SAND2015-5946C

Exceptional service in the national interest

Sandia National Laboratories

Progress on Manycore Multiphysics Simulation in Albany using the Kokkos Portable Hardware Abstraction Library

Andrew Bradley, Irina Demeshko, Brian Granzow[1], Glen Hansen, Alejandro Mota, Andy Salinger, Irina Tezaur

Sandia National Laboratories

[1]Rensselaer Polytechnic Institute

USNCCM13, July 2015

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

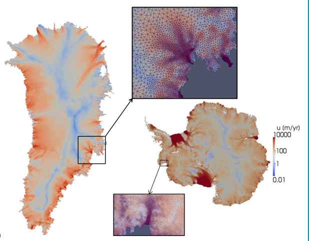
Overview

- Albany supports a wide variety of application physics areas including heat transfer, fluid dynamics, structural mechanics, plasticity, quantum device modeling, climate modeling, and many others
- Leverages Trilinos framework for linear and nonlinear solvers, preconditioning, load balancing, hardware portability layers, finite element data structures and multiphysics code support infrastructure
- Advanced analysis capabilities (embedded SA and UQ), supported by parallel scalable RPI adaptive meshing technologies, support for topology optimization

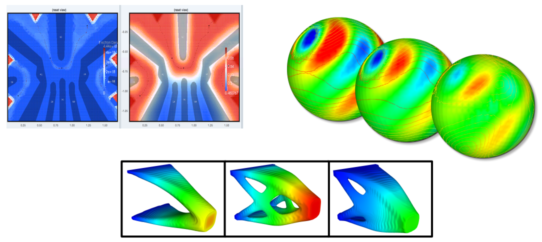
Albany ecosystem

Application Impact: Ice Sheets

- Surface flow velocities for Greenland and Antarctic Ice Sheets
- Demonstrates nonlinear solves, linear solves, UQ, adaptivity, and performance portability
- Employs automatic differentiation, discretizations, partitioning, mesh database

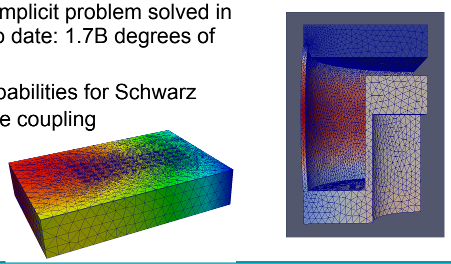


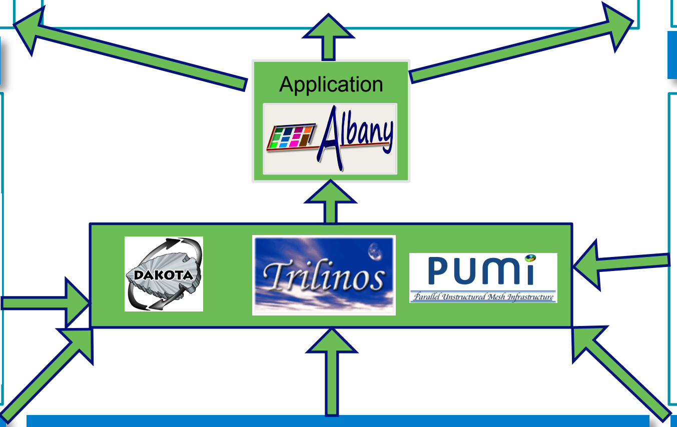
Additional Application Impact



Application Impact: Computational Mechanics

- Largest implicit problem solved in Albany to date: 1.7B degrees of freedom
- Initial capabilities for Schwarz multiscale coupling

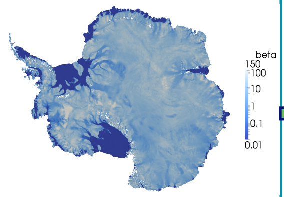




Nonlinear Solvers and Inversion

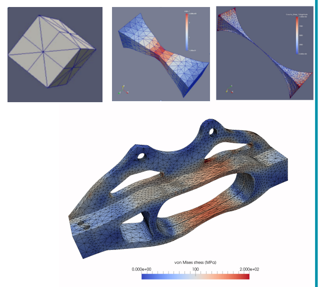
- Homotopy and Anderson Acceleration in Trilinos::NOX
- The robustness of nonlinear solvers are critical when an application is to be called as a sub-component within a larger application code.
- Uses Automatic Differentiation, Preconditioning, Optimization algorithms from Trilinos

$$\frac{dg}{dp} = \frac{\partial g}{\partial x}^T \frac{\partial f}{\partial x}^{-1} \frac{\partial f}{\partial p} + \frac{\partial g}{\partial p}$$



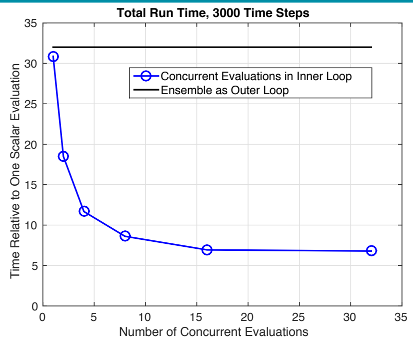
Mesh Adaptivity

- Mesh adaptation can be essential for efficiency and robustness
- Cube geometry subjected to large deformation (elasticity and J2 plasticity results shown)



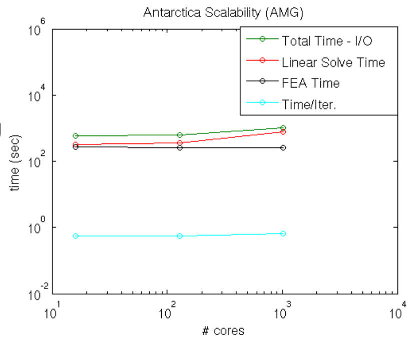
Embedded UQ

- New *Ensemble* data type in Sacado package
- Vectorization of kernels over ensembles
- Contiguous memory access in arrays



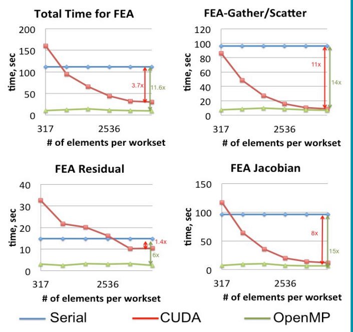
Scalable Linear Algebra

- Scalability of simulations requires effective preconditioning
- Multi-level solves are essential for the largest problems



Performance Portability

- The Kokkos programming mode supports performance portability of kernels.
- Kokkos' abstraction layer allows code to be tailored for specific devices



Extreme scale strategy

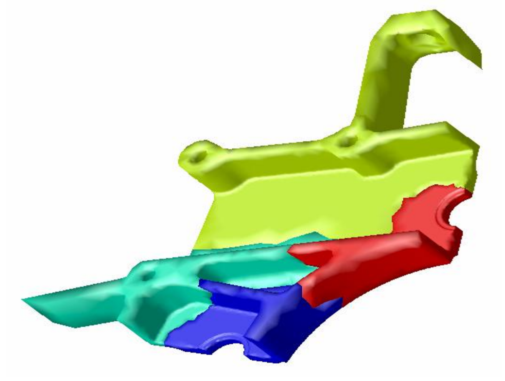
- Use of *embedded meshing* to address problem input challenges on advanced hierarchical platforms
- Implement performance portable kernels
  - The *target* hardware architecture and programming environment is not yet clear
  - Focus on *performance portability*; use the Kokkos hardware abstraction library to allow hosting of parallel on-rank algorithms on multiple parallel devices (multicore CPUs, Intel Phi, Nvidia GPU) via recompile
- Embedded Analysis, including UQ

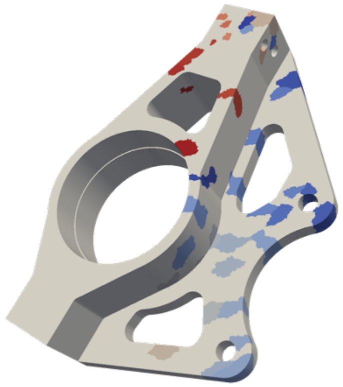
Embedded meshing workflow

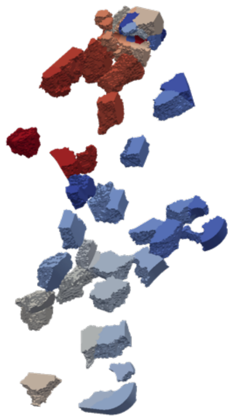
- Read the problem statement that describes the problem and domain, distribute to the coarse tasks (ranks)
- Each rank generates the mesh within its partition, such that the overall mesh is conformal across the parallel machine
- Rebalancing of the final partitions is performed as needed

Partitioned problem statement

- The domain geometry is tagged with initial and boundary condition data and partitioned in advance by the user, into *parts*
- Each part is meshed as an independent parallel task
- Initially, a high order coarse mesh is used that represents both the part volumes and the geometry of the surfaces in the domain
  - Adaptive refinement is used to achieve the target element size, using linear elements as the default case

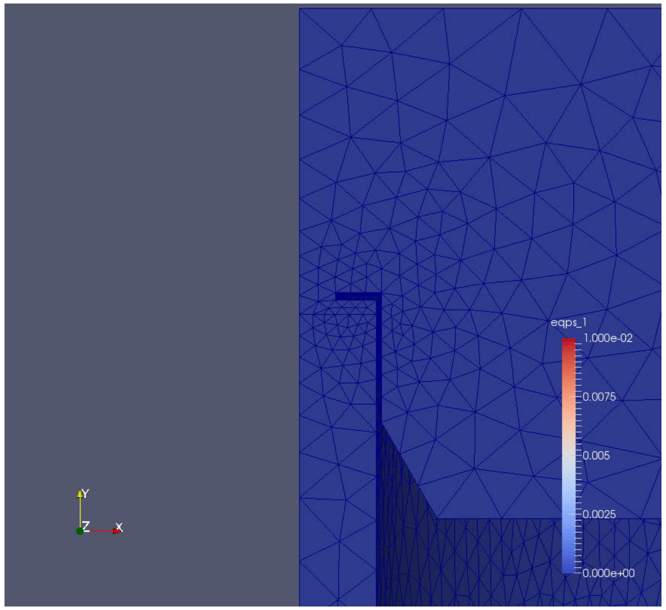


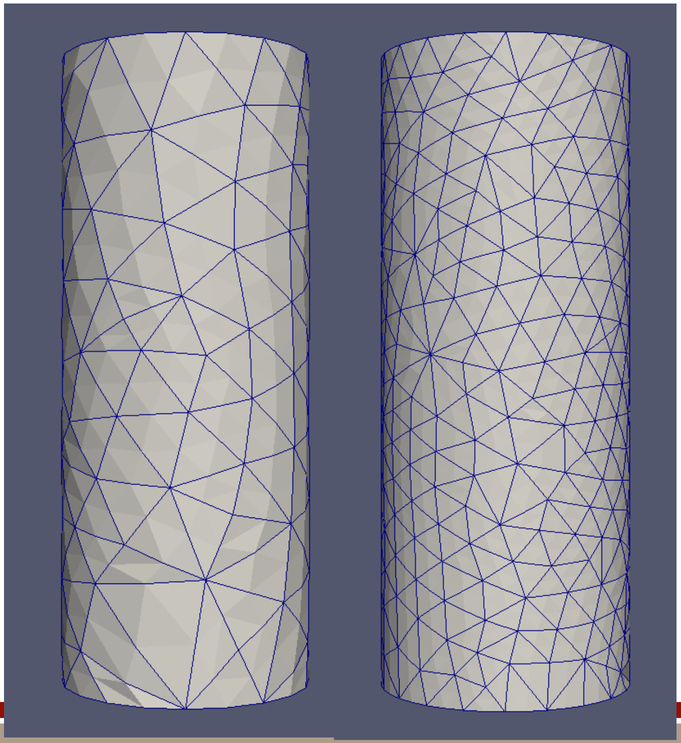




Embedded meshing demonstration

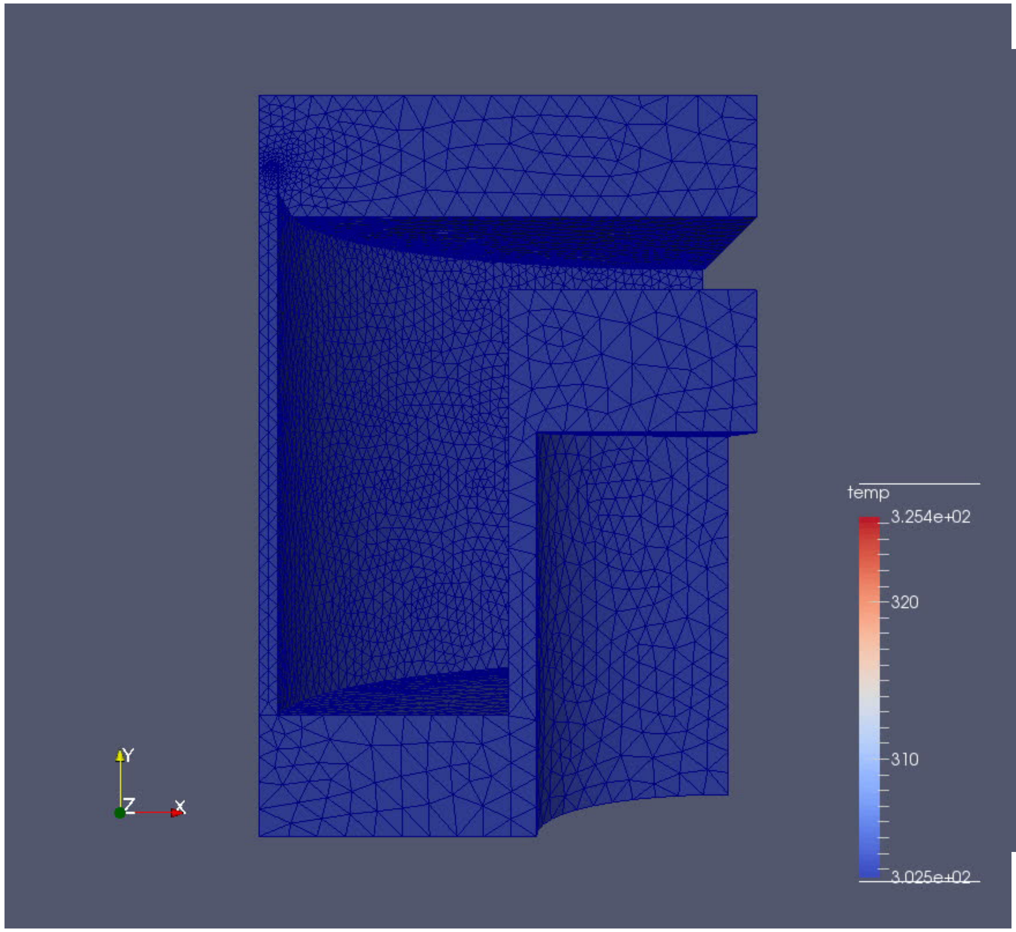
- Higher order coarse elements capture surface geometry
- Element sizes chosen to represent feature length scales in underlying model
- Mesh generated on coarse elements to meet simulation requirements

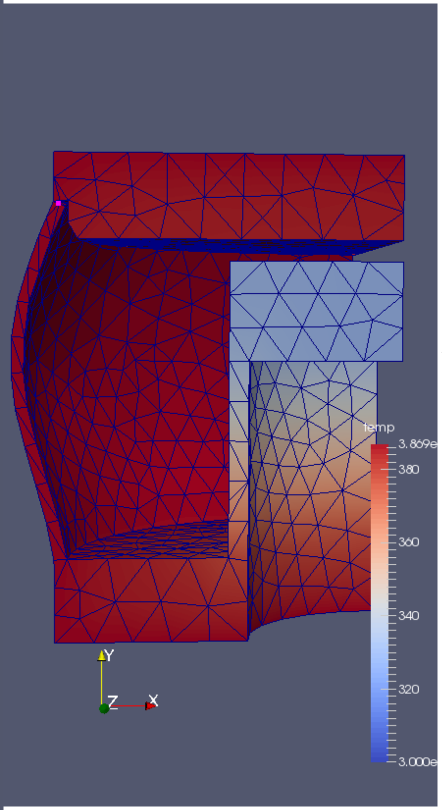


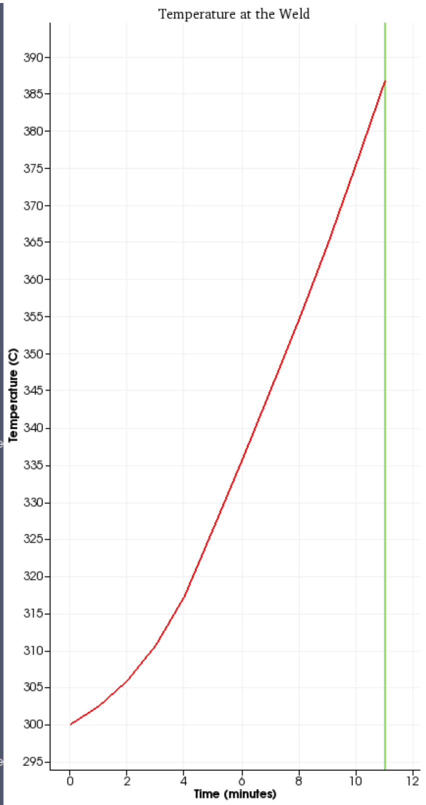


Full scale example

- Coupled thermomechanics, J2 plasticity, temperature dependent material hardening at the weld

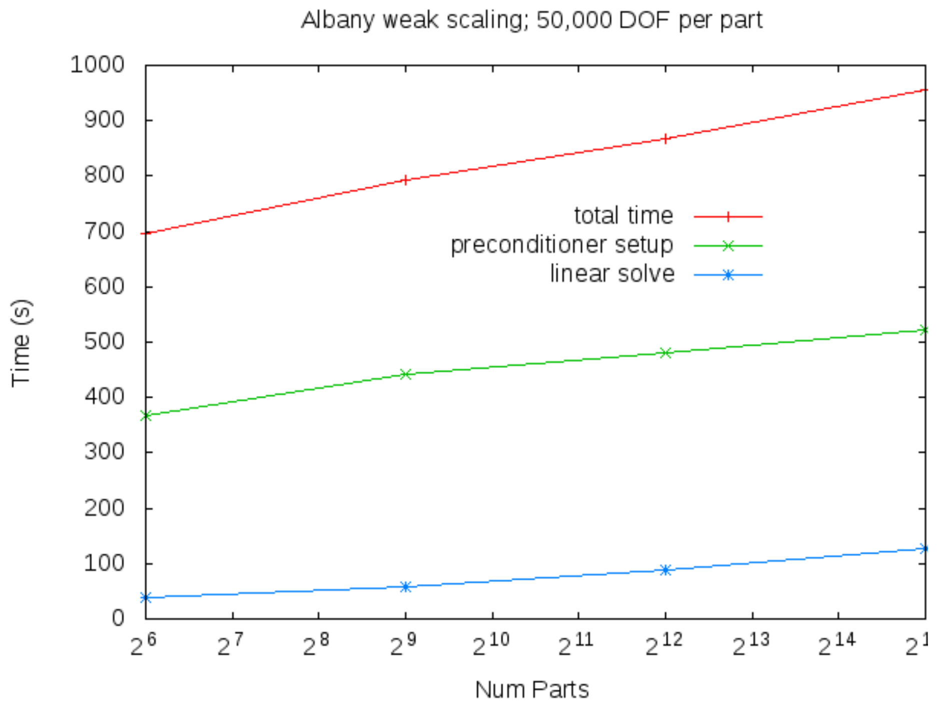


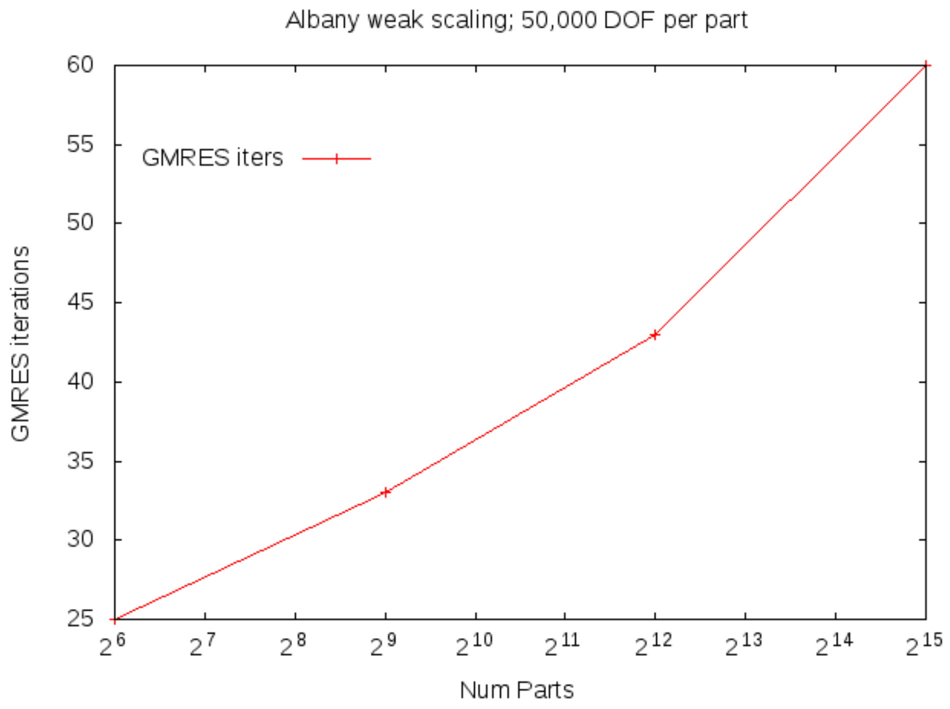




Mechanics scalability

- Setup, solution, and solver scalability as a function of partitioning granularity on BGQ



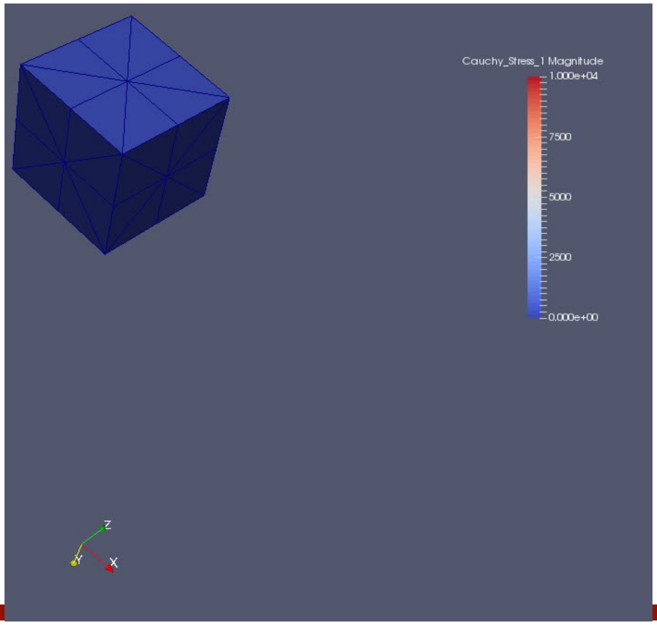


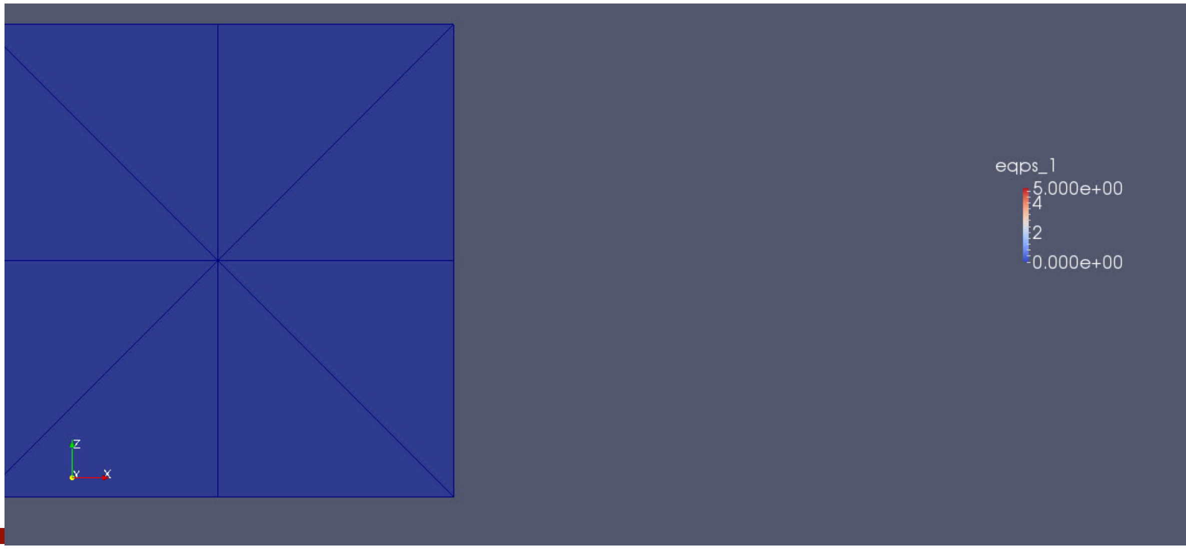
Mesh adaptation

Goals: Maintain element quality in large deformation, high strain rate scenarios

- Method: Employ evolving mesh topology to delay the transition to ALE or full-Eulerian regime
- Challenges: Ensuring conservation as the mesh adapts, maintaining element quality, robustness of the approach during complex deformation/mixing states

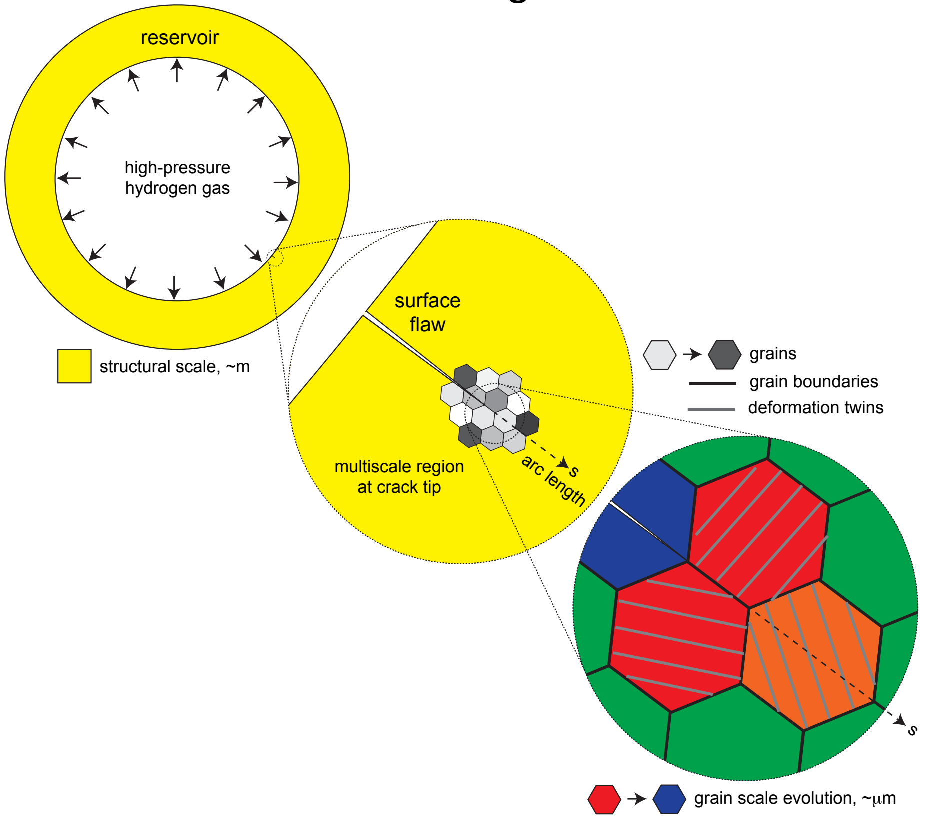
- Method:
  - Stress model is a function of the deformation gradient tensor F.
  - F is stored at integration points.
  - In a step, F is (a) updated and then (b) transferred to the adapted mesh.
  - Interpolation data transfer is used in the examples below. Need to develop reconnection-friendly conservative interpolation and recovery methods!





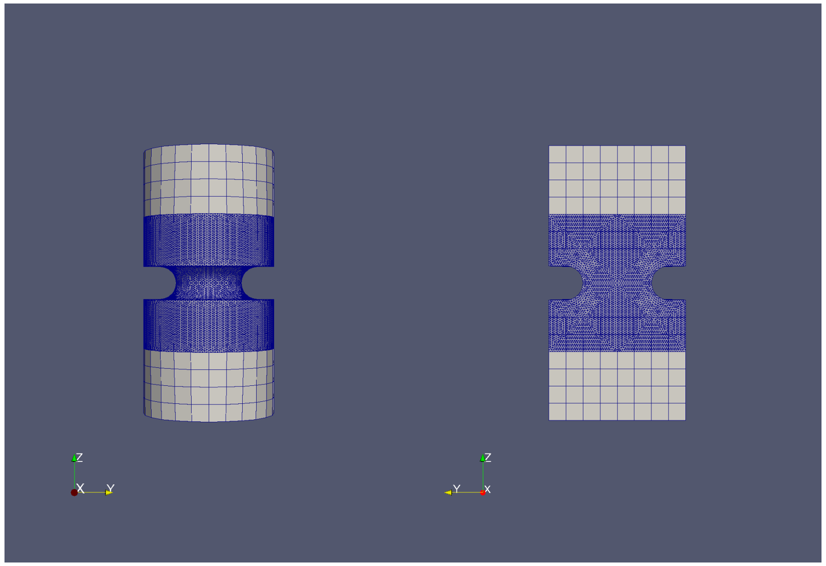
Schwarz coupling: concurrent multiscale

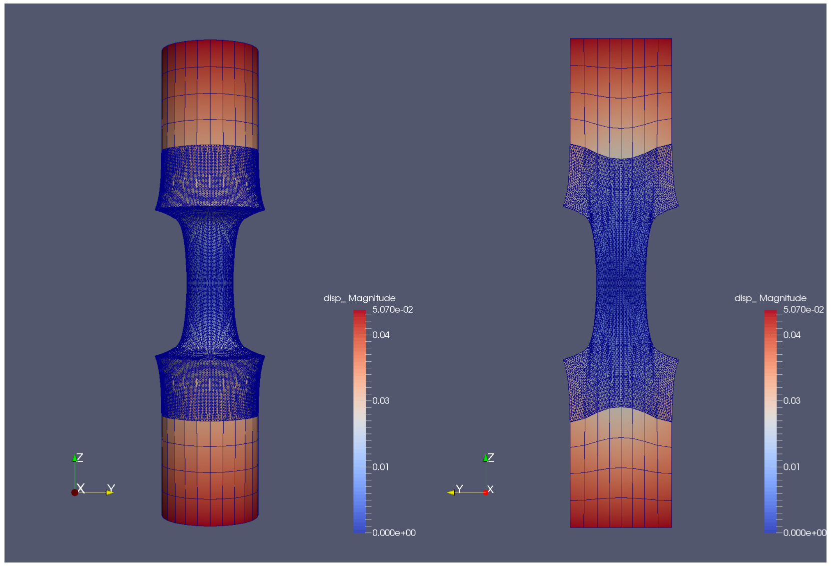
- Concurrent multiscale methods for solid mechanics are essential for the understanding and prediction of behavior of engineering systems when a small scale event will eventually determine the performance of the entire system.
- The Schwarz coupling method has been adapted and implemented for use in concurrent multiscale modeling in Albany, thus allowing the study of models where information is exchanged back and forth between small and large scales.



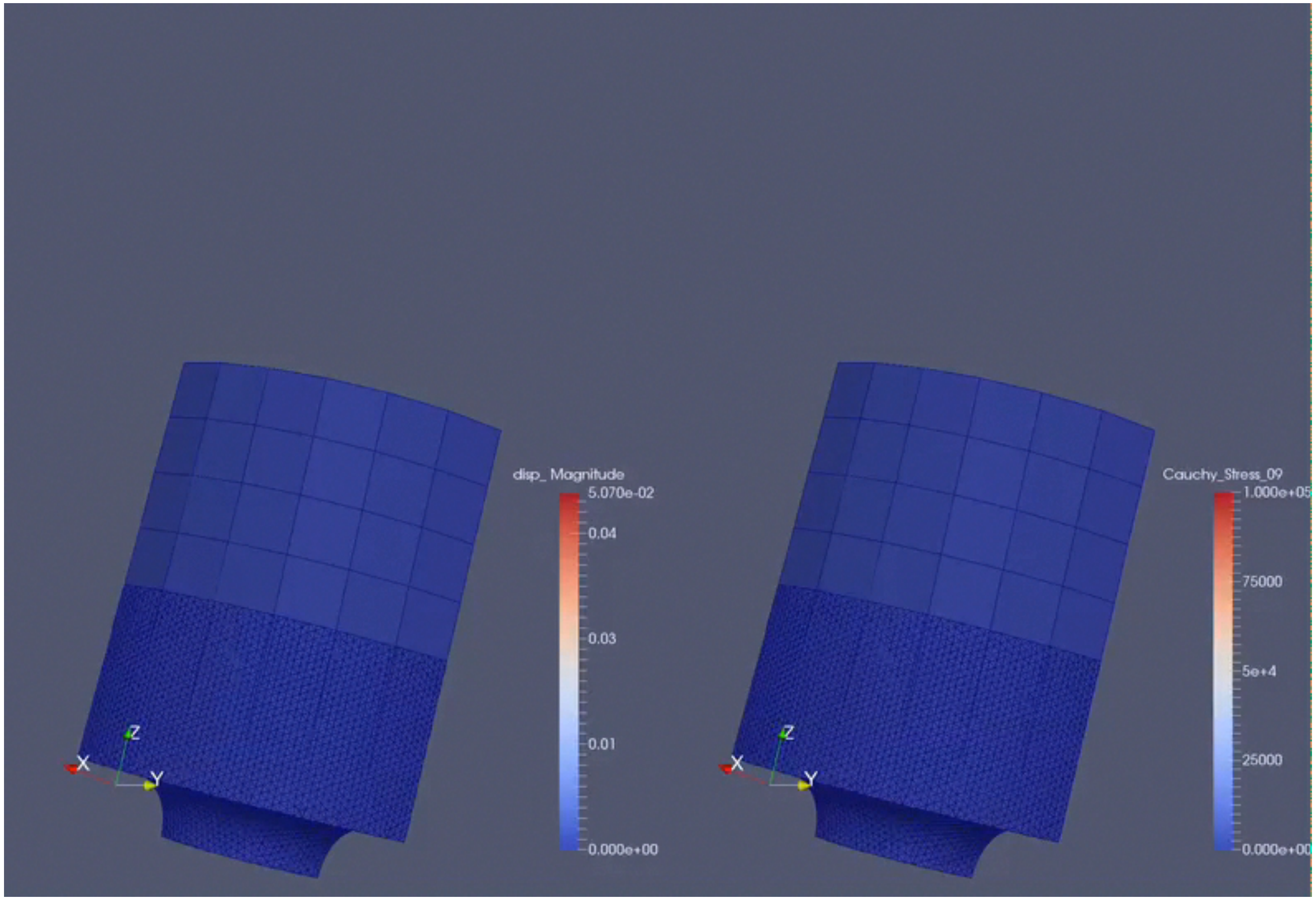
Schwarz coupling: notched cylinder

- The method is capable of coupling different mesh topologies.
- The figure below shows the results of pulling on a notched cylinder.
- The notched region, where stress concentrations are expected, is finely meshed with tetrahedral elements.
- The top and bottom regions, presumably of less interest, are meshed with coarser hexahedral elements.
- This provides the flexibility of using more refined meshes in critical regions of complex geometry and phenomena where tetrahedral meshers excel, and less refined meshes away from critical regions that can be meshed by hexahedra that are traditionally less amenable to very complex geometries.





Schwarz coupling: movie of half notched cylinder



Schwarz coupling: summary of findings

- Preconditioners for Schwarz: the method works with a variety of preconditioners provided with Trilinos, or none at all for very simple problems. To obtain the best performance, however, Trilinos provides a block system preconditioner package called Teko. The Schwarz method can be formulated as a block system of equations, and therefore the best preconditioner to use is Teko.
- Now that each mesh is its own independent application from the point of view of Albany, this opens the possibility of using different discretizations altogether for each, as shown.
- This demonstration calculation used linear tetrahedral elements for the fine region and trilinear hexahedra for the coarse region.
- It is known that linear tetrahedra are notoriously bad at representing stress, specially under large distortion, thus the non-smooth stress field near the notch in this demonstration calculation. The emphasis here is in the ability to couple different mesh topologies, which is a first for Albany.
- We have started preliminary performance studies of Albany under Kokkos using OpenMP. Kokkos can use a variety of alternate accelerators such as OpenMP, Cuda, Intel Phi, etc.

Performance portability using *Kokkos*

Strategy to extend *Albany* to provide good performance *on **new architecture machines*** (hybrid systems) and ***manycore devices*** (multi-core CPU, NVIDIA GPU, Intel Xeon Phi, etc.)

- ***Kokkos***: *Trilinos* library and programming model that provides performance portability across diverse devises with different memory models.
- The *Kokkos* strategy: write an algorithm once, and change a template parameter to get the optimal data layout for the target hardware.

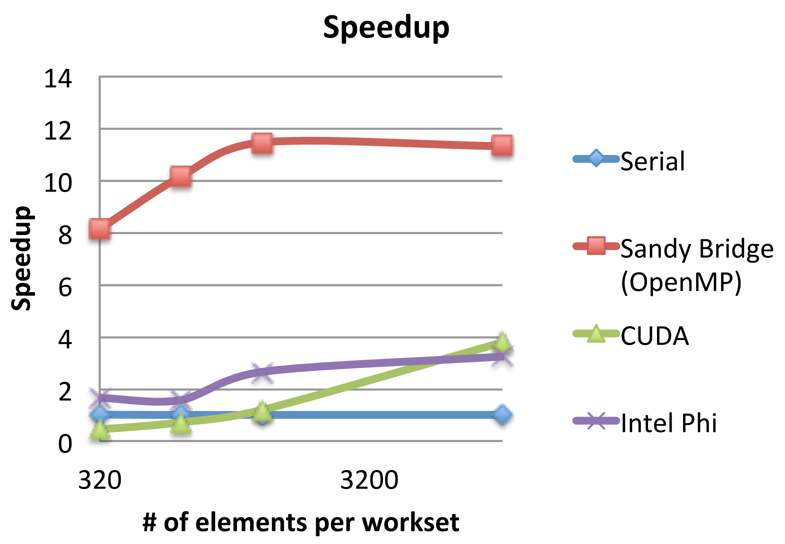
Performance portability details

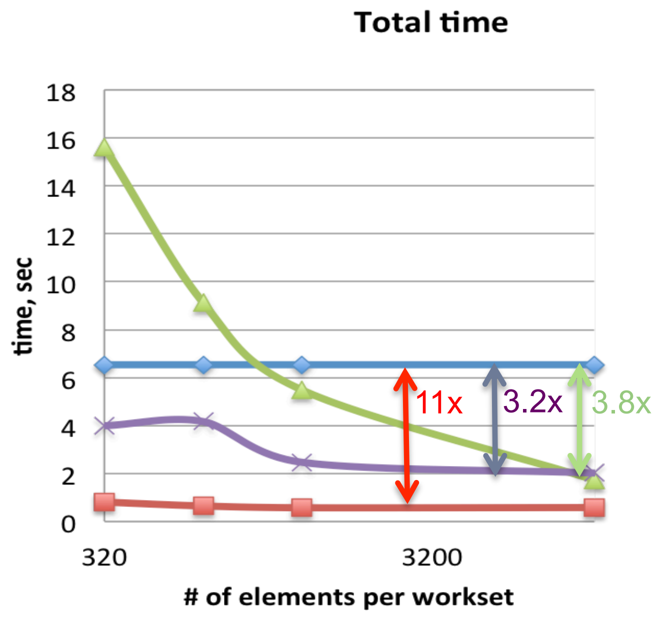
- The Kokkos programming model focuses on performance portability of kernels.
- Kokkos uses the ExecutionSpace parameter to tailor code for a device:
    1. Memory layout for the MultiDimVector (Accessor syntax v(i, j, k) is unchanged.)
    2. Parallel kernel launch directives under the Kokkos::Parallel_for() call.
- New architectures handled by new ExecutionSpace parameters being implemented in Kokkos.
    - Leverage ongoing DOE investments in hardware and software
- Kokkos implementation requires separating thread-safe kernel into separate function that is launched with integer loop index (e.g., cell).

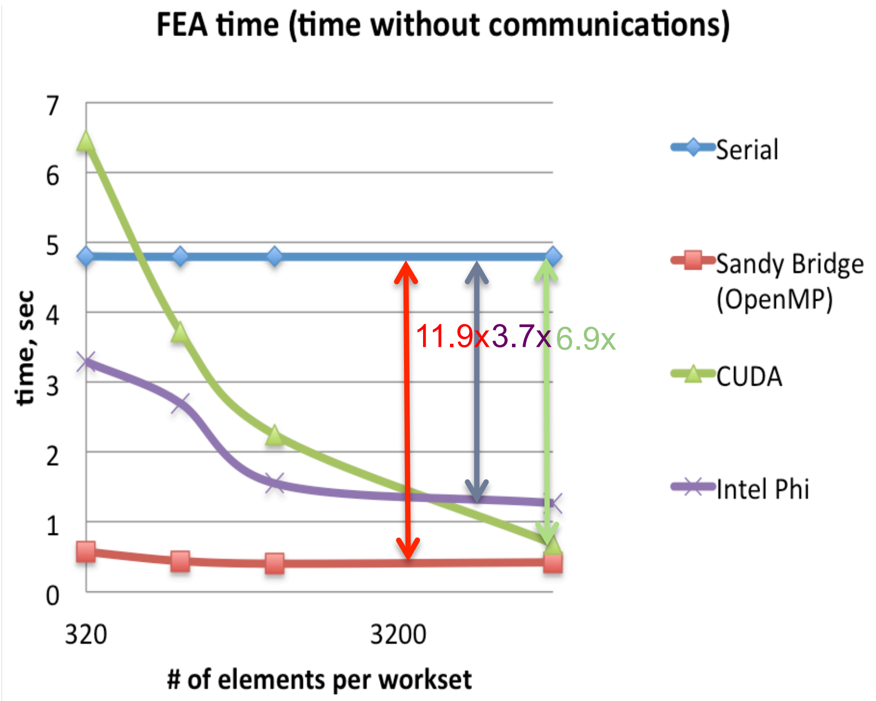
Kokkos-ification of FE assembly

```
typedef Kokkos::OpenMP ExecutionSpace;
//typedef Kokkos::CUDA ExecutionSpace;
//typedef Kokkos::Serial ExecutionSpace;
template<typename ScalarT>
vectorGrad<ScalarT>::vectorGrad()
{
Kokkos::View<ScalarT****, ExecutionSpace> vecGrad("vecGrad", numCells, numQP, numVec, numDim);
}
**********************************************
template<typename ScalarT>
void vectorGrad<ScalarT>::evaluateFields()
{
  Kokkos::parallel_for<ExecutionSpace> (numCells, *this);
}
**********************************************
template<typename ScalarT>
KOKKOS_INLINE_FUNCTION
void vectorGrad<ScalarT>:: operator() (const int cell) const
{
  ~~for (int cell = 0; cell < numCells; cell++)~~
  for (int qp = 0; qp < numQP; qp++) {
    for (int dim = 0; dim < numVec; dim++) {
      for (int i = 0; i < numDim; i++) {
        for (int nd = 0; nd < numNode; nd++) {
          vecGrad(cell, qp, dim, i) += val(cell, nd, dim) * basisGrad(nd, qp, i);
} } } } }
```

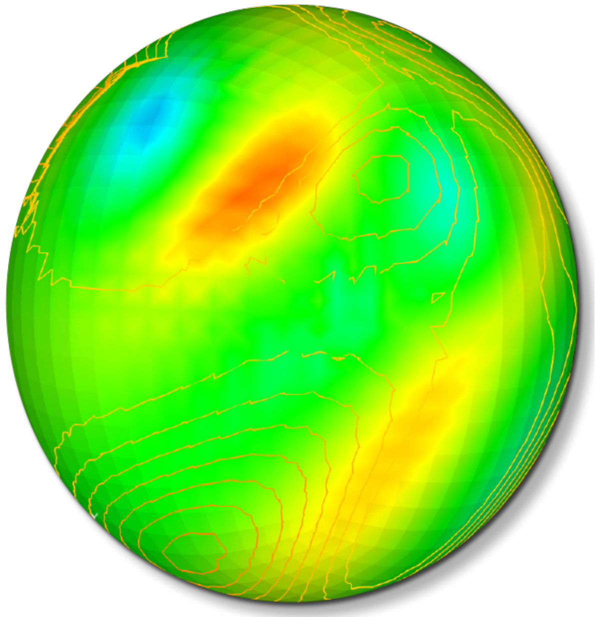
Multicore scaling using Kokkos





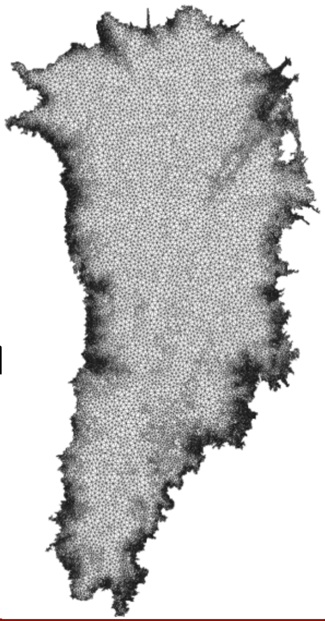


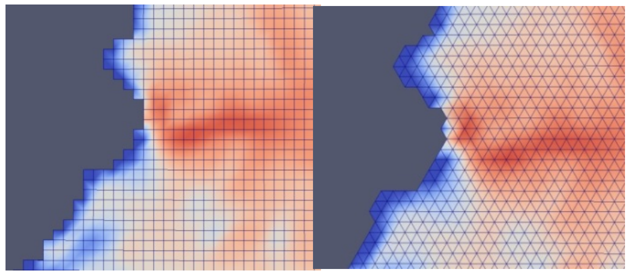
Climate models: Aeras and FELIX



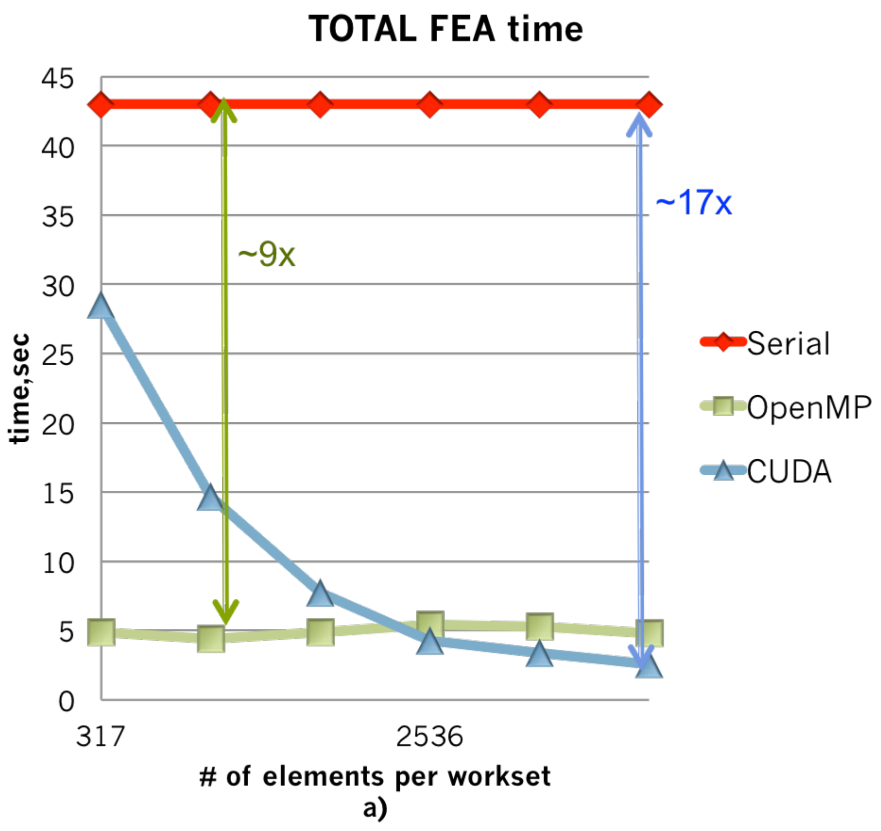
- Next generation global atmosphere model
- Model development: shallow water, X-Z hydrostatic, 3D hydrostatic, clouds, 3D non-hydrostatic

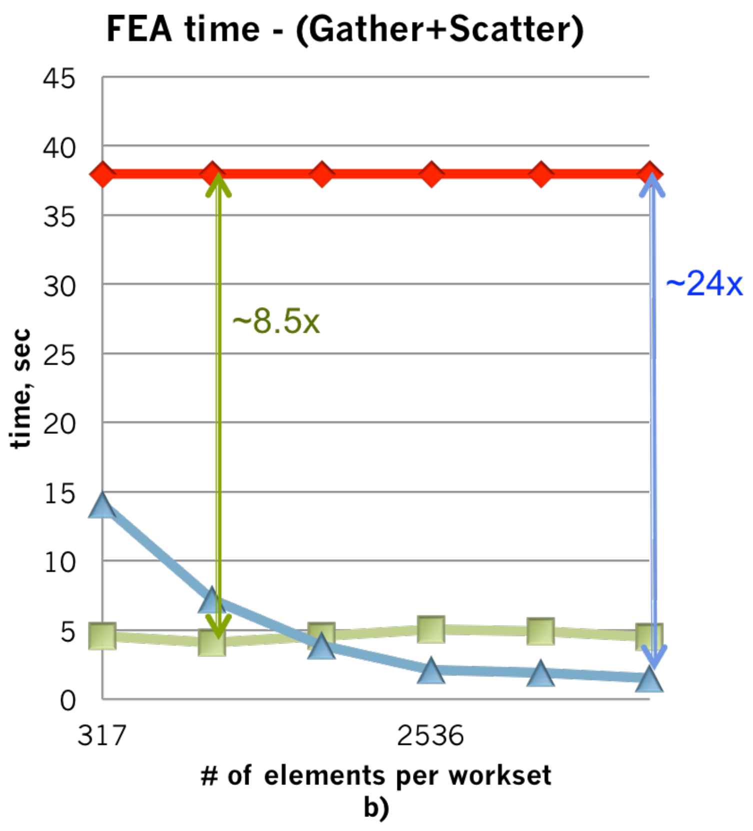
- Finite Elements for Land Ice eXperiments
- First order Stokes (Glen's law), stress/velocity solution of ice flow over Greenland





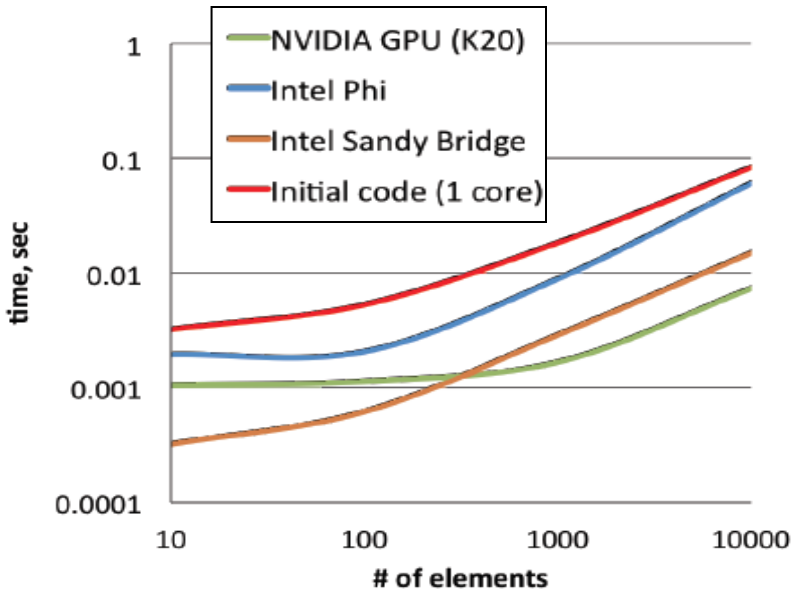
Aeras performance and portability





FELIX performance portability results

- ***Right:*** results for a **mini-app** that uses finite element kernels from *Albany/FELIX* but none of the surrounding infrastructure.
  - "# of elements" = threading index (allows for on-node parallelism).
  - # of threads required before the Phi and GPU accelerators start to get enough work to warrant overhead: ~100 for the Phi and ~1000 for the GPU.

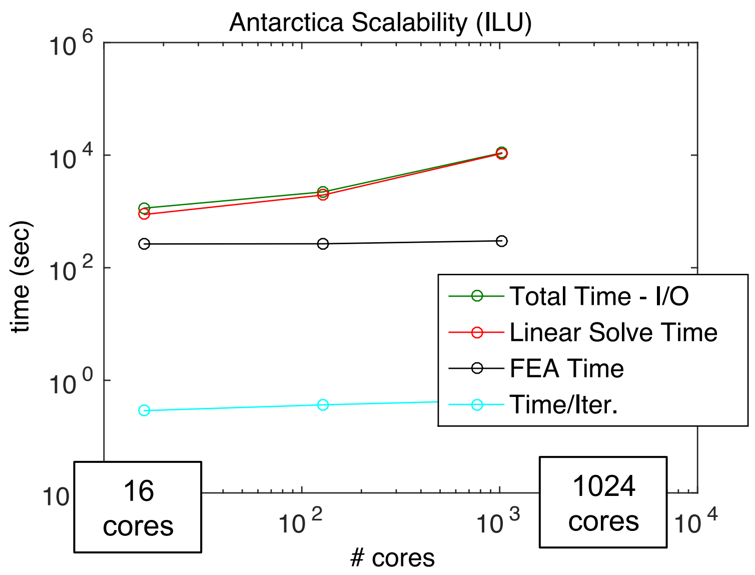


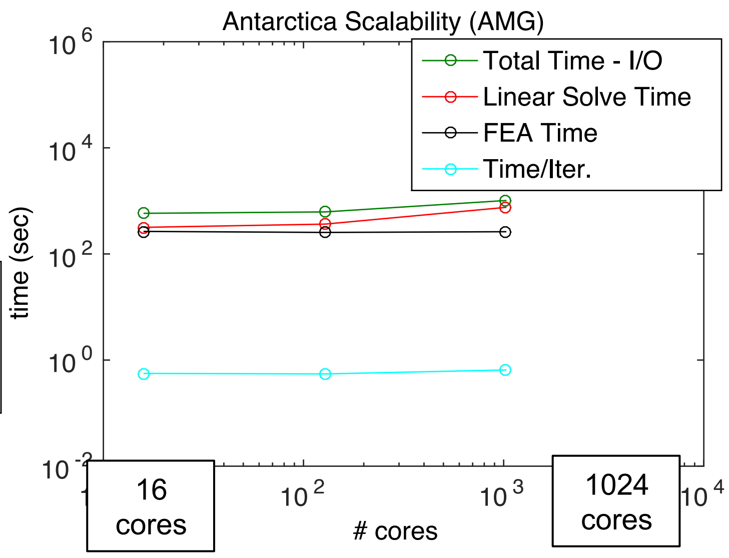
- ***Below:*** preliminary results for 3 of the finite element assembly kernels, as part of **full *Albany/FELIX*** code run.

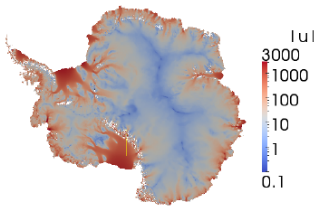
Kernel	Serial	16 Threads OpenMP	GPU
Viscosity Jacobian	20.39 s	2.06 s	0.54s
Basis Functions w/ FE Transforms	8.75 s	0.94 s	1.23s
Gather Coordinates	0.097 s	0.107 s	5.77s

Note: *Gather Coordinates routine requires copying data from host to GPU.*

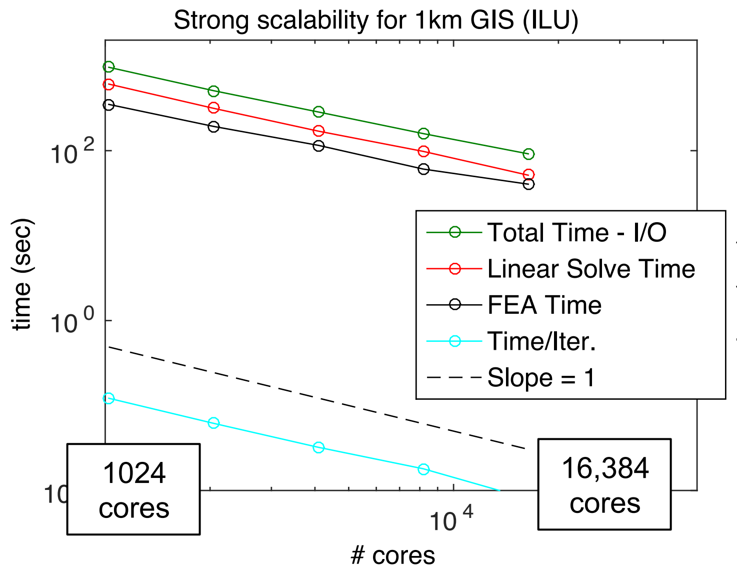
FELIX scalability

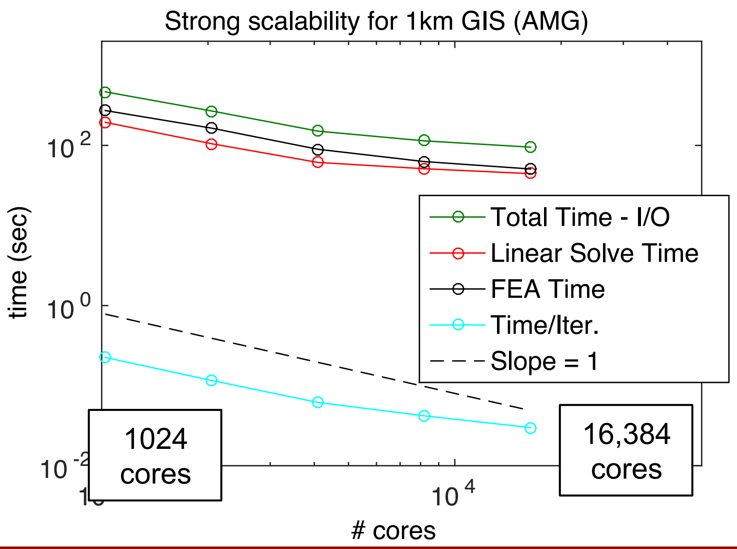


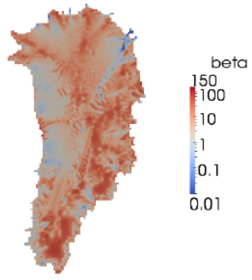




Weak scaling: 8km, 5 layers to 2km, 20 layers







Strong scaling: 143M DOFs

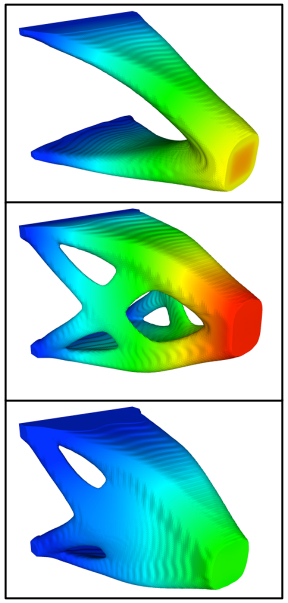
Embedded analysis algorithms

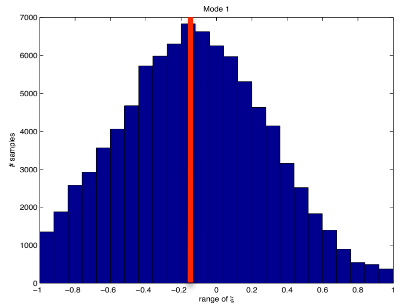
Strong potential exists to improve analysis capabilities *beyond forward simulation*:

- *Parameter Studies; Sensitivity Analysis, Bifurcation and Linear Stability Analysis; Optimization, Calibration, and Inversion; Uncertainty Quantification*

Ensembles of calculations are a key tool in uncertainty analysis:

- Random sampling, stochastic collocation, …
- Computes probability distribution of responses with respect to probabilistic inputs
  - Ranges of properties, geometries, parameters, forcing calculations
- Ensembles typically implemented as an outer loop around the code execution



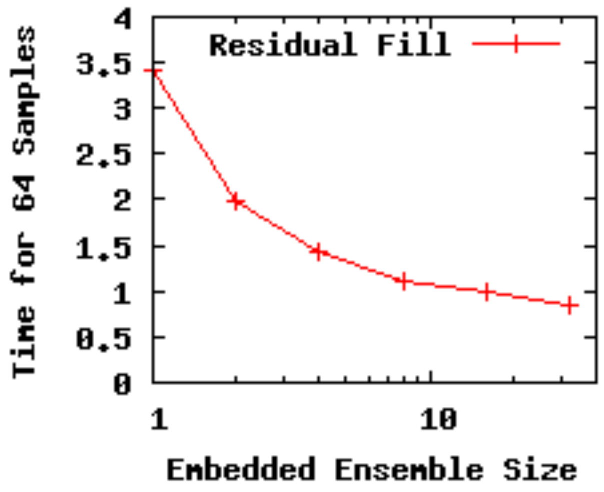


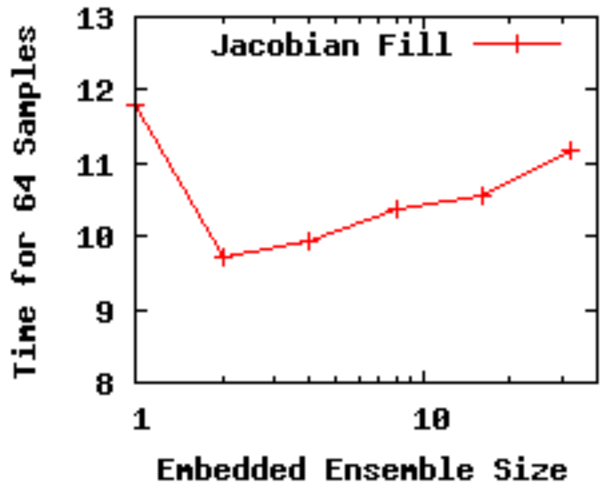
Exascale embedded ensembles

Embedding ensembles moves them to an inner loop.

Performance gains are realized by:

- Amortizing costs for mesh-dependent calculations
- Supports compiler vectorization of kernels over ensembles
- Amortizing latency over larger MPI messages
- Contiguous memory access for arrays of data





Timings for computing 64 total ensemble members with varying embedded sample sizes. Solving 3D Elasticity problem over a range of elastic modulus and Poisson ratio. *Recent evidence shows larger speedups on Phi, GPU.*

Ack: E. Phipps; ASCR Equinox project

Embedding ensembles in FE assembly

```
typedef Stokhos::Ensemble<double> ScalarT;
```

```
typedef Kokkos::OpenMP ExecutionSpace;
//typedef Kokkos::CUDA ExecutionSpace;
//typedef Kokkos::Serial ExecutionSpace;
template<typename ScalarT>
vectorGrad<ScalarT>::vectorGrad()
{
Kokkos::View<ScalarT****, ExecutionSpace> vecGrad(“vecGrad”, numCells, numQP, numVec, numDim);
}
**********************************************
template<typename ScalarT>
void vectorGrad<ScalarT>::evaluateFields()
{
  Kokkos::parallel_for<ExecutionSpace> (numCells, *this);
}
**********************************************
template<typename ScalarT>
KOKKOS_INLINE_FUNCTION
void vectorGrad<ScalarT>:: operator() (const int cell) const
{
  ~~for (int cell = 0; cell < numCells; cell++)~~
  for (int qp = 0; qp < numQP; qp++) {
    for (int dim = 0; dim < numVec; dim++) {
      for (int i = 0; i < numDim; i++) {
        for (int nd = 0; nd < numNode; nd++) {
          vecGrad(cell, qp, dim, i) += val(cell, nd, dim) * basisGrad(nd, qp, i);
} } } } }
```

Summary

- Created a tightly coupled thermomechanics demonstration problem possessing complex temperature dependent properties
- Demonstrate proof-of-principle of parallel embedded meshing technology on quasi-static thermomechanics example
- Show portability of ATDM agile components on conventional MPI, Nvidia Cuda GPUs, Intel multicore, and Intel Phi coprocessors.
- Show principle of parallel embedded ensemble UQ calculations using component technologies
- Open source: **http://gahansen.github.io/Albany/**