(int node = 0; node < numNodes; node++){
           jacobian(cell, qp, row, col) +=
                       coordVec(cell, node, row)
                      *basisGrads(node, qp, col);
         } // node
       } // col
     } // row
   } // qp
} // cell
```

Kokkos Implementation:

```
Kokkos::parallel_for ( worksetNumCells,
 compute_jacobian<ScalarT, Device, numQPs, numDims, numNodes>
(basisGrads, jacobian, coordVec));
```

```
template < typename ScalarType, clas DeviceType, int numQPs_, int numDims_, int numNodes_ >
class compute_jacobian {
  Array3 basisGrads_;
  Array4 jacobian_;
  Array3_const coordVec_;
 public:
  typedef DeviceType device_type;
  compute_jacobian(Array3 &basisGrads,  Array4 &jacobian, Array3 &coordVec)
                     : basisGrads_(basisGrads) , jacobian_(jacobian), coordVec_(coordVec){}
KOKKOS_INLINE_FUNCTION
 void operator () (const std::size_t cell) const
 {
 for(int qp = 0; qp < numQPs_; qp++) {
     for(int row = 0; row < numDims_; row++){
       for(int col = 0; col < numDims_; col++){
         for(int node = 0; node < numNodes_; node++){
           jacobian_(cell, qp, row, col) += coordVec_(cell, node, row)*basisGrads_(node, qp, col);
         } // node
       } // col
     } // row
   } // qp
 }
 };
```

4) Call *Kokkos::parallel_for<…> (…)* over the number of elements in workset:

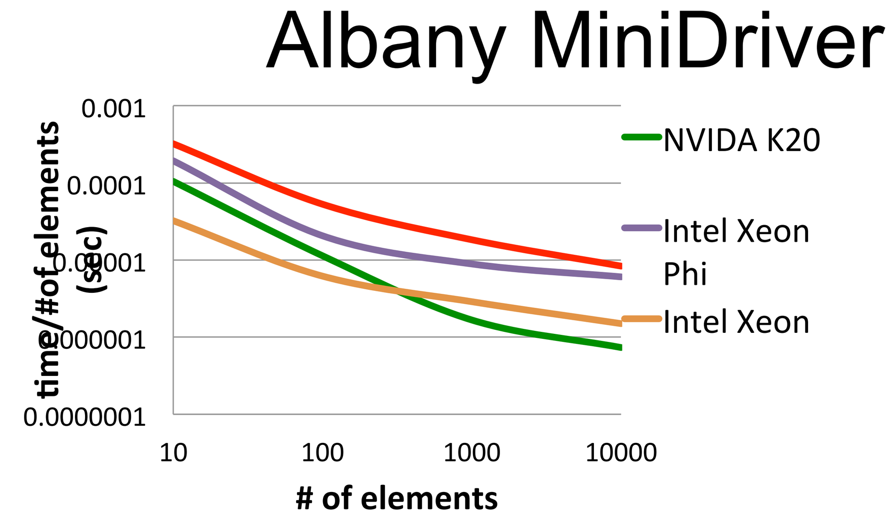


5) Set Device to 'Cuda', 'OpenMP' or 'Threads' and run on specified Device

typedef Kokkos::Cuda Device; or typedef Kokkos::OpenMP Device; or typedef Kokkos::Threads Device;

Performance Results*

Albany MiniDriver test code

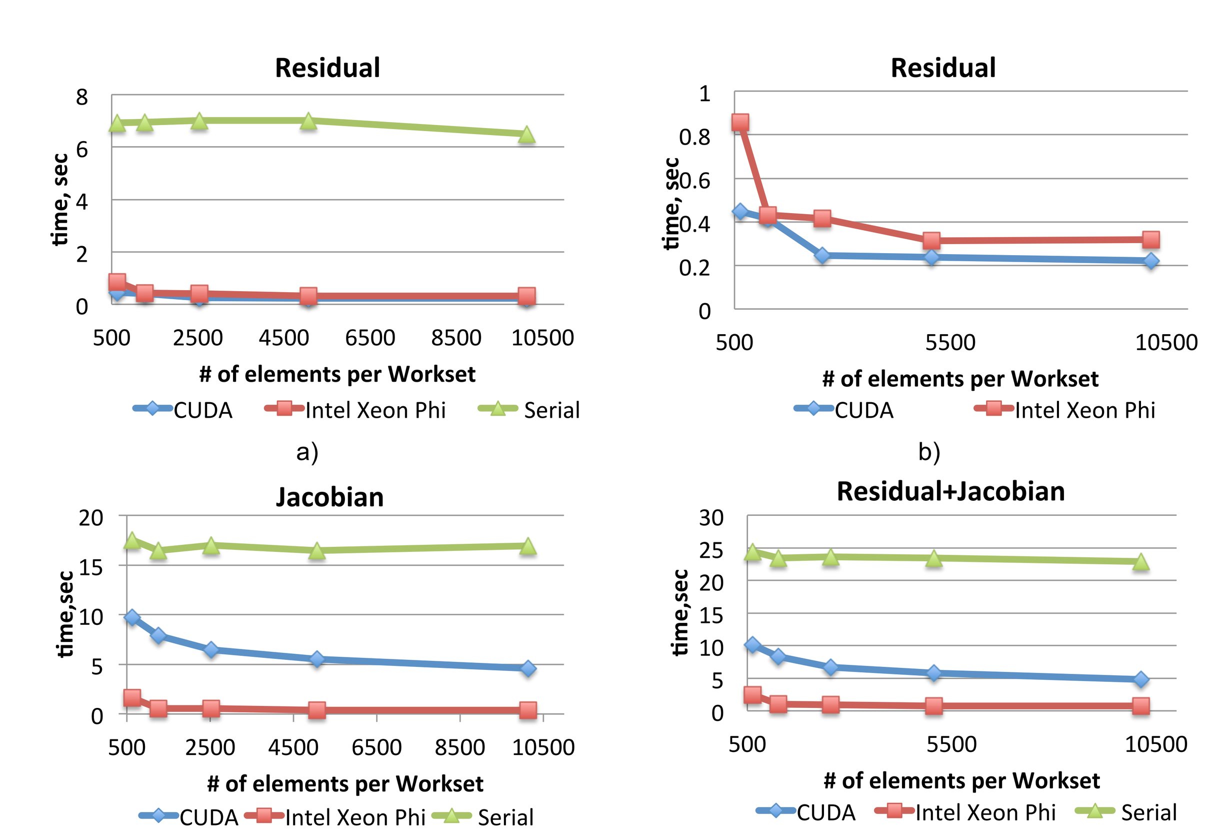


Evaluation Environment:
Compton (Intel MIC cluster) :
42 nodes:
- Two 8-core Sandy Bridge Xeon E5-2670 @ 2.6GHz (HT activated) per node,
- 24GB (3*8Gb) memory per node,
- Two Pre-production KNC (Intel MIC) 2 per node (57 cores per each)

Shannon (NVIDIA GPU cluster):
32 nodes:
- Two 8-core Sandy Bridge Xeon E5-2670 @ 2.6GHz (HT deactivated) per node,
- 128GB DDR3 memory per node,
- 2x NVIDIA K20x per node

- "Fused" means that all the Kokkos kernels are fused togeather, when in "Separated" version we have several independent kernels
- ** "Nvidia K20 GPU" and "Initial Implementation" results were obtained on Shannon, "Intel MIC" were obtained on Compton

Albany FELIX code



Performance evaluation results for the Kokkos implementation of Albany/FELIX code: weak scalability comparison for Serial, CUDA and Intel Xeon Phi implementations. a)-b) Performance results for Residual computations. c) Performance results for Jacobian computations. d) Total execution time for Fiinite Element Assembly in Felix code.

References:
1) Edwards, H. Carter, Trott, Christian R., Sunderland, Daniel *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns.* Journal of Parallel and Distributed Computing, 2014.
2) Salinger, Andrew G. *Component-based Scientific application Development.*, December 01, 2012
3) Willenbring, James M.,and Michael Allen Heroux. *Trilinos Users Guide.*, August 01, 2003.

c)
d)

SAND2014-18787C