Achieving and maintaining performance and performance portability within the Albany multi-physics code: perspectives & tools

Irina Tezaur¹, Max Carlson¹, Jerry Watkins¹, Mauro Perego¹, Kyle Shan², Carolyn Kao³, Kim Liegeois¹

¹ Sandia National Laboratories, Livermore, CA, USA.
² Micron Technology, Boise, ID, USA.
³ TSMC, Hsinchu City, Taiwan.

SIAM CS&E 2023, Amsterdam, Netherlands.
February 26 - March 3, 2023
The Albany Code Base

**Albany**: open-source\(^1\) parallel C++ unstructured-grid multi-physics finite element code built for **rapid application development** from Trilinos\(^2\) Agile Components

Distinguishing features of **Albany**:

- Funded entirely by applications residing within
- Both a “sand-box” for prototyping new approaches and a production code
- Algorithms/software are developed & matured directly on applications
- Applications are “born” scalable, fast, robust, performance-portable, and...
- Equipped with **embedded advanced analysis capabilities**: sensitivities, bifurcation analysis, adjoint-based inversion, ...

\(^1\) [https://github.com/sandialabs/Albany](https://github.com/sandialabs/Albany)

\(^2\) [https://github.com/trilinos/Trilinos](https://github.com/trilinos/Trilinos)
History of Albany

Since its creation ~2009, Albany has housed many **diverse algorithmic projects & applications**:  

- Demo PDEs  
- Quantum Devices (QCAD)  
- Ice Sheets (Albany Land-Ice)  
- Mechanics (LCM)  
- Atmosphere Dynamics (Aeras)  
- Particle-continuum coupling (Peridigm-LCM)  
- Additive Manufacturing Design (ATO)  
- Additive Manufacturing Processing (AMP)  
- Arctic Coastal Erosion (ACE)  
- Coupled Geomechanics (Albotran)


Andy Salinger  
“Father” of Albany
Albany and its Requirements Today

Primary customer/funder: U.S. DOE SciDAC-funded project land-ice modeling project, FAnSSIE (Framework for Antarctic System Science in E3SM, FY23-FY27)

- Albany houses the velocity solver of MALI (MPAS-Albany Land Ice), the land-ice component of the U.S. DOE’s Energy Exascale Earth System Model (E3SM)

Required capabilities in Albany for science using MALI:

- Easy way to add new physics/PDEs
- Performance portability to advanced heterogeneous platforms
- Fast, scalable and robust linear solves across different architectures
- PDE-constrained optimization capabilities for ice sheet inversion
- Adaptive mesh refinement (AMR) of extruded tetrahedral meshes

Requirements for software quality:

- Pruning of unfunded applications/unsupported code
- Version control
- Automated parameter tuning for various architectures
- Automated nightly regression and performance testing using VOTD Albany & Trilinos

See MS284 and MS318 on Thur. Mar. 2 for more on MALI and ice-sheet modeling.
Albany and its Requirements Today

Primary customer/funder: U.S. DOE SciDAC-funded project land-ice modeling project, FAnSSIE (Framework for Antarctic System Science in E3SM, FY23-FY27)

- Albany houses the velocity solver of MALI (MPAS-Albany Land Ice), the land-ice component of the U.S. DOE’s Energy Exascale Earth System Model (E3SM)

Required capabilities in Albany for science using MALI:

- Easy way to add new physics/PDEs
- Performance portability to advanced heterogeneous platforms
- Fast, scalable and robust linear solves across different architectures
- PDE-constrained optimization capabilities for ice sheet inversion
- Adaptive mesh refinement (AMR) of extruded tetrahedral meshes

Requirements for software quality:

- Pruning of unfunded applications/unsupported code
- Version control
- Automated parameter tuning for various architectures
- Automated nightly regression and performance testing using VOTD Albany & Trilinos

This talk: highlight capabilities for achieving/maintaining performance & performance portability in Albany, with focus on pros/cons and lessons learned.
Outline

• Albany capabilities & supporting tools
  ➢ Components effort
  ➢ Albany under the hood
  ➢ Kokkos for performance portability
  ➢ Phalanx for template-based evaluators and DAG-based finite element assembly
  ➢ Sacado for automatic differentiation (AD)
  ➢ Some performance & performance portability results

• Maintaining software quality of Albany
  ➢ Code pruning
  ➢ PyAlbany: a Python interface to Albany
  ➢ Nightly regression testing
  ➢ Automated performance testing
  ➢ Automated performance tuning

• Summary & perspectives
Outline

• Albany capabilities & supporting tools
  ➢ Components effort
  ➢ Albany under the hood
  ➢ Kokkos for performance portability
  ➢ Phalanx for template-based evaluators and DAG-based finite element assembly
  ➢ Sacado for automatic differentiation (AD)
  ➢ Some performance & performance portability results

• Maintaining software quality of Albany
  ➢ Code pruning
  ➢ PyAlbany: a Python interface to Albany
  ➢ Nightly regression testing
  ➢ Automated performance testing
  ➢ Automated performance tuning

• Summary & perspectives
Many components are **Trilinos** packages:

- Mesh tools (**STK**)
- Discretization tools (**Intrepid2**)
- Nonlinear/Linear solver (**NOX/Belos**)
- Distributed memory linear algebra (**Tpetra**)
- Multigrid Preconditioner (**MueLu**)
- Field DAG (**Phalanx**)
- Automatic differentiation (**Sacado**)
- Shared memory parallelism (**Kokkos**)
- Many more...

**Components in Albany** = cutting-edge technology from Trilinos, SierraToolKit, DAKOTA, FASTMath, Kitware, etc.

Albany developers work with **Version of the Day (VOTD)** Trilinos, which has pros and cons.

**Pro:** code can inherit unexpected enhancements “for free”

**Con:** code might also inherit bugs/regressions which can be frustrating to track down and get fixed to maintain clean dashboard

* https://github.com/trilinos/Trilinos

40+ packages; 120+ libraries
Albany Under the Hood

Albany provides the “glue” that connects components (via abstract interfaces).

Four Key Ingredients:

1. MPI+X programming model for performance portability
   - Handled by *Kokkos* package

2. Template-based generic programming (TBGP)
   - E.g., *Phalanx* evaluators templated on evaluation type (right)

3. Graph-based finite element assembly (FEA)
   - Handled by *Phalanx* package

4. Templated-based automatic differentiation
   - Handled by *Sacado* package

For more details on these ingredients, please see talk by Jerry Watkins (MS284: Thurs. Mar. 2).
Albany Supporting Tools: Kokkos – Performance Portability

• **Kokkos**\(^1\) is a C++ library that provides *performance portability* across multiple *shared memory* computing architectures via *MPI+X programming model*
  ➢ *Examples*: Multicore CPU, NVIDIA GPU, Intel KNL and much more...

• Abstract *data layouts* and *hardware features* for optimal performance on *current* and *future* architectures

• Allows researchers to focus on *application* or *algorithmic development* instead of *architecture specific programming*

**Pros:**
- Same code runs on diff. platforms
- Code is “future-proof”
- Forces you to write better code

**Cons:** better performance may be possible with architecture-specific optimizations

With Kokkos, you write an algorithm *once* for multiple hardware architectures. *Template parameters* are used to get *hardware specific features*.

\(^1\) [https://github.com/kokkos/kokkos](https://github.com/kokkos/kokkos).
Albany Supporting Tools: Phalanx Evaluator – Templated Phalanx Node

• A Phalanx node (evaluator) is constructed as a C++ class

• Each evaluator is templated on an evaluation type (e.g., residual, Jacobian)

• The evaluation type is used to determine the data type (e.g., double, Sacado data types)

Pros:
• Easy to implement new physics

```cpp
template<typename EvalT, typename Traits>
void StokesFOResid<EvalT, Traits>::
operator() (const int& cell) const{
    for (int cell=0; cell < numCells; cell++) {
        for (int node=0; node < numNodes; ++node){
            Residual(cell,node,0)=0.;
        }
    }
    for (int cell=0; cell < numCells; cell++) {
        for (int node=0; node < numNodes; ++node) {
            for (int qp=0; qp < numQPs; ++qp) {
                Residual(cell,node,0) += 
                Ugrad(cell,qp,0,0)*wGradBF(cell,node,qp,0) + 
                Ugrad(cell,qp,0,1)*wGradBF(cell,node,qp,1) + 
                force(cell,qp,0)*wBF(cell,node,qp);
            }
        }
    }
}
```
Albany Supporting Tools: Phalanx Evaluator – Templated Phalanx Node

- A Phalanx node (evaluator) is constructed as a C++ class.
- Each evaluator is templated on an evaluation type (e.g., residual, Jacobian).
- The evaluation type is used to determine the data type (e.g., double, Sacado data types).
- Kokkos `RangePolicy` is used to parallelize over cells over an Execution Space (e.g., Serial, OpenMP, CUDA).
- Inline functors are used as kernels.
- MDField data layouts
  - Serial/OpenMP – LayoutRight (row-major)
  - CUDA – LayoutLeft (col-major)

```cpp
typedef Kokkos::CUDA ExeSpace;

template<typename EvalT, typename Traits>
void StokesFOResid<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset) {
  Kokkos::parallel_for(
    Kokkos::RangePolicy<ExeSpace>(0, workset.numCells),
    *this);
}

template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void StokesFOResid<EvalT, Traits>::
operator() (const int& cell) const{
  for (int node=0; node < numNodes; ++node) {
    Residual(cell, node, 0) = 0.0;
    for (int qp=0; qp < numQPs; ++qp) {
      for (int cell=0; cell < numCells; cell++) {
        Ugrad(cell, qp, 0, 0) * wGradBF(cell, node, qp, 0) +
        Ugrad(cell, qp, 0, 1) * wGradBF(cell, node, qp, 1) +
        force(cell, qp, 0) * wBF(cell, node, qp);
      }
    }
  }
}
```

**Pros:**
- Easy to implement new physics
- Easy to Kokkos-ize
Albany Supporting Tools: Phalanx – Directed Acyclic Graph (DAG)

**Pros:**
- Increased flexibility, extensibility, usability
- Arbitrary data type support
- Potential for task parallelism

**Cons:** Performance loss through fragmentation
- **Mitigation:** introduction of memoization

**DAG Example**

- Scatter
- Residual
- Interpolate Parameter
- Gather Parameter
- Interpolate Solution
- Basis Functions
- Gather Solution
- Gather Coordinates

**DAG Example (memoization)**

**Pros:**
- Increased flexiblity, extensibility, usability
- Arbitrary data type support
- Potential for task parallelism

**Cons:** Performance loss through fragmentation
- **Mitigation:** introduction of memoization

**Diagram**

![Diagram showing DAG example with memoization](image)

**Figure**

- Improvements:
  - \(a\): Base
  - \(b\): Memoization

<table>
<thead>
<tr>
<th></th>
<th>SMAssembly</th>
<th>DMAssembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswell 16MPI</td>
<td>2.0x</td>
<td>a</td>
</tr>
<tr>
<td>Haswell 16(MPI+2OMP)</td>
<td>2.0x</td>
<td>a</td>
</tr>
<tr>
<td>KNL 68(MPI+4OMP)</td>
<td>2.1x</td>
<td>b</td>
</tr>
<tr>
<td>P100 1(MPI+GPU)</td>
<td>3.9x</td>
<td>3.9x</td>
</tr>
</tbody>
</table>

**Legend**

- Single CPU socket or GPU
Albany Supporting Tools: Sacado – Automatic Differentiation (AD)

**Pros:**
- AD provides exact derivatives – no Jacobian derivation or hand-coding required
- Allows for advanced analysis capabilities – easily construct any derivative, Hessian
  - Examples: optimization, sensitivity analysis

**Con:** AD has some overhead
- **Mitigation:** specify Sacado data types for deriv. components via class templates
  - DFad (most flexible) – size set at run-time
  - SLFad (flexible/efficient) – max size set at compile-time
  - SFad (most efficient) – size set at compile-time

**Fad Type Comparison (Serial, OpenMP (12 threads), CUDA)**

**Significant speedups** possible when deriv. array sizes are known at compile time on GPU (50-250x)

**Size Example:** Tetrahedral elements (4 nodes), 2 equations, ND = 4*2 = 8
Performance Portability Demonstration: Antarctica Weak Scalability Study

Architectures:
- NERSC Cori-Haswell (HSW): 32 cores/node
- NERSC Cori-KNL (KNL): 68 cores/node
- OLCF Summit-POWER9-only (PWR9): 44 cores/node
- OLCF Summit-POWER9-V100 (V100): 44 cores/node + 6 GPU/node

Benchmark:
- First-order Stokes solve in ALI
- Structured hexahedral element mesh
- 16 to 1km structured Antarctica meshes, 20 layers
- Scaled up from 1 to 256 compute nodes

Mesh Example: 16km, structured Antarctica mesh (2.20E6 DOF: 20 layer, 2 equations)

Benchmark used to assess performance & performance portability.
Performance on Cori and Summit

Setup:
• Same input file for all cases
  ➢ Performance portable point smoothers
  ➢ No architecture specific tuning

Results:
• Performance degrades at higher resolutions
  ➢ (645 → 1798 total linear iterations)
  ➢ GPU scaling slightly better
• Speedup on GPU
  ➢ 3.2-4.1x speedup Summit over Cori
  ➢ 2.1-2.3x speedup V100 over POWER9

Speedup achieved over MPI-only simulations without architecture specific tuning!

For additional results/analysis, please see talks by Jerry Watkins (MS284: Thurs. Mar. 2) and Mauro Perego (MS72: Tues. Feb. 28).
Performance Highlights

Major improvements to finite element assembly time:

- Memoization to avoid unnecessary data movement and computation
- Boundary condition refactor to reduce memory footprint and data movement
- Tpetra::FECrsMatrix refactor to reduce memory footprint and data movement

Solver portability on Cori and Summit:

- MueLu SemiCoarsen refactor using Kokkos
- Ifpack2 portable smoothers tuned to GPU hardware

From architecture-agnostic improvements to Albany FEA.

From Trilinos enhancements, which Albany inherited.

Albany performance improves in time due to Albany’s development model and maintenance workflows.

Outline

• Albany capabilities & supporting tools
  ➢ Components effort
  ➢ Albany under the hood
  ➢ Kokkos for performance portability
  ➢ Phalanx for template-based evaluators and DAG-based finite element assembly
  ➢ Sacado for automatic differentiation (AD)
  ➢ Some performance & performance portability results

• Maintaining software quality of Albany
  ➢ Code pruning
  ➢ PyAlbany: a Python interface to Albany
  ➢ Nightly regression testing
  ➢ Automated performance testing
  ➢ Automated performance tuning

• Summary & perspectives
Albany in February 2019

https://github.com/SNLComputation/Albany

➢ Demo PDEs
➢ Quantum Devices (QCAD)
➢ Ice Sheets (Albany Land-Ice)
➢ Mechanics (LCM)
➢ Atmosphere Dynamics (Aeras)
➢ Particle-continuum coupling (Peridigm-LCM)
➢ Additive Manufacturing Design (ATO)
➢ Additive Manufacturing Processing (AMP)
➢ Arctic Coastal Erosion (ACE)
➢ Coupled Geomechanics (Albotran)

Albany Today

---

**New capability:** PyAlbany, a python wrapper for running Albany!

Main Albany repo*: [https://github.com/sandialabs/Albany](https://github.com/sandialabs/Albany)
Tags: [https://github.com/sandialabs/Albany/tags](https://github.com/sandialabs/Albany/tags)

Albany-LCM repo*: [https://github.com/sandialabs/LCM](https://github.com/sandialabs/LCM)

Albany-RPI repo: [https://github.com/scorec/albany](https://github.com/scorec/albany)

* Maintained & tested nightly by Sandia teams.

---

**Lesson learned:** supporting unfunded capabilities in large HPC code is not sustainable long-term...

---

New capability: Albany is now a spack-age ([https://github.com/E3SM-Project/spack.git](https://github.com/E3SM-Project/spack.git))

---

- **Demo PDEs**
- Quantum Devices (QCAD)
- **Ice Sheets (MPAS-Albany Land Ice)**
- Mechanics (LCM)
- Atmosphere Dynamics (Aeras)
- Particle-continuum coupling (Peridigm-LCM)
- Additive Manufacturing Design (ATO)
- Additive Manufacturing Processing (AMP)
- Arctic Coastal Erosion (ACE)
- Coupled Geomechanics (Albotran)
Lesson learned: PyAlbany allows to easily and quickly...

• Use Albany without C++ or bash knowledge (convenient for students)
• Prototype applications that require multiple Albany solves
• Enable fast pre-processing and post-processing in Python
• Use Python as a glue language to couple Albany with other software: for UQ methods (PyDakota), machine learning (TensorFlow, Keras, Scikit-learn), plotting (Matplotlib, Paraview), ...

Running Albany through *PyAlbany* is faster for MCMC analysis! (don’t need to go through the setup phase of Albany for every sample)

Please see talk by **Kim Liegeois** (MS284: Thurs. Mar. 2) for more on PyAlbany.

Lesson learned: since MALI uses Trilinos and Albany Versions of the Day (VOTD), nightly testing across variety of architectures is essential for maintaining code quality!

(Automated Regression) Testing, Testing, Testing

<table>
<thead>
<tr>
<th>Repository*</th>
<th>Nightly test harness</th>
<th>Mailing lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version control</td>
<td>Unit tests</td>
<td>Issue tracking</td>
</tr>
<tr>
<td>Build system</td>
<td>Verification tests</td>
<td>Web pages</td>
</tr>
<tr>
<td>Config mgmt</td>
<td>Code coverage</td>
<td>Licensing</td>
</tr>
<tr>
<td>Regression tests</td>
<td>Performance tests</td>
<td>Release process</td>
</tr>
</tbody>
</table>

* https://github.com/sandialabs/Albany
... and More (Automated Performance) Testing!

**Lesson learned:** nightly regression testing is *not sufficient*! Performance testing is also needed to secure investments in performance & portability in evolving software.

- Changes in code base could cause performance deterioration
- Performance improvements in one architecture could decrease performance in another
- Manual analysis is time consuming and imprecise

**Solution:** changepoint detection algorithm automatically applied to nightly performance test data to identify/flag large changes in performance.

- Infrastructure is provided in a Jupyter notebook, exported as html to a website ([https://sandialabs.github.io/ali-perf-data](https://sandialabs.github.io/ali-perf-data))
- Daily email provides nightly performance test summary

![Figure below: Total simulation time for a 2-20km resolution Antarctica problem, executed nightly](image)
Detecting Performance Regressions/Improvements

**Example:** transition to Kokkos 3.5.0 caused a performance regression but was soon fixed

**Regression**
- Date: 2021-11-06T00:00:00
- Albany commit: e14f44a
- Trilinos commit: e15c42
- Mean (99% CI): 2.04 (2.02, 2.05)
- Ratio (99% CI): 1.15 (1.14, 1.16)

**Improvement**
- Date: 2021-12-14T00:00:00
- Albany commit: 814a3d3
- Trilinos commit: f975d21
- Mean (99% CI): 1.78 (1.75, 1.80)
- Ratio (99% CI): 1.15 (1.13, 1.16)

**Total Fill time:** for a 1-to-7 km resolution Greenland mesh, executed nightly in Albany Land Ice.
Monitoring Performance Comparisons

**Example:** Memoization comparison (with & without) shows that relative performance has improved

Speedup of Total Fill time: from memoization for a 1-to-7 km resolution Greenland mesh, executed nightly in Albany Land Ice
Automated Parameter/Performance Tuning

**Lesson learned:** hand-tuning solver parameters can be a long/painful process, does not translate b/w architectures.

**Solution:** create a framework for determining optimal parameter values to achieve best performance (smallest CPU time) on HPC systems using offline and real-time data.

![Graph showing best runtimes for a range of total # allotted Albany run budgets: 3-to-20 km resolution Greenland mesh]

**Best runtimes for a range of total # allotted Albany run budgets:** 3-to-20 km resolution Greenland mesh

![Graph showing best runtimes using GPTune Bayesian optimization, Latin Hypercube Sampling and Monte Carlo Sampling: 3-to-20 km resolution Greenland mesh]

**Best runtimes using GPTune Bayesian optimization, Latin Hypercube Sampling and Monte Carlo Sampling:** 3-to-20 km resolution Greenland mesh

**Example:** autotuning used to improve performance of multigrid smoothers on GPU

Outline

• Albany capabilities & supporting tools
  ➢ Components effort
  ➢ Albany under the hood
  ➢ Kokkos for performance portability
  ➢ Phalanx for template-based evaluators and DAG-based finite element assembly
  ➢ Sacado for automatic differentiation (AD)
  ➢ Some performance & performance portability results
• Maintaining software quality of Albany
  ➢ Code pruning
  ➢ PyAlbany: a Python interface to Albany
  ➢ Nightly regression testing
  ➢ Automated performance testing
  ➢ Automated performance tuning
• Summary & perspectives
Summary & Perspectives

- In developing HPC codes, there is often a **tradeoff** between **flexibility** and **efficiency**. Libraries like **Kokkos** and **Trilinos** can enable both to a large extent.
  - Reasonable scalability and performance is obtained across different architectures **without** architecture-specific optimizations within the Albany code base
  - Code promised to be “future proof”

- **Maintaining performance** and **portability** is crucial for an active code base
  - Supporting **unfunded capabilities** in large HPC code is not sustainable long-term
  - Regular **regression** and **performance testing** is crucial, especially in presence of every-changing codes

- **Automatic** processes like **testing** and **parameter tuning** can save developers a lot of time
  - A **change-point detection algorithm** can help identify performance variation automatically
  - **Optimal solver parameters** can be determined for a specific architecture **automatically** using black-box optimization algorithms
Bridging the Gap between Sandia & Academia, and Promoting Inclusion & Diversity through Albany

Recently-funded 5-year Grande CARES MSIPP consortium\(^1\) will connect students/academics from minority serving institutions in U.S. southwest with Sandia National Laboratories via tools like Albany, Trilinos, Kokkos, ...

---

UTEP Receives $1.25M Grant from DOE to Produce Pipeline of Scientists and Engineers\(^2\)

Project will focus on recruiting and training scientists and engineers from underrepresented groups

---

\(^1\) [https://sites.google.com/view/grande-cares/home](https://sites.google.com/view/grande-cares/home)

Special Issue of CiSE on Research Software Engineers

CLOSED Call for Papers: Special Issue on the Future of Research Software Engineers in the US

“The goal of this special issue is to explore the future of research software engineers in the U.S., with emphasis on the cultural, educational, and professional paradigm shifts that need to occur.”*

Estimated publication of special CiSE issue: March/April 2024.

* https://www.computer.org/digital-library/magazines/cs/future-research-software-engineer
References


Support for this work was provided by Scientific Discovery through Advanced Computing (SciDAC) projects funded by the U.S. Department of Energy, Office of Science (OS), Advanced Scientific Computing Research (ASCR) and Biological and Environmental Research (BER).

Computing resources provided by the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF).
Start of backup slides
MALI (MPAS-Albany Land Ice) Software Ecosystem

**MPAS**\(^1\):
- Thickness & temperature evolution

**Albany Land Ice**\(^2\):
- First-order Stokes velocity solver

**Trilinos**\(^3\):
- Mesh tools (*STK*)
- Discretization tools (*Intrepid2*)
- Nonlinear/Linear solver (*NOX/ Belos*)
- Distributed memory linear algebra (*Tpetra*)
- Multigrid Preconditioner (*MueLu*)
- Field DAG (*Phalanx*)
- Automatic differentiation (*Sacado*)
- Shared memory parallelism (*Kokkos*)
- Many more...

---

\(^1\) [https://github.com/MALI-Dev/E3SM](https://github.com/MALI-Dev/E3SM).

\(^2\) [https://github.com/sandialabs/Albany](https://github.com/sandialabs/Albany).

\(^3\) [https://github.com/trilinos/Trilinos](https://github.com/trilinos/Trilinos).
First-Order (FO) Stokes Velocity Model

Ice behaves like a very viscous non-Newtonian shear-thinning fluid (like lava flow) and is modeled quasi-statically using nonlinear incompressible Stokes equations.

\[ \begin{align*}
-\nabla \cdot \tau + \nabla p &= \rho g \quad \text{in } \Omega \\
\nabla \cdot u &= 0
\end{align*} \]

FO Stokes \((u, v)\) in \(\Omega \in \mathbb{R}^3\)

\[ \begin{align*}
-\nabla \cdot (2\mu \dot{e}_1) &= -\rho g \frac{\partial s}{\partial x} \quad \text{in } \Omega \\
-\nabla \cdot (2\mu \dot{e}_2) &= -\rho g \frac{\partial s}{\partial y}
\end{align*} \]

- Fluid velocity vector: \(u = (u_1, u_2, u_3)\)
- Isotropic ice pressure: \(p\)
- Deviatoric stress tensor: \(\tau = 2\mu \varepsilon\)
- Strain rate tensor: \(\varepsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)\)
- Glen’s Law Viscosity: \(\mu = \frac{1}{2} A(T)^{-\frac{1}{n}} \left( \frac{1}{2} \sum_{ij} \varepsilon_{ij}^2 \right)^{\left( \frac{1}{2n} - \frac{1}{2} \right)}\)
- Flow factor: \(A(T) = A_0 e^{-\frac{Q}{RT}}\)

* Implicit solver: FEA* = 50% CPU-time
  Linear solve = 50% CPU-time

**First-Order Stokes model:** nice elliptic approximation to the full Stokes equations, derived using hydrostatic approximation & scaling argument based on ice sheets being thin with near-vertical normal

- **3D model** for two unknowns, the \((u_1, u_2)\) ice velocities.
- **Highly nonlinear** rheology with viscosity \(\mu\) given by Glen’s flow law
- **Valid for both Greenland and Antarctica** continental-scale simulations
Albany finite element assembly (FEA):
- **Tpetra** manages *distributed* memory linear algebra (MPI+X)
- **Phalanx** manages *shared* memory computations (X)
  - Gather fills element local solution
  - Interpolate solution/gradient to quad points
  - Evaluate residual/Jacobian
  - Scatter fills global residual/Jacobian

Adding new PDEs requires just implementing a new **Evaluate** routine

Albany design highlights:
- **Piro** manages the solve (e.g., Newton, time-stepper)
- Jacobians (+ sensitivities, Hessians, etc.) obtained via automatic differentiation (Sacado)
- **Kokkos** achieves performance portability using **MPI+X**

**Albany** provides the “glue” that connects components (via abstract interfaces).
Albany Supporting Tools: Belos/MueLu – Preconditioned Iterative Solver

**Problem:** Ice sheet meshes are thin with high aspect ratios

**Solution:** Matrix dependent semi-coarsening algebraic multigrid (MDSC-AMG)\(^1\)
- First, apply algebraic **structured** multigrid to coarsen vertically
- Second, apply **SA-AMG** on single layer

**Solver:** Preconditioned Newton-Krylov
- MDSC-AMG is used as preconditioner for GMRES
- Performance portability through Trilinos/MueLu (multigrid) + Trilinos/Belos (GMRES)

\(^1\) See (Tezaur *et al.*, 2015), (Tuminaro *et al.*, 2016)
Future Performance Overview

Applications
- E3SM + MALI
- MALI-standalone

DOE Exascale
- NERSC Perlmutter
- OLCF Frontier
- ALCF Aurora

Performance
- Algorithmic improvements
- Performance Optimization
Summary & Perspectives

- HPC architectures **changing rapidly** poses a significant challenge for mod/sim codes
  - The Albany, Trilinos and Kokkos software stack offers an efficient way to meet this challenge for large scale **finite element analysis**
  - There are both **pros** and **cons** to using ever-changing TPLs like these
    - **Pro:** code can inherit unexpected enhancements “for free”
    - **Con:** code might also inherit bugs/regressions which can be frustrating to track down and get fixed to maintain clean dashboard
  - **Performance portability** to next-generation architectures including GPUs is enabled via Kokkos
    - **Reasonable scalability** and **performance** can be obtained across different architectures **without** architecture-specific optimizations
    - Code promised to be **“future proof”**
- **Maintaining performance** and **portability** is crucial for an active code base
  - Supporting **unfunded capabilities** in large HPC code is not sustainable long-term
  - Regular **regression** and **performance testing** is crucial, especially in presence of every-changing codes
  - A **change-point detection algorithm** can help identify performance variation automatically
- **Optimal solver parameters** can be determined for a specific architecture **automatically** using black-box optimization algorithms