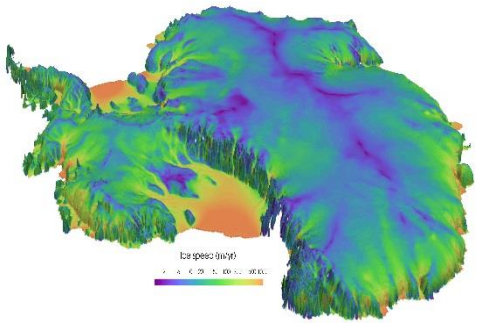
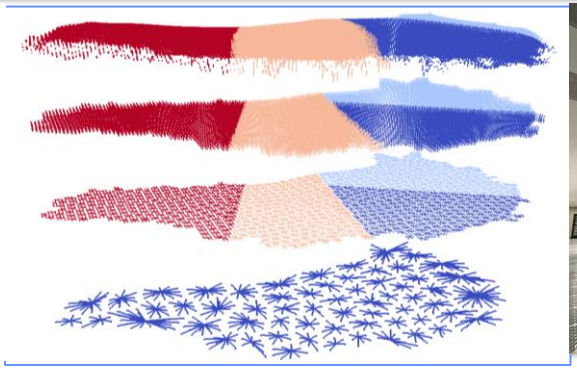
Exceptional service in the national interest

Sandia National Laboratories


Trilinos/Kokkos-based strategy towards achieving a performance portable land-ice model

Irina Tezaur[1], Jerry Watkins[1], Ray Tuminaro[1], Mauro Perego[1], Andy Salinger[1], Steve Price[2]

[1]Sandia National Laboratories. [2]Los Alamos National Laboratory.

BiRS Workshop: Math. Model. in Glaciology Banff, AB, Canada January 13-17, 2020

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

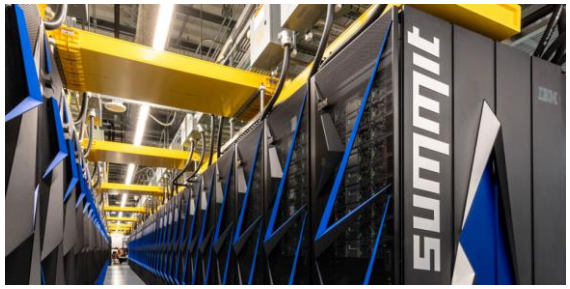
Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2020-0015C

Motivation for performance portability

Sandia National Laboratories

- Earth system models (ESMs) and ice sheet models (ISMs) need more ***computational power*** to achieve ***higher resolutions***.
- High performance computing (HPC) architectures are becoming increasingly more ***heterogeneous*** in a move towards ***exascale***.
- Climate models need to adapt to execute ***correctly*** & ***efficiently*** on new HPC architectures with drastically ***different memory models***.



An application is "**performance portable**" if it achieves a ***consistent*** level of ***performance*** across a ***variety*** of computer architectures.

Trends in HPC architectures

Sandia National Laboratories

CPUs:

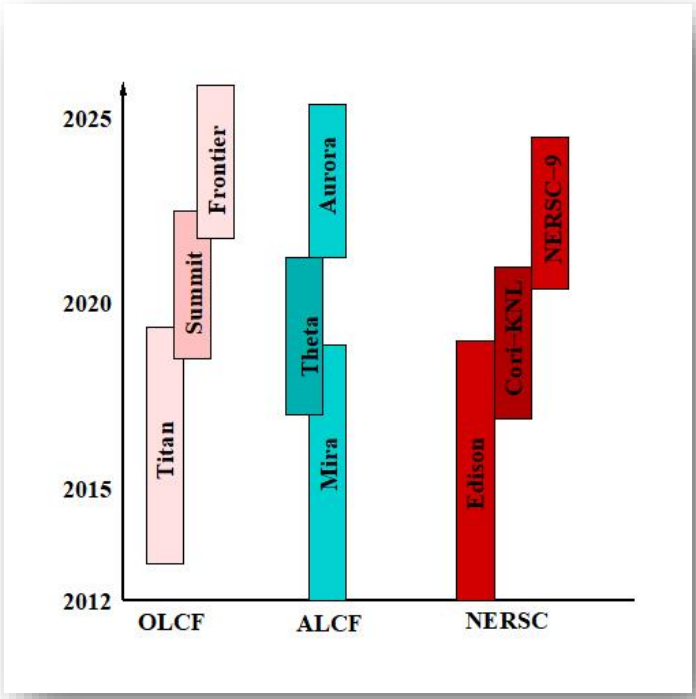
- Intel Xeon (Ivy Bridge, Haswell, Skylake), AMD Epyc, ARM

KNLs:

- NERSC Cori, ALCF Theta (→ 2021)
- Follow on KNH architecture cancelled

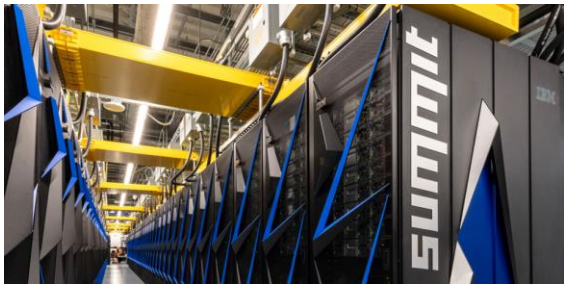
GPUs:

- OLCF Summit, 15MW of NVIDIA V100s
- NERSC Perlmutter (NVIDIA GPU+AMD CPU): 2020
- OLCF Frontier (AMD GPUs) in 2021: exascale in 2021 for 30MW
- ALCF Aurora (Intel GPUs): 2021



GPU and ***heterogeneous (CPU+GPU)*** architectures seem to be the ***future*** in moving towards ***exascale***.

- ***Computations*** are ***cheap***, ***memory transfer*** is ***expensive***.
- ***MPI alone*** is ***not enough*** to exploit available parallelism.



MPI+X programming model/approaches

Sandia National Laboratories

MPI+X Approach: *two levels of parallelism for CPU+accelerator*

- MPI for ***inter-node*** parallelism (for CPU)
- X for ***intra-node*** parallelism (for accelerator, e.g. GPU)

1. **Architecture specific approaches such as CUDA.**
   - Recent paper on FastICE v1.0 CUDA-based ISM: (Räss *et al., GMDD,* 2019).
   - Conversion to HIP for AMD GPU supposed to be simple, but not portable.
2. **Directive-based approaches such as OpenMP and OpenACC.**
   - Some success in achieving performance portability for climate, e.g. OpenACC port of HOMME atmosphere dycore (Norman *et al., J Comput. Sci.,* 2015).
   - Relies on compiler to address all the standards.
3. **Abstraction layers of data/task parallelism such as Kokkos and RAJA.**
   - Performance-portable, “future proof”: optimal data layout selected at compile-time.
   - Requires code to be written using C++.
   - Kokkos is path forward for DOE E3SM (SCREAM, ProSPect): (Demeshko *et al. J. HPC. Appl.*, 2018), (Bertagna *et al.*, *GMD,* 2019), (Watkins *et al. LNCSE,* 2020).

MPI+X programming model/approaches

Sandia National Laboratories

MPI+X Approach: *two levels of parallelism for CPU+accelerator*

- MPI for ***inter-node*** parallelism (for CPU)
- X for ***intra-node*** parallelism (for accelerator, e.g. GPU)

1. **Architecture specific approaches such as CUDA.**
   - Recent paper on FastICE v1.0 CUDA-based ISM: (Räss *et al., GMDD,* 2019).
   - Conversion to HIP for AMD GPU supposed to be simple, but not portable.
2. **Directive-based approaches such as OpenMP and OpenACC.**
   - Some success in achieving performance portability for climate, e.g. OpenACC port of HOMME atmosphere dycore (Norman *et al., J Comput. Sci.,* 2015).
   - Relies on compiler to address all the standards.

Our strategy and this talk!

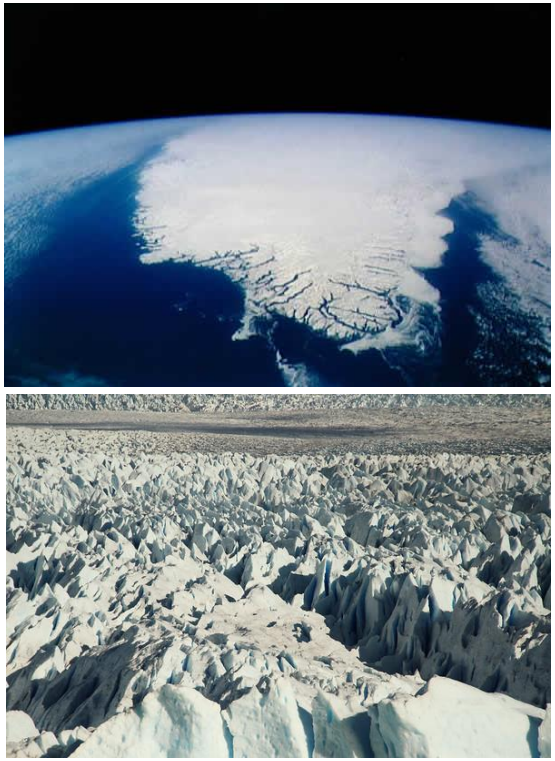
3. **Abstraction layers of data/task parallelism such as *Kokkos* and RAJA.**
   - Performance-portable, "future proof": optimal data layout selected at compile-time.
   - Requires code to be written using C++.
   - ***Kokkos*** is path forward for DOE E3SM (SCREAM, ProSPect): (Demeshko *et al. J. HPC. Appl.*, 2018), (Bertagna *et al.*, *GMD,* 2019), (Watkins *et al. LNCSE,* 2020).

Outline

Sandia National Laboratories

This talk describes our **efforts** towards creating a **performance portable** implementation of the **Albany/Land Ice (ALI)** model using the **Kokkos** programming model and **Trilinos** libraries.

1. Overview of the Albany/Land Ice (ALI) model/code developed under ProSPect SciDAC.
2. Performance portability of the finite element assembly in ALI using Kokkos.
3. Performant algebraic multi-grid linear solvers implemented in Trilinos.
4. Summary and discussion

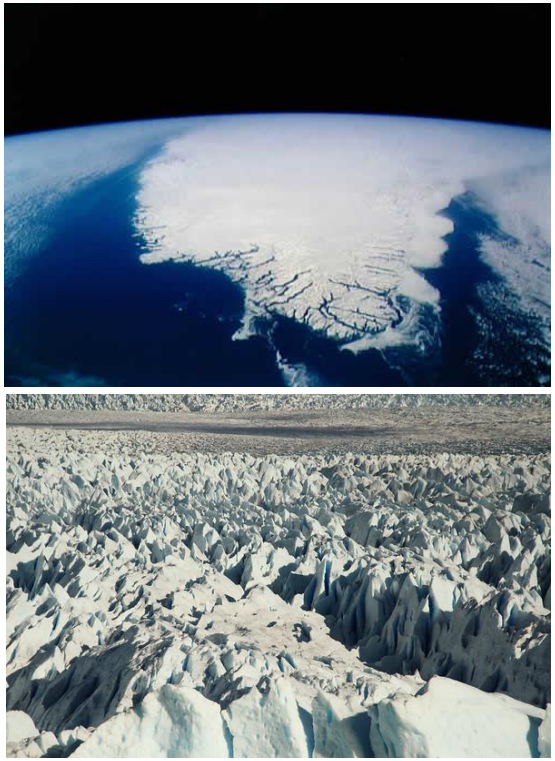


Outline

Sandia National Laboratories

This talk describes our **efforts** towards creating a **performance portable** implementation of the **Albany/Land Ice (ALI)** model using the **Kokkos** programming model and **Trilinos** libraries.

1. **Overview of the Albany/Land Ice (ALI) model/code developed under ProSPect SciDAC.**
2. Performance portability of the finite element assembly in ALI using Kokkos.
3. Performant algebraic multi-grid linear solvers implemented in Trilinos.
4. Summary and discussion



ProSPect project for land-ice modeling

Sandia National Laboratories

SciDAC Scientific Discovery through Advanced Computing

"ProSPect" = Probabilistic Sea Level Projections from ISMs and ESMs
5 year SciDAC4 project (2017-2022), https://doe-prospect.github.io/

E3SM Energy Exascale Earth System Model

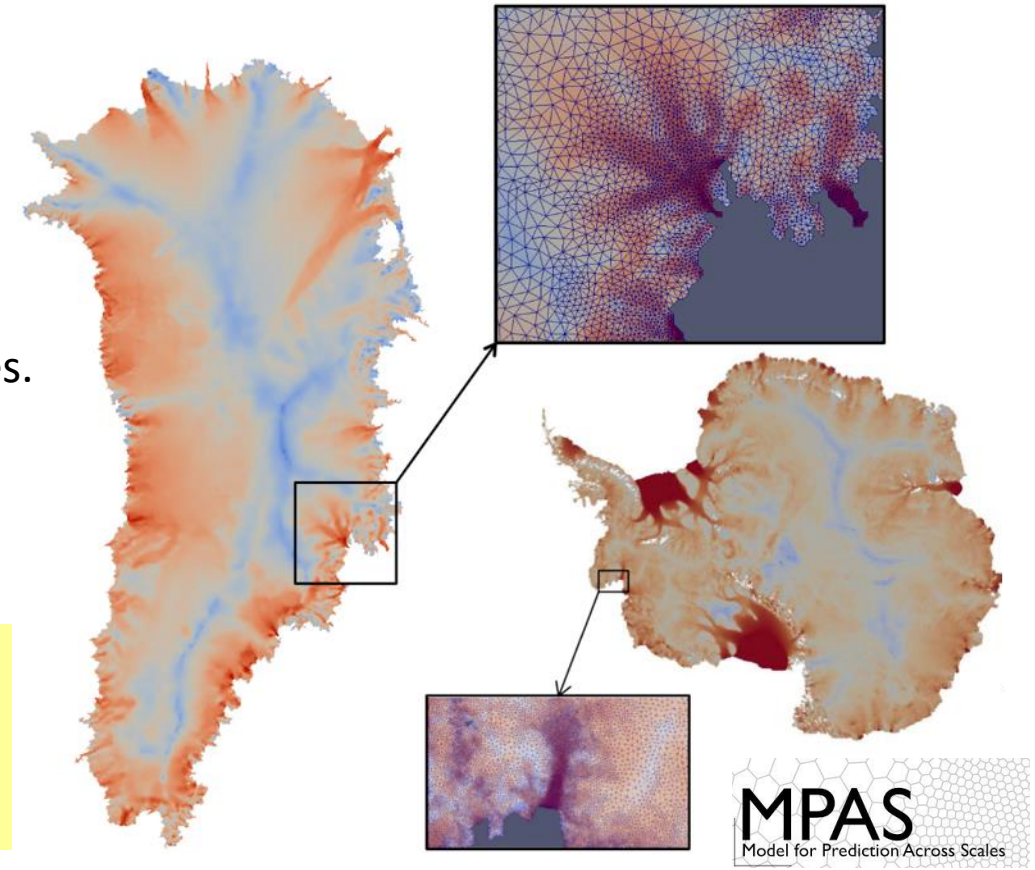
Sandia's Role in the ProSPect Project: to **develop** and **support** a robust and scalable land ice solver based on the "First-Order" (FO) Stokes equations → *Albany/Land Ice (ALI)**

Requirements for Albany/Land Ice (ALI)*:

- ***Unstructured grid*** meshes.
- ***Scalable, fast*** and ***robust.***
- ***Verified*** and ***validated.***
- ***Portable*** to new architecture machines.
- ***Advanced analysis*** capabilities: deterministic inversion, calibration, uncertainty quantification.

Hooked up to ***DOE's E3SM Earth System Model*** through ***MPAS***

(MPAS + ALI = MALI)



* Formerly Albany/FELIX.

ProSPect project for land-ice modeling

Sandia National Laboratories

SciDAC Scientific Discovery through Advanced Computing

"ProSPect" = Probabilistic Sea Level Projections from ISMs and ESMs
5 year SciDAC4 project (2017-2022), https://doe-prospect.github.io/

E3SM Energy Exascale Earth System Model

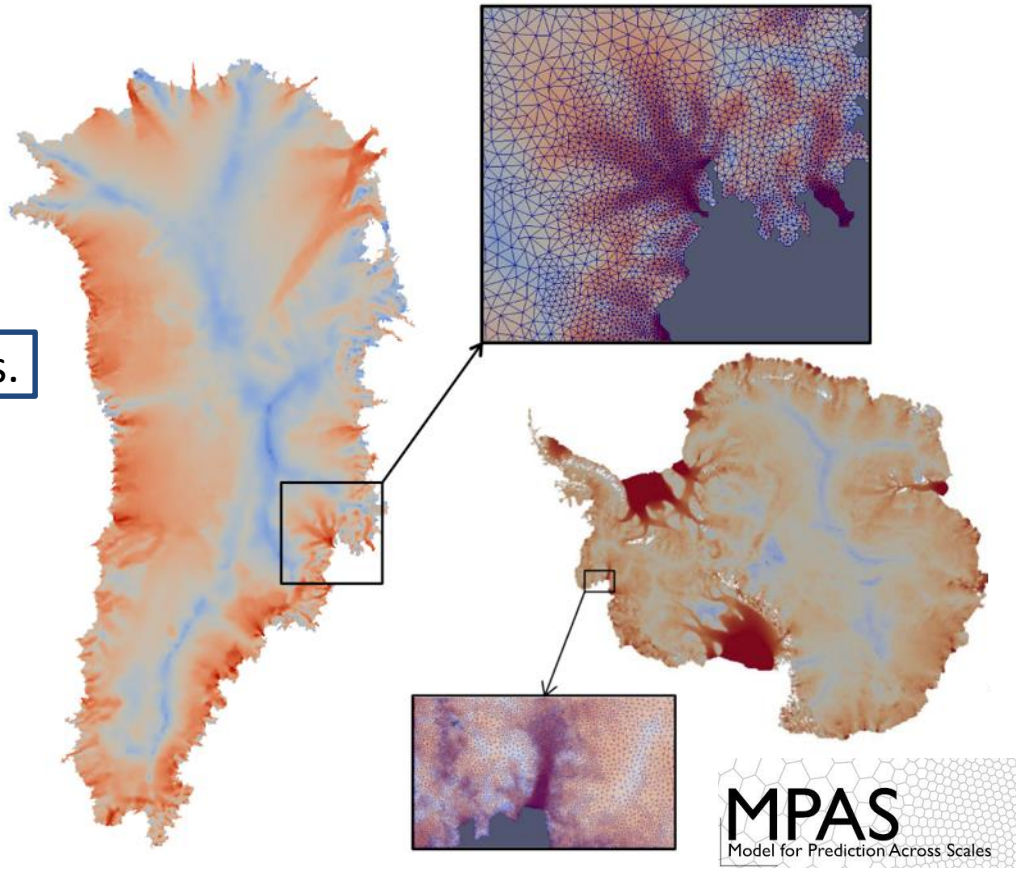
Sandia's Role in the ProSPect Project: to **develop** and **support** a robust and scalable land ice solver based on the "First-Order" (FO) Stokes equations → *Albany/Land Ice (ALI)**

Requirements for Albany/Land Ice (ALI)*:

- ***Unstructured grid*** meshes.
- ***Scalable, fast*** and ***robust.***
- ***Verified*** and ***validated.***
- ***Portable*** to new architecture machines.
- ***Advanced analysis*** capabilities: deterministic inversion, calibration, uncertainty quantification.

Hooked up to ***DOE's E3SM Earth System Model*** through ***MPAS***

(MPAS + ALI = MALI)



* Formerly Albany/FELIX.

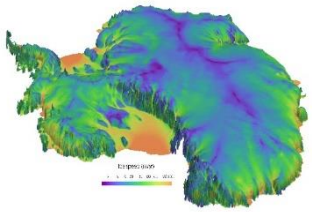
Albany finite element C++ code base

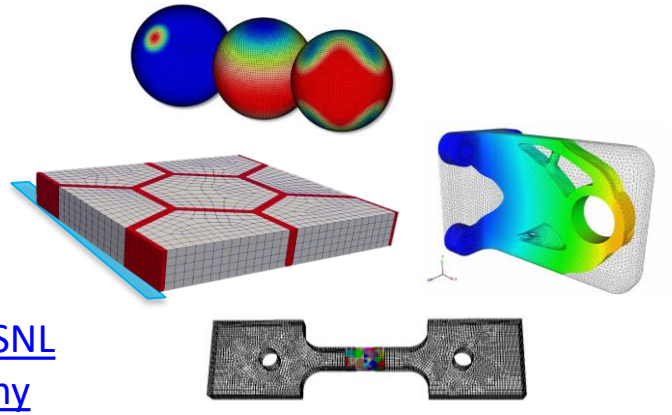
Sandia National Laboratories

The ***Albany/Land Ice*** First Order Stokes solver is implemented in a Sandia open-source parallel C++ multi-physics finite element code known as…

Albany

- Land Ice Equation Set (***ALI***)
- Other Equation Sets





Albany: https://github.com/SNLComputation/Albany

"Agile Components"

- Discretizations/meshes
- Solver libraries
- Preconditioners
- Automatic differentiation
- Performance portable kernels
- Many others!

Trilinos

- Parameter estimation
- Uncertainty quantification
- Optimization
- Bayesian inference

DAKOTA

By using ***software components***, we have been able to ***leverage*** years of ***R&D*** in ***algorithms***, ***software***, and ***performance portability***!

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

Dakota: https://dakota.sandia.gov/

First-Order (FO) Stokes model

- Ice velocities given by the ***"First-Order" Stokes PDEs*** with nonlinear viscosity:

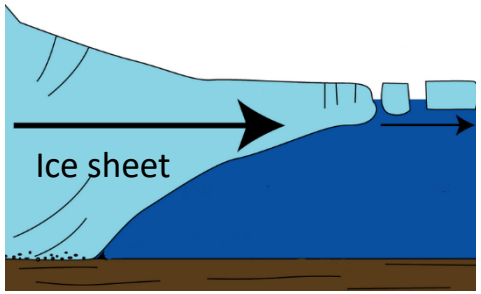
$$\begin{cases} -\nabla \cdot (2\mu\dot{\epsilon}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla \cdot (2\mu\dot{\epsilon}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}$$

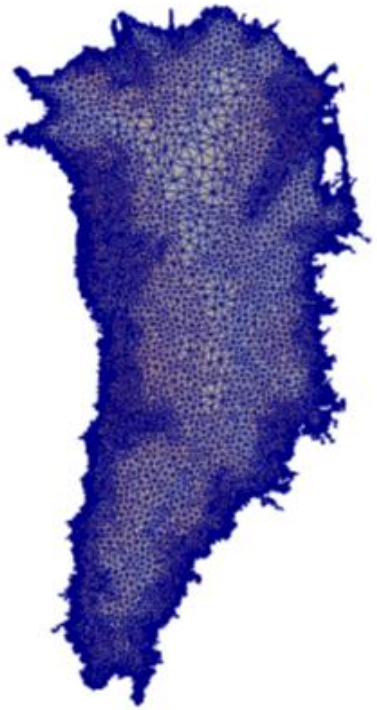
$$\mu = \frac{1}{2} A^{-\frac{1}{n}} \left(\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}{}^2 \right)^{\left(\frac{1}{2n} - \frac{1}{2}\right)}$$

$$\dot{\epsilon}_1{}^T = (2\dot{\epsilon}_{11} + \dot{\epsilon}_{22}, \dot{\epsilon}_{12}, \dot{\epsilon}_{13})$$
$$\dot{\epsilon}_2{}^T = (2\dot{\epsilon}_{12}, \dot{\epsilon}_{11} + 2\dot{\epsilon}_{22}, \dot{\epsilon}_{23})$$
$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

Algorithmic choices for ALI:

- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- Preconditioned ***Krylov iterative linear solvers***.
- ***Advanced analysis capabilities***: deterministic inversion, calibration, UQ.





First-Order (FO) Stokes model

- Ice velocities given by the ***"First-Order" Stokes PDEs*** with nonlinear viscosity:

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

$$\mu = \frac{1}{2} A^{-\frac{1}{n}} \left(\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}{}^2 \right)^{\left(\frac{1}{2n} - \frac{1}{2}\right)}$$

$$\dot{\epsilon}_1{}^T = (2\dot{\epsilon}_{11} + \dot{\epsilon}_{22}, \dot{\epsilon}_{12}, \dot{\epsilon}_{13})$$
$$\dot{\epsilon}_2{}^T = (2\dot{\epsilon}_{12}, \dot{\epsilon}_{11} + 2\dot{\epsilon}_{22}, \dot{\epsilon}_{23})$$
$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

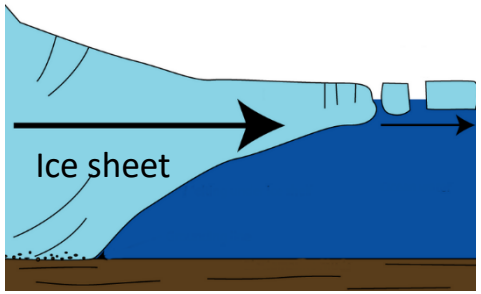
Algorithmic choices for ALI:

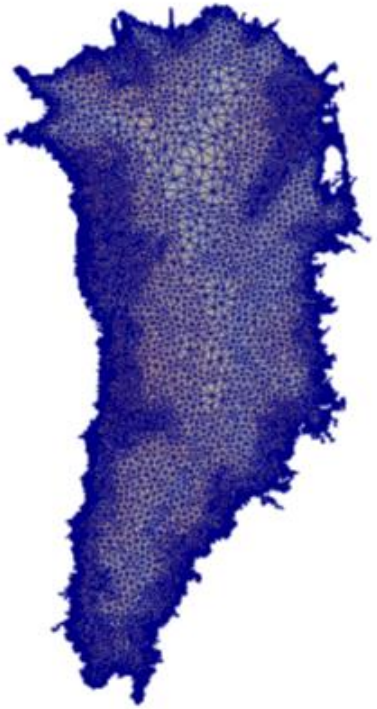
- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- Preconditioned ***Krylov iterative linear solvers***.
- ***Advanced analysis capabilities***: deterministic inversion, calibration, UQ.

Implicit solver:

FEA* = 50% CPU-time

Linear solve = 50% CPU-time





* Finite Element Assembly

First-Order (FO) Stokes model

- Ice velocities given by the ***"First-Order" Stokes PDEs*** with nonlinear viscosity:

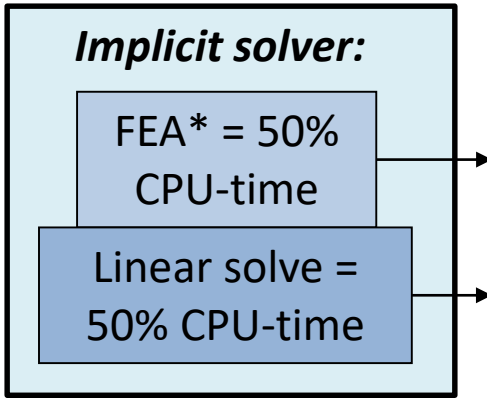
$$\begin{cases} -\nabla \cdot (2\mu\dot{\epsilon}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla \cdot (2\mu\dot{\epsilon}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}$$

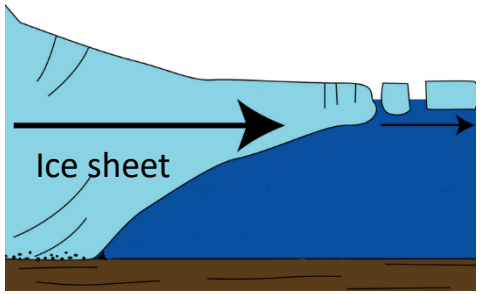
$$\mu = \frac{1}{2} A^{-\frac{1}{n}} \left(\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}{}^2 \right)^{\left(\frac{1}{2n} - \frac{1}{2}\right)}$$

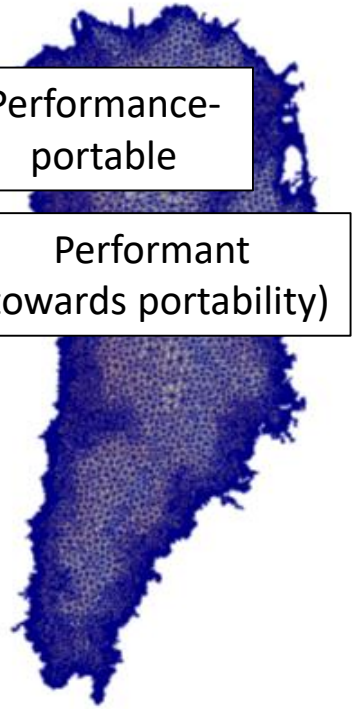
$$\dot{\epsilon}_1{}^T = (2\dot{\epsilon}_{11} + \dot{\epsilon}_{22}, \dot{\epsilon}_{12}, \dot{\epsilon}_{13})$$
$$\dot{\epsilon}_2{}^T = (2\dot{\epsilon}_{12}, \dot{\epsilon}_{11} + 2\dot{\epsilon}_{22}, \dot{\epsilon}_{23})$$
$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

Algorithmic choices for ALI:

- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- Preconditioned ***Krylov iterative linear solvers***.
- ***Advanced analysis capabilities***: deterministic inversion, calibration, UQ.







E3SM Energy Exascale Earth System Model

Albany

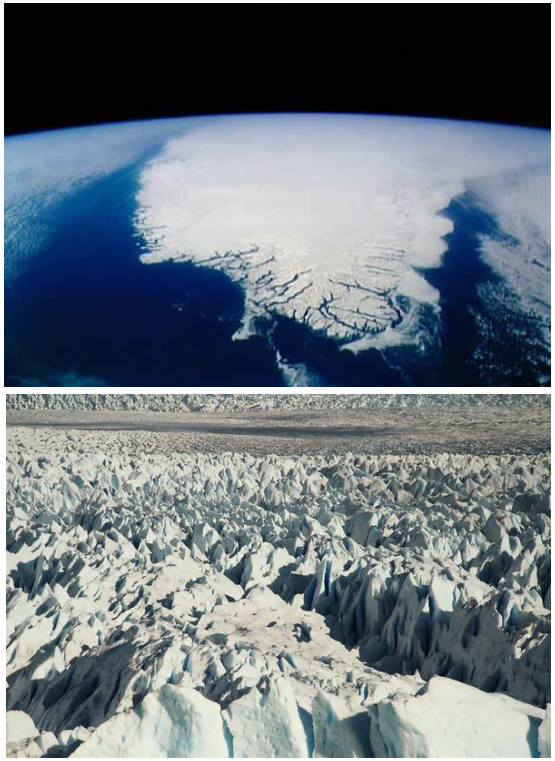
* Finite Element Assembly

Outline

Sandia National Laboratories

This talk describes our **experience** in creating a **performance portable** implementation of the **Albany/Land Ice (ALI)** model using the **Kokkos** programming model and **Trilinos** libraries.

1. Overview of the Albany/Land Ice (ALI) model/code developed under ProSPect SciDAC.
2. **Performance portability of the finite element assembly in ALI using Kokkos.**
3. Performant algebraic multi-grid linear solvers implemented in Trilinos.
4. Summary and discussion



Performance portability via *Kokkos*

Sandia National Laboratories

We need to be able to run ALI/E3SM on ***new architecture machines*** (GPUs, KNLs) and ***hybrid*** (CPU+GPU) ***systems***.

- **Kokkos*** is a C++ library that provides **performance portability** across multiple **shared memory** computing architectures using the MPI+X programming model
  - A *programming model* as much as a software library.
  - Provides automatic access to OpenMP, CUDA, Pthreads, ...
  - Templated meta-programming: parallel_for, parallel_reduce
  - Memory layout abstraction ("array of structs" vs. "struct of arrays", locality).

With ***Kokkos***, you write an algorithm once, and just change a template parameter to get the optimal data layout for your hardware.

- Allows researchers to focus on **application development** instead of **architecture specific programming.**
- **Finite element assembly** in ALI has been rewritten using Kokkos functors.

* https://github.com/kokkos/kokkos

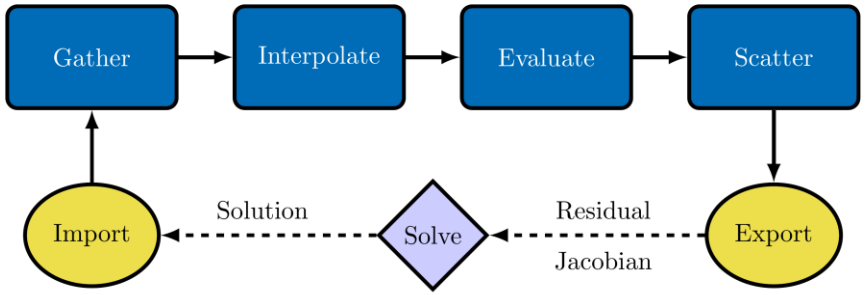
ALI Finite Element Assembly (FEA)

Albany Land Ice performance is split between the **linear solve** (50%) and **FEA** (50%)

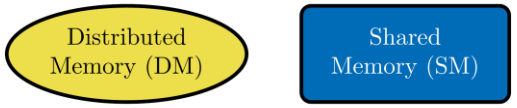
- **Piro** manages the nonlinear solve
- **Tpetra** manages **distributed** memory linear algebra (**MPI+X**)
- **Phalanx** manages **shared** memory computations (**X**)
  - **Gather** fills element local solution
  - **Interpolate** solution/gradient to quadrature points
  - **Evaluate** residual/Jacobian
  - **Scatter** fills global residual/Jacobian
- Jacobians (+ sensitivities, Hessians, ...) obtained via **automatic differentiation** (**Sacado**).

Trilinos Packages

FEA Overview



Memory Model



MPI+X FEA via Kokkos

Sandia National Laboratories

MPI-only FEA ⇨ MPI+X FEA

Shared Memory (SM)

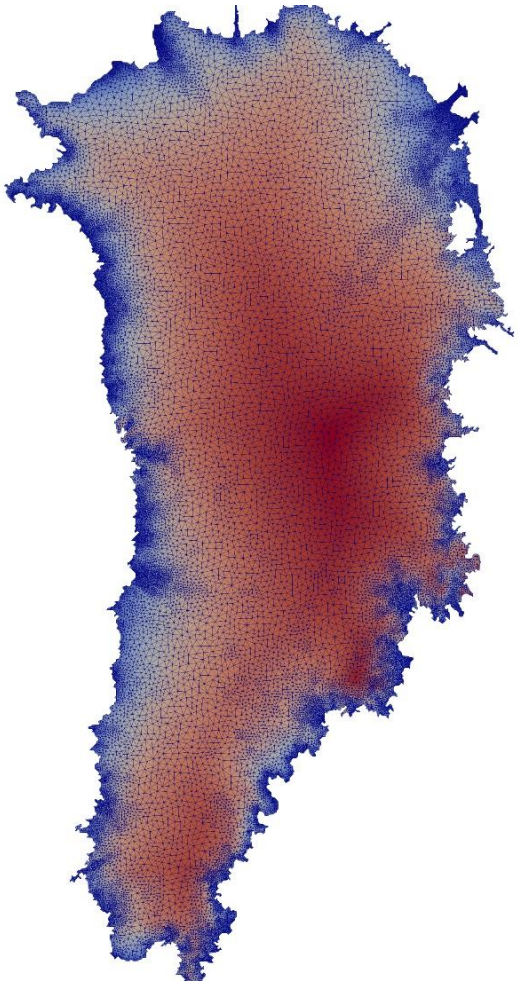
```
typedef Kokkos::OpenMP ExecutionSpace;
//typedef Kokkos::CUDA ExecutionSpace;
//typedef Kokkos::Serial ExecutionSpace;
template<typename ScalarT>
vectorGrad<ScalarT>::vectorGrad()
{
Kokkos::View<ScalarT****, ExecutionSpace> vecGrad("vecGrad", numCells, numQP, numVec, numDim);
}
***********************************************
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);
} } } } }
```

Albany

ExecutionSpace parameter tailors code for device (e.g., OpenMP, CUDA, etc.)

Performance study: Greenland Ice Sheet (GIS)

Mesh	Resolution	# Elements
GIS4k-20k	4km-20km	1.51 million
GIS1k-7k	1km-7km	14.4 million



- Unstructured **tetrahedral** element meshes
- **Wall-clock time** averaged over 100 global assembly evaluations (residual + Jacobian)
- Performance analysis focuses on **finite element assembly**
- **Notation** for performance results:

$$r(\mathrm{MPI} + j\mathrm{X}), \qquad \mathrm{X} \in \{\mathrm{OMP}, \mathrm{GPU}\}$$

r = # MPI ranks
j = # OpenMP threads or GPUs/rank
X = architecture for shared memory parallelism

Performance study: Architectures

Sandia National Laboratories

Performance-portability of FEA in Albany has been tested across ***multiple architectures***: Intel Sandy Bridge, Intel Skylake, IBM Power8/9, Keplar/Pascal/Volta GPUs, KNL Xeon Phi

Architectures:

- **Cori (NERSC):** 2,388 Haswell nodes [2 **Haswell** (32 cores)]
  9,688 KNL nodes [1 Xeon Phi **KNL** (68 cores)] (Cray Aries)
- **Blake (SNL):** 40 nodes [2 **Skylake** (48 cores)] (Intel OmniPath Gen-1)
- **Mayer (SNL):** 43 nodes [2 **ARM64 Cavium ThunderX2** (56 cores)] (Mx EDR IB)
- **Ride (SNL):** 12 nodes [2 POWER8 (16 cores) + **P100** (4 GPUs)] (Mx C-X4 IB)
- **Waterman (SNL):** 10 nodes [2 POWER9 (40 cores) + **V100** (4 GPUs)] (Mx EDR IB)

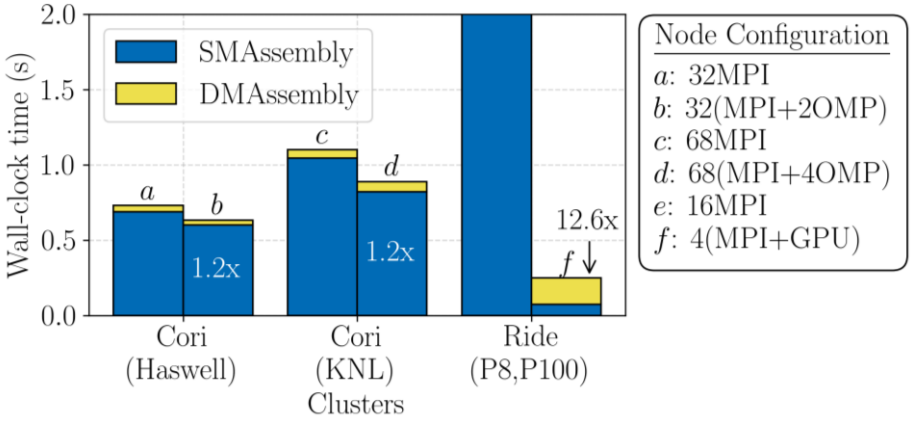
Compilers: gcc/icpc/xlC

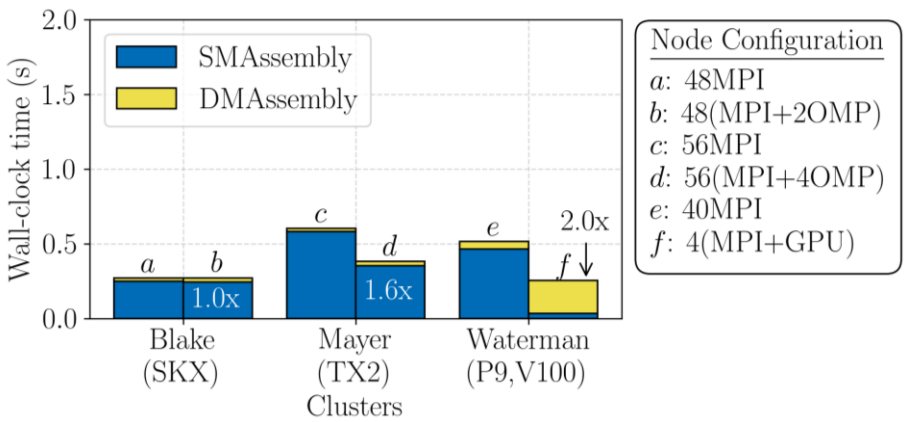
Models:

- **3 models:** MPI-only, MPI+OpenMP, MPI+CUDA
- **MPI+OpenMP:** **MPI ranks** are mapped to **cores**,
  OpenMP threads are mapped to **hardware-threads**
- **MPI+GPU:** MPI ranks assigned a **single core per GPU**
  - ➢ CUDA UVM used for host to device communication

Performance results: Node utilization

Node: Single dual-socket CPU or quad-GPU





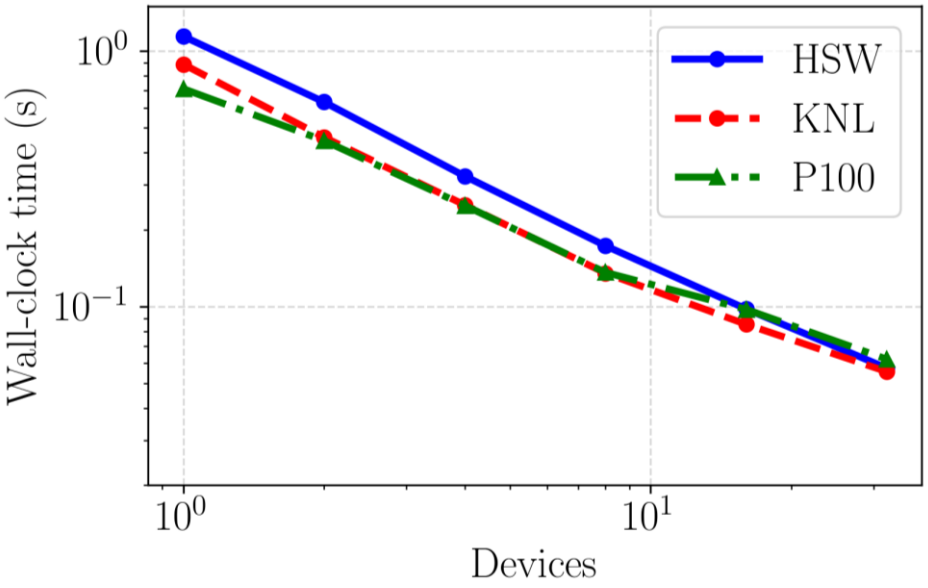
- Speedup achieved across **most** execution spaces
  - Kokkos Serial vs. OpenMP or CUDA
  - **12.6x** speedup on POWER8+P100, **2.0x** speedup on POWER9+V100 (~16x speedup would be expected if memory bound, but we are latency bound due to Export/Import).
  - In general, should expect no speedup with MPI+OpenMP – slight speedups on Mayer and Cori may be due to thread caching.
- Tpetra Export poor on GPU machines (WIP within Albany and GPUDirect issue on POWER systems: CUDA does not play well with MPI!)

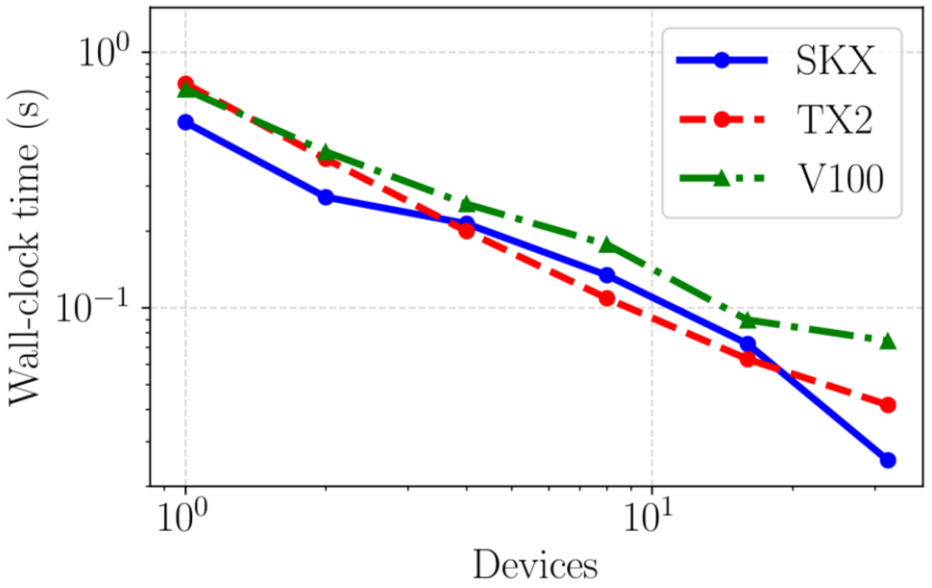
Blue (SMAssembly): shared memory local/global assembly (assembly/computation)
Yellow (DMAssembly): distributed memory global assembly handled by ***Tpetra*** (mostly communication)

Performance results: Strong scalability

Sandia National Laboratories

Legend: **HSW, SKX=Haswell, Skylake CPU; KNL=Xeon Phi; TX2=ThunderX2; P100,V100=GPU**





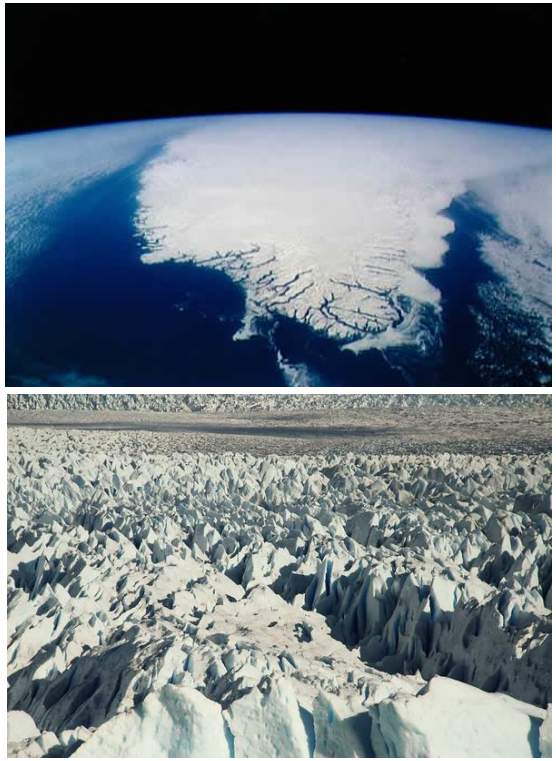
- Reasonable scaling across all devices **without** machine-specific optimization in Albany
  - ➢ Poor GPU scaling (Export WIP within Albany and GPUDirect issue on POWER)
  - ➢ Best case: Skylake at 32 devices (768 cores)

Outline

Sandia National Laboratories

This talk describes our **efforts** towards creating a **performance portable** implementation of the **Albany/Land Ice (ALI)** model using the **Kokkos** programming model and **Trilinos** libraries.

1. Overview of the Albany/Land Ice (ALI) model/code developed under ProSPect SciDAC.
2. Performance portability of the finite element assembly in ALI using Kokkos**.**
3. **Performant algebraic multi-grid linear solvers implemented in Trilinos.**
4. Summary and discussion



Motivation for linear solvers work

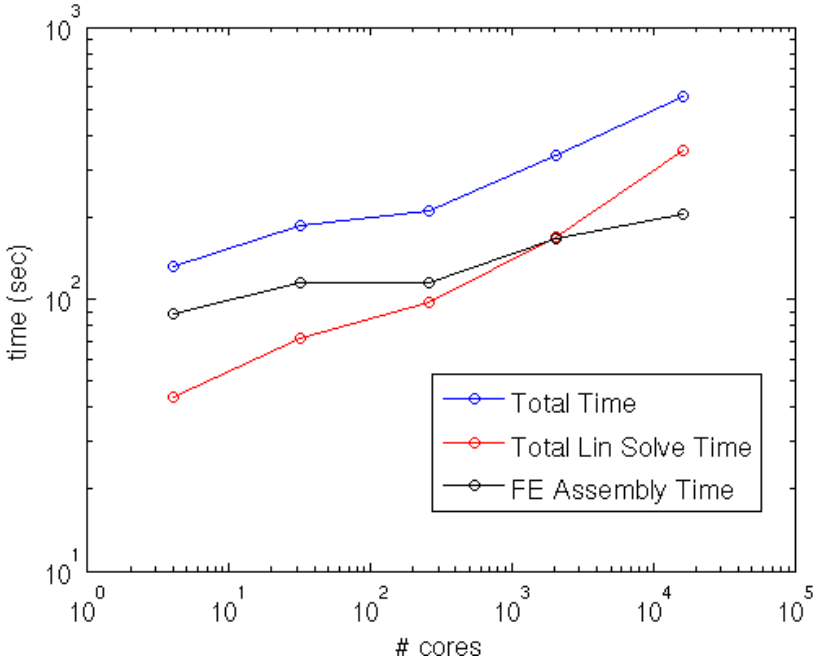
Sandia National Laboratories

- ***Linear solver*** takes ***~50%*** of total CPU time for ALI ***diagnostic solve***
  - FEA is only ***half the story***: we need to make linear solver performant (and ultimately performance portable)

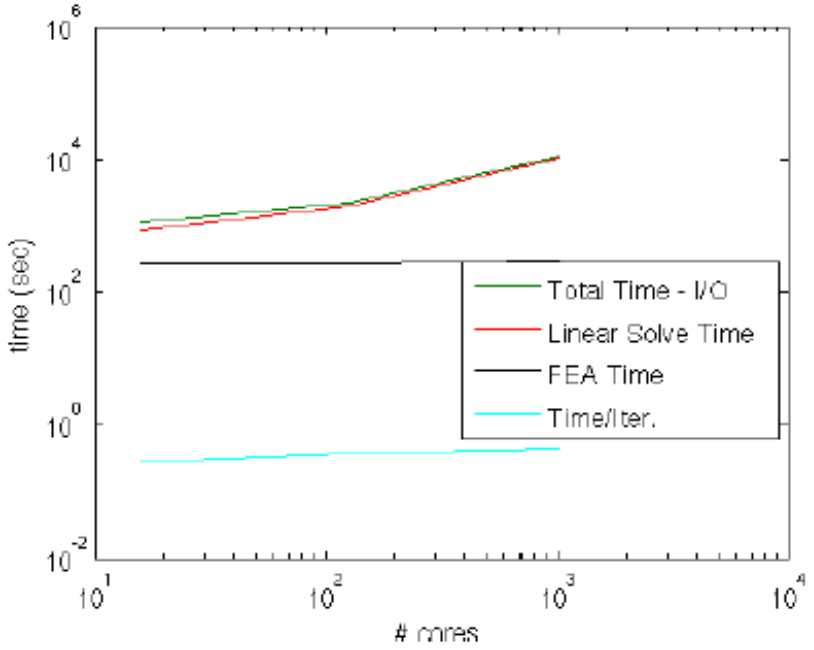
Motivation for linear solvers work

Sandia National Laboratories

Greenland Ice Sheet



Antarctic Ice Sheet



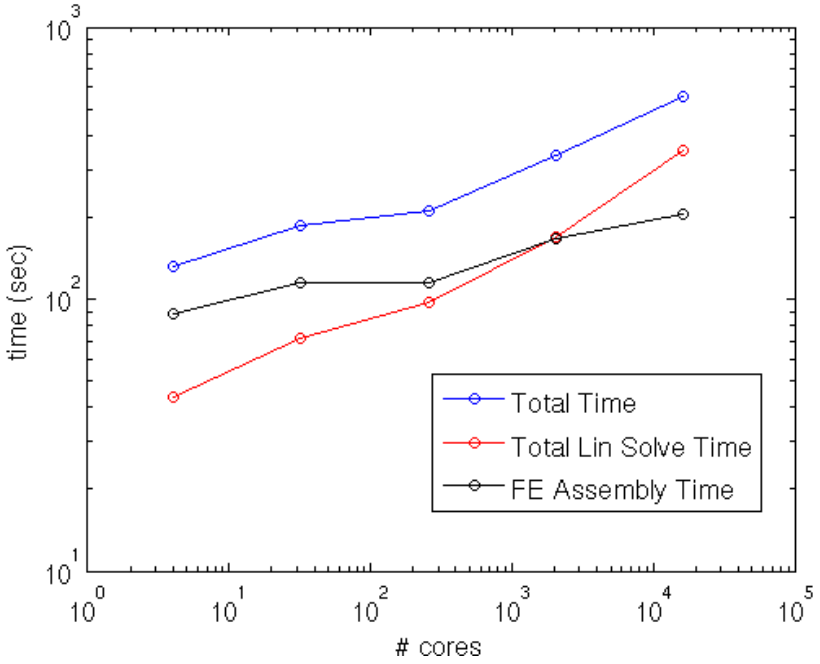
Off-the-shelf linear solvers (ILU, AMG*) ***do not*** always work that ***well!***

* Algebraic Multi-Grid.

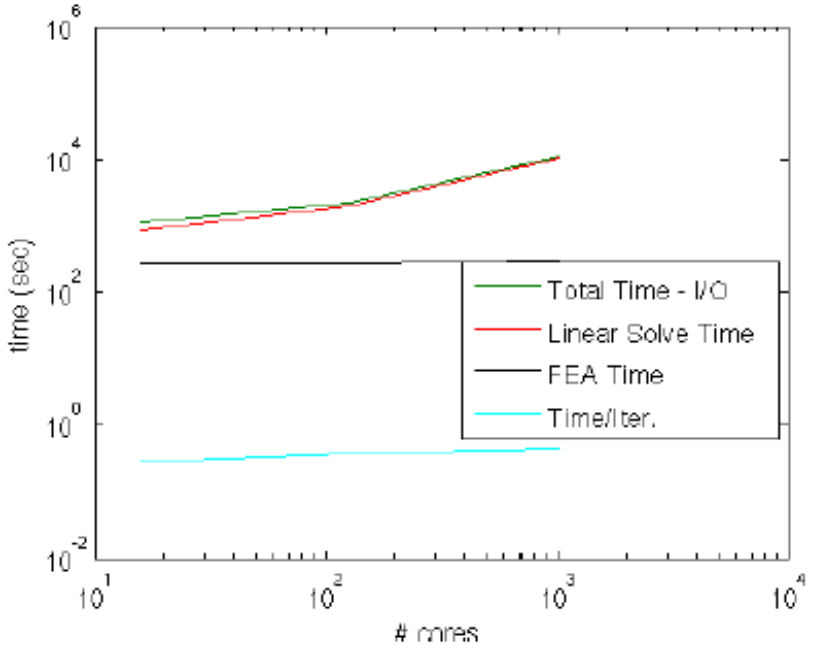
Motivation for linear solvers work

Sandia National Laboratories

Greenland Ice Sheet



Antarctic Ice Sheet



Off-the-shelf linear solvers (ILU, AMG*) ***do not*** always work that ***well!***

*Why is scalability so **bad** for off-the-shelf preconditioners?*

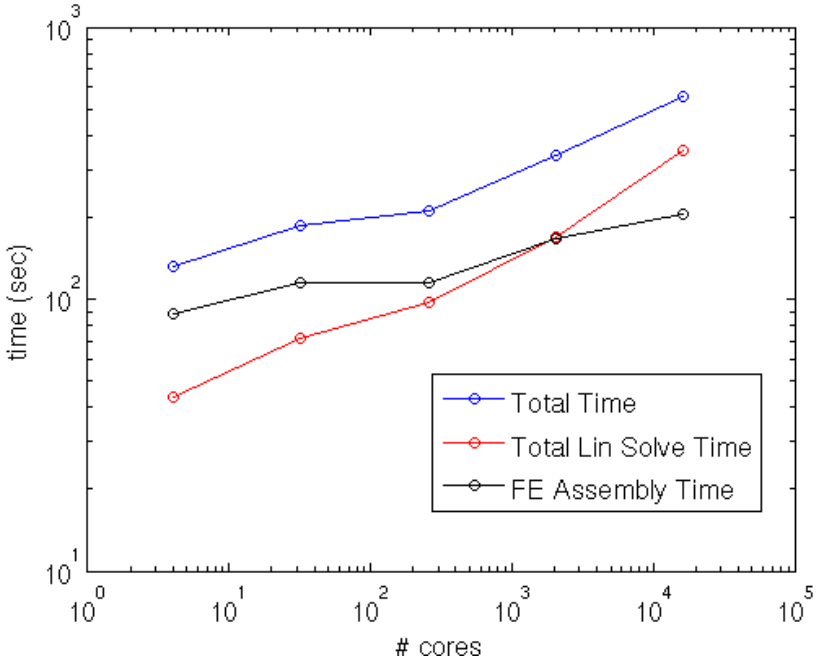
1. Ice sheet geometries have ***bad aspect ratios*** $(dx \gg dz)$.
2. ***Ice shelves*** can generate ***problematic linear systems***.
3. ***Islands*** and ***hinged peninsulas*** lead to ***solver failures***.

* Algebraic Multi-Grid.

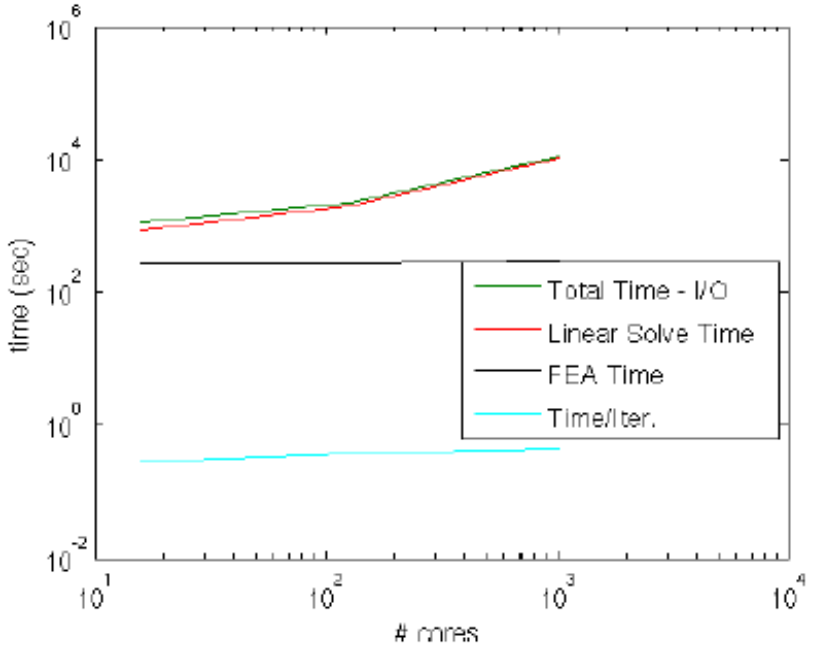
Motivation for linear solvers work

Sandia National Laboratories

Greenland Ice Sheet



Antarctic Ice Sheet



Off-the-shelf linear solvers (ILU, AMG*) ***do not*** always work that ***well!***

*Why is scalability so **bad** for off-the-shelf preconditioners?*

1. Ice sheet geometries have ***bad aspect ratios*** $(dx \gg dz)$.
2. ***Ice shelves*** can generate ***problematic linear systems***.
3. ***Islands*** and ***hinged peninsulas*** lead to ***solver failures***.

We **mitigate** these difficulties through the development of:

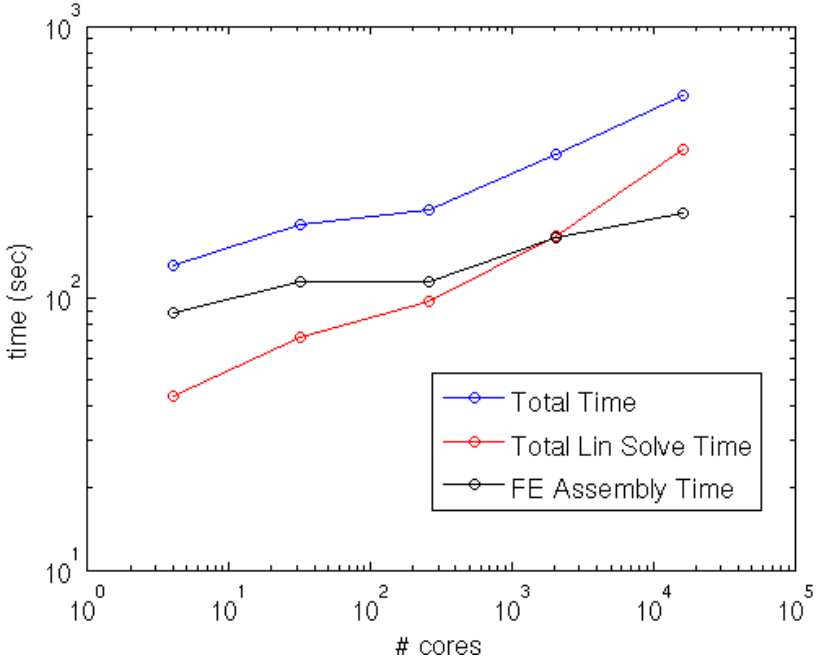
- *New AMG* preconditioner based on semi-coarsening.*
- *Island/hinge removal algorithm.*

* Algebraic Multi-Grid.

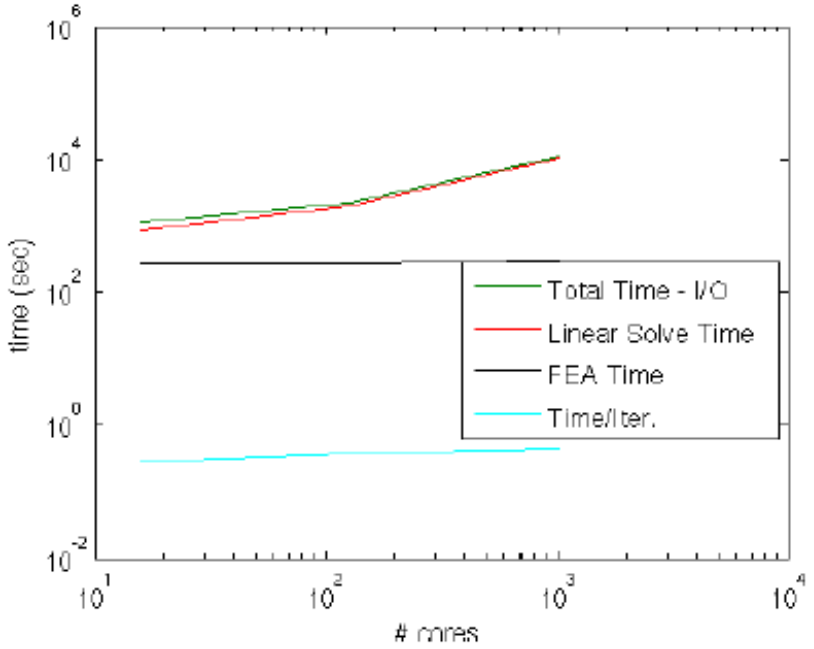
Motivation for linear solvers work

Sandia National Laboratories

Greenland Ice Sheet



Antarctic Ice Sheet



Off-the-shelf linear solvers (ILU, AMG*) ***do not*** always work that ***well!***

*Why is scalability so **bad** for off-the-shelf preconditioners?*

1. Ice sheet geometries have ***bad aspect ratios*** $(dx \gg dz)$.
2. ***Ice shelves*** can generate ***problematic linear systems***.
3. ***Islands*** and ***hinged peninsulas*** lead to ***solver failures***.

We **mitigate** these difficulties through the development of:

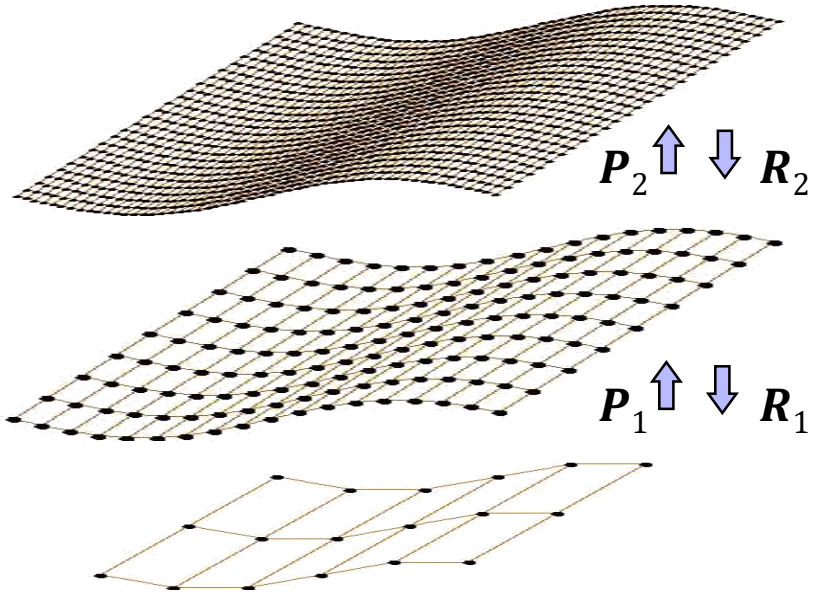
- ***New AMG* preconditioner based on semi-coarsening.***
- *Island/hinge removal algorithm*.

* Algebraic Multi-Grid.

How Does Multi-Grid Work?

Basic idea: accelerate convergence of an iterative method on a given grid by solving a series of (cheaper) problems on coarser grids.

- Create set of ***coarse approximations.***
- Apply ***restriction operator*** $\boldsymbol{R}_i$ to interpolate from fine to coarse grid.
- ***Solve*** problem on coarse grid.
- Apply ***prolongation operator*** $\boldsymbol{P}_i$ to get back to original (fine) grid.
- ***Smoothers*** are applied throughout procedure to reduce short wavelength errors.

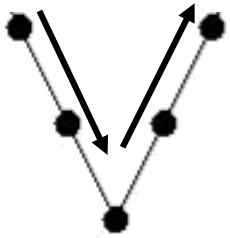


Solve $\boldsymbol{A}_3\boldsymbol{u}_3 = \boldsymbol{f}_3$

Smooth $\boldsymbol{A}_3\boldsymbol{u}_3 = \boldsymbol{f}_3$. Set $\boldsymbol{f}_2 = \boldsymbol{R}_2\boldsymbol{r}_3$.

Smooth $\boldsymbol{A}_2\boldsymbol{u}_2 = f_2$. Set $\boldsymbol{f}_1 = \boldsymbol{R}_1\boldsymbol{r}_2$.

Solve $\boldsymbol{A}_1\boldsymbol{u}_1 = \boldsymbol{f}_1$ directly.



Set $\boldsymbol{u}_3 = \boldsymbol{u}_3 + \boldsymbol{P}_2\boldsymbol{u}_2$. Smooth $\boldsymbol{A}_3\boldsymbol{u}_3 = \boldsymbol{f}_3$.

Set $\boldsymbol{u}_2 = \boldsymbol{u}_2 + \boldsymbol{P}_1\boldsymbol{u}_1$. Smooth $\boldsymbol{A}_2\boldsymbol{u}_2 = \boldsymbol{f}_2$.

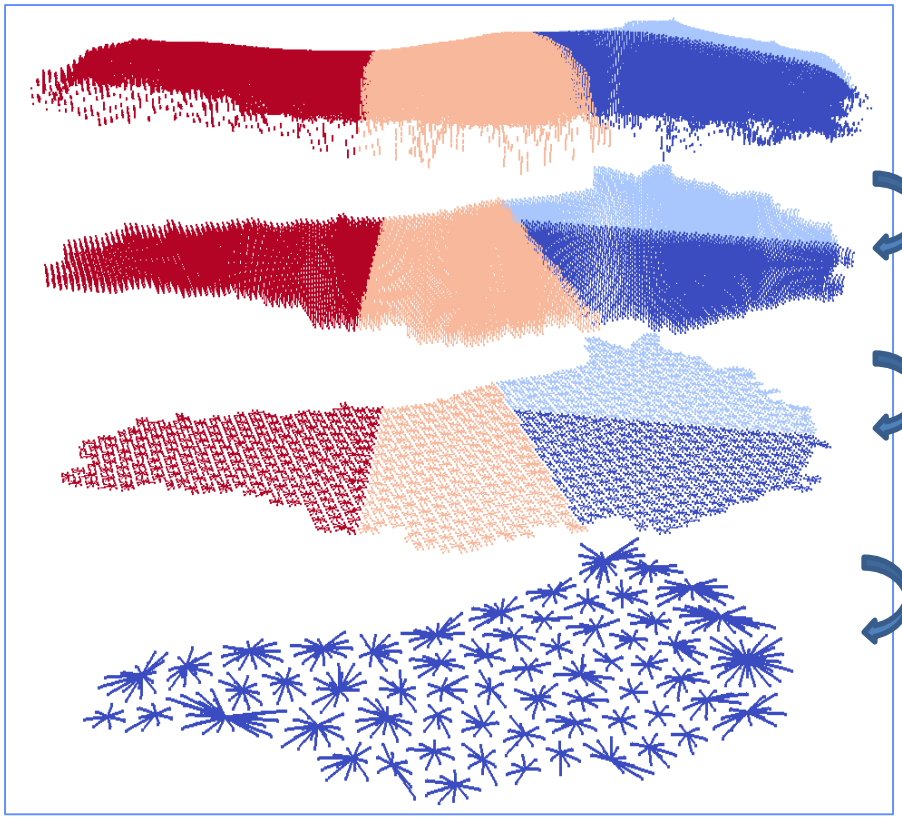
Scalable Algebraic Multi-Grid (AMG) Preconditioners

Sandia National Laboratories

Bad aspect ratios $(dx \gg dz)$ ruin classical AMG convergence rates!

- relatively small horizontal coupling terms, hard to smooth horizontal errors

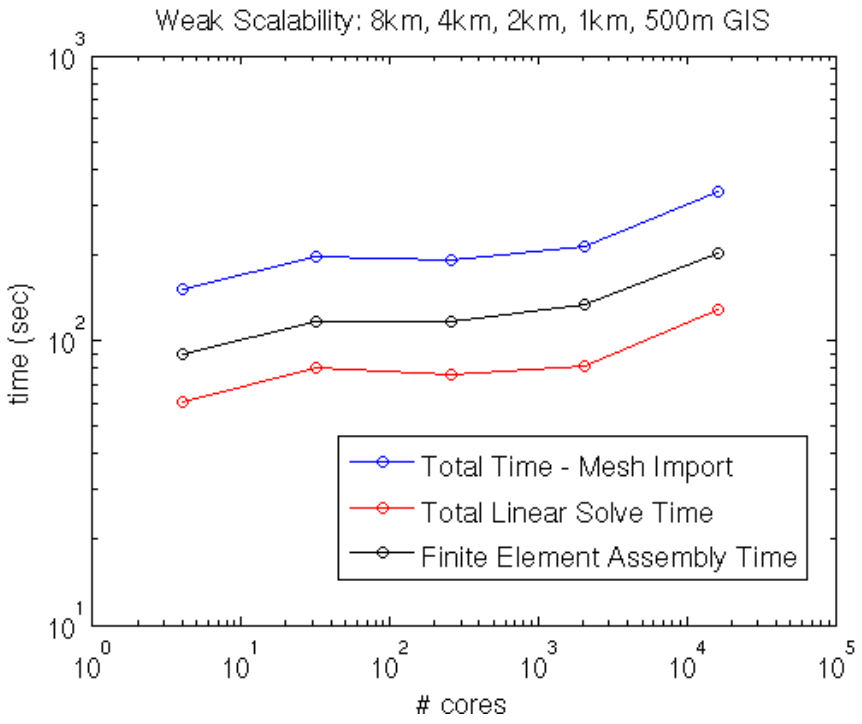
$\Rightarrow$ Solvers (AMG and ILU) must take **aspect ratios** into account!



We developed a **new AMG solver** based on aggressive **semi-coarsening** (available in *ML/MueLu* packages of *Trilinos*)

See (Tezaur *et al., Procedia CS,* 2015), (Tuminaro *et al., SISC,* 2016).

Weak scalability: Greenland



4 cores
334K dofs
8 km Greenland,
5 vertical layers

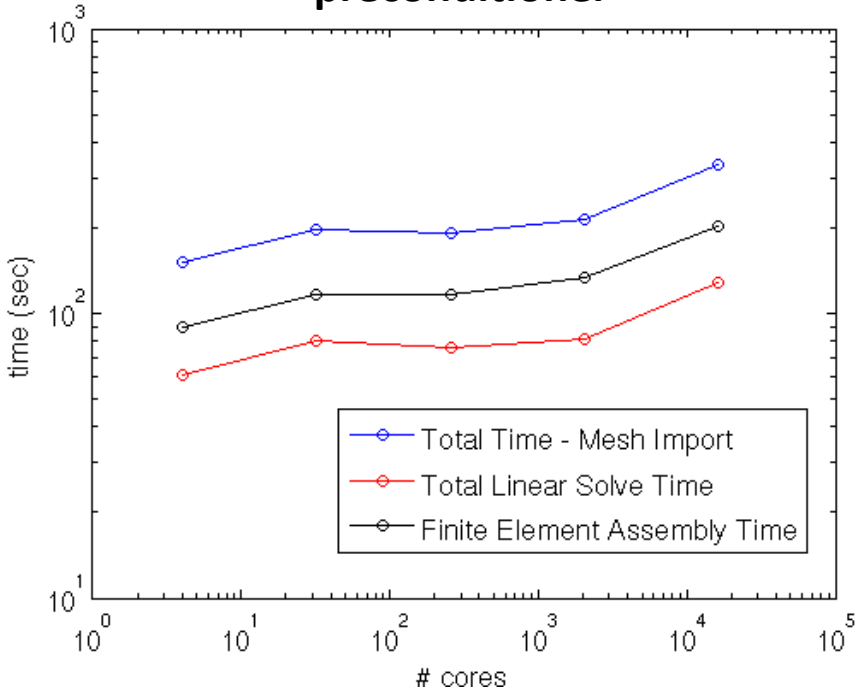
$\times 8^4$
scale up

16,384 cores
1.12B dofs(!)
0.5 km Greenland,
80 vertical layers

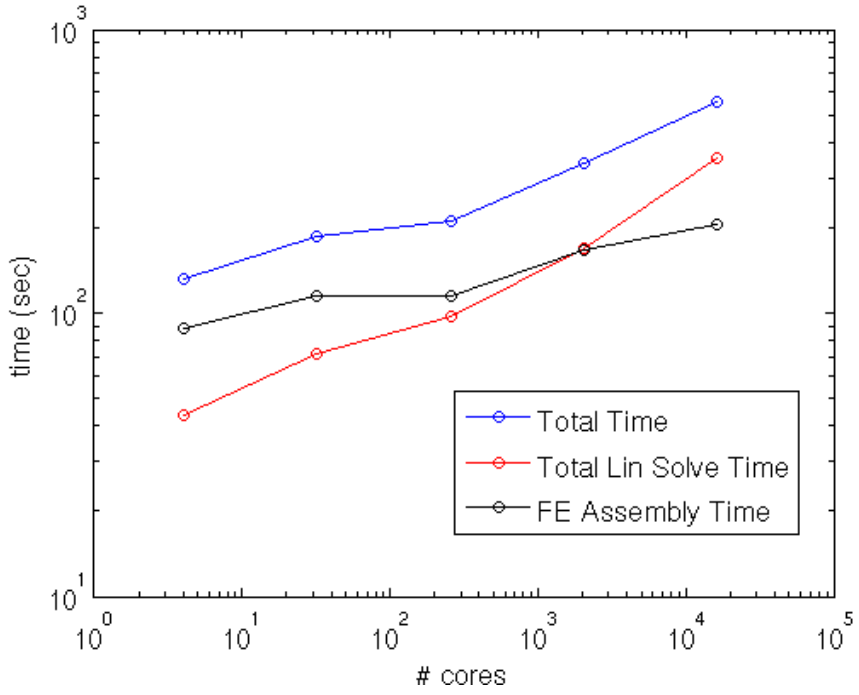
- Weak scaling study with fixed dataset, 4 mesh bisections.
- ~70-80K dofs/core.
- ***Conjugate Gradient (CG) iterative method*** for linear solves (faster convergence than GMRES).
- ***New AMG preconditioner*** developed by R. Tuminaro based on ***semi-coarsening*** (coarsening in z-direction only).
- ***Significant improvement*** in scalability with new AMG preconditioner over ILU preconditioner!

Weak scalability: Greenland

New AMG preconditioner preconditioner



ILU preconditioner



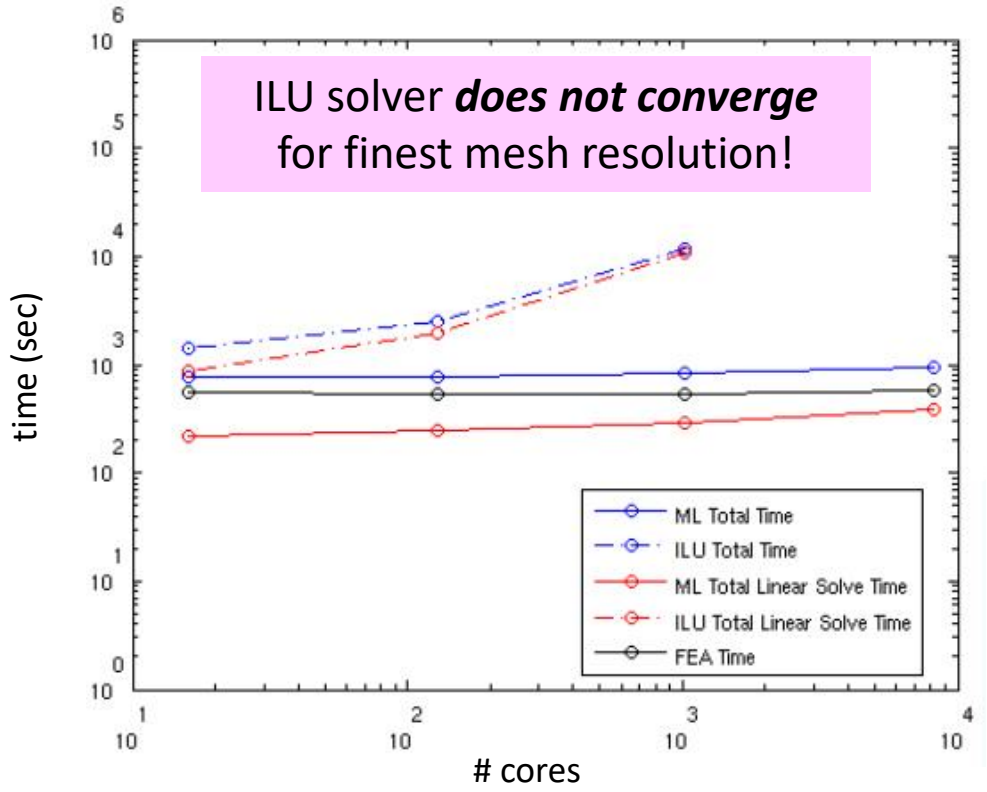
4 cores
334K dofs
8 km Greenland,
5 vertical layers

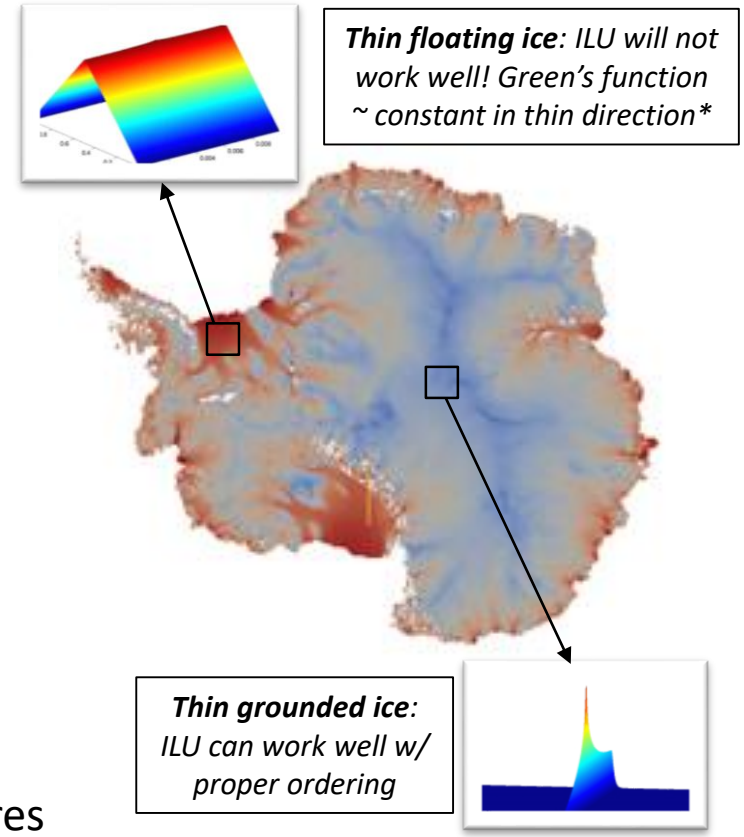
$\times 8^4$
scale up

16,384 cores
1.12B dofs(!)
0.5 km Greenland,
80 vertical layers

- ***Significant improvement*** in scalability with new AMG preconditioner over ILU preconditioner!

Weak scalability: Antarctica





***Thin floating ice**: ILU will not work well! Green's function ~ constant in thin direction**

***Thin grounded ice**: ILU can work well w/ proper ordering*

- **Weak scaling study**: 2.5M → 1.1B dofs, 16 → 8192 cores
- Initialized with realistic basal friction (from deterministic inversion) and temperature field from BEDMAP2.
- **Iterative linear solver**: GMRES.
- **Preconditioner**: ILU vs. new AMG based on aggressive semi-coarsening.

See (Tuminaro *et al., SISC,* 2016).

* A^{-1} will have large number of non-zeroes, so approximate inverse ILU preconditioner is ineffective.

Towards linear solver performance portability

- ***Trilinos templated software stack*** for sparse algebra interfaces/linear solvers (Tpetra, Belos, MueLu, Ifpack2) integrates Kokkos for ***performance portability.***

```
MueLu_CoarseMapFactory.hpp → MueLu_CoordinatesTransferFactory_kokkos.hpp
MueLu_CoordinatesTransferFactory.hpp → MueLu_CoarseMapFactory_kokkos.hpp
     MueLu_NullspaceFactory.hpp → MueLu_NullspaceFactory_kokkos.hpp
                                  …
```

- Semi-coarsening algorithm ***need not be redesigned*** for GPUs.
- Performance portability of MueLu solvers on advanced architectures including GPUs has been demonstrated for ***Maxwell*** and ***compressible flow*** equations.
  - ☑ ***Mat/vecs, orthogonalizations*** in Belos done on GPU.
  - ☑ ***Smoothers*** in Ifpack2 created/applied on GPU.
  - ☑ ***Coarse grid solve*** performed on host (direct solvers on GPUs is R&D topic).

Evaluating the ***performance portability*** of our ***AMG semi-coarsening-based*** solver in ALI is in the project plan for FY20 – all necessary ***pieces*** are in Trilinos!

Towards linear solver performance portability

- ***Trilinos templated software stack*** for sparse algebra interfaces/linear solvers (Tpetra, Belos, MueLu, Ifpack2) integrates Kokkos for ***performance portability.***

```
MueLu_CoarseMapFactory.hpp → MueLu_CoordinatesTransferFactory_kokkos.hpp
MueLu_CoordinatesTransferFactory.hpp → MueLu_CoarseMapFactory_kokkos.hpp
     MueLu_NullspaceFactory.hpp → MueLu_NullspaceFactory_kokkos.hpp
                                  …
```

- Semi-coarsening algorithm ***need not be redesigned*** for GPUs.
- Performance portability of MueLu solvers on advanced architectures including GPUs has been demonstrated for ***Maxwell*** and ***compressible flow*** equations.
  - ☑ ***Mat/vecs, orthogonalizations*** in Belos done on GPU.
  - ☑ ***Smoothers*** in Ifpack2 created/applied on GPU.
  - ☑ ***Coarse grid solve*** performed on host (direct solvers on GPUs is R&D topic).

Evaluating the ***performance portability*** of our ***AMG semi-coarsening-based*** solver in ALI is in the project plan for FY20 – all necessary ***pieces*** are in Trilinos!

*We will be looking to hire a **summer intern** in SNL/CA to help with this task!* ***Posting coming soon!***

Advertisement: Climate MS at ESCO 2020

European Seminar on COmputing (ESCO) 2020
June 8-12, 2020 (abstracts due Feb. 14, 2020)
Pilsen, Czech Republic
https://www.esco2020.femhub.com/

Computational Methodologies for Next-Generation Climate Models

Mauro Perego (mperego@sandia.gov, Sandia National Laboratories, USA), Irina Tezaur (ikalash@sandia.gov, Sandia National Laboratories, USA)

The development and application of global climate models for understanding and predicting the effects of global climate change and sea-level rise is critical, since it can direct energy and infrastructure planning, as well as inform public policy. Earth System Models (ESMs), which are global climate models including biogeochemistry, integrate the interactions between atmosphere, ocean, land, ice, and biosphere to enable the simulation of the state of regional and global climate under a wide variety of conditions. In recent years, there has been a push to develop "next generation" ESMs, models which: (1) are able to perform realistic, high-resolution, continental scale simulations, (2) are robust, efficient and scalable on next-generation hybrid systems (multi-core, many-core, GPU) towards achieving exascale performance, and (3) possess built-in advanced analysis capabilities (e.g., sensitivity analysis, optimization, uncertainty quantification).

This minisymposium will consist of talks describing new and ongoing research in the development of accurate and tractable "next-generation" models for stand-alone climate components (e.g., atmosphere, land-ice, sea-ice, ocean, land, biogeochemistry), as well talks addressing the challenges in coupling climate components for integration into ESMs. Of particular interest are:

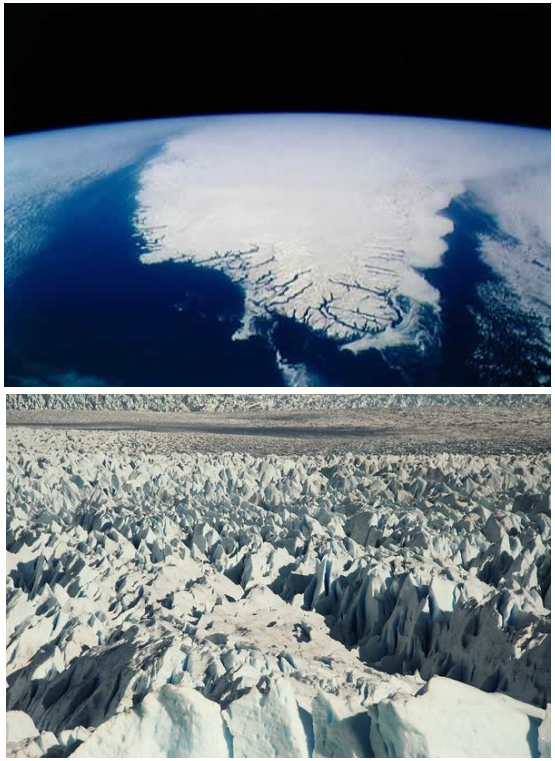
1. efficient computational strategies and software for tackling the complex, nonlinear, multi-scale, multi-physics problems arising in climate modeling, with an eye towards next-generation hybrid platforms, and
2. advanced analysis techniques that can inform/enhance existing models through the incorporation of observational data, e.g., approaches for model initialization/calibration, uncertainty quantification and data assimilation.

Outline

Sandia National Laboratories

This talk describes our **efforts** towards creating a **performance portable** implementation of the **Albany/Land Ice (ALI)** model using the **Kokkos** programming model and **Trilinos** libraries.

1. Overview of the Albany/Land Ice (ALI) model/code developed under ProSPect SciDAC.
2. Performance portability of the finite element assembly in ALI using Kokkos**.**
3. Performant algebraic multi-grid linear solvers implemented in Trilinos.
4. **Summary and discussion**



Summary and discussion points

Sandia National Laboratories

We are making **progress** towards running *Albany/Land Ice* on **heterogeneous HPC architectures** with the help of **Kokkos** and **Trilinos!**

Comments/discussion points:

- Kokkos (and similar libraries) ***not a magic bullet***!
  - Some algorithms need to be redesigned substantially to be efficient on GPUs/hybrid architectures (e.g. ILU), and Kokkos will not circumvent this fact.
- How ***feasible*** is it to port non-C++/Sandia codes to Kokkos?
  - E3SM seems to support C++/Kokkos route: BER-funded SCREAM project is aimed at rewriting (Fortran) HOMME atmospheric dycore using C++/Kokkos.
- There is always some ***tradeoff*** between portability and performance.
  - Getting the best possible performance on GPUs using Kokkos may require some platform-specific optimizations.
- Relying on libraries can be a ***blessing and a curse***.
  - Code can speed up *and* slow down with no changes on your side!
- ***Regression/performance testing*** is critical when targeting multiple architectures!
- ***Other solvers*** besides MG for GPUs worth considering (e.g. hierarchical solvers).

Funding/Acknowledgements

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

ProSPect team members: *K. Evans, M. Hoffman, M. Perego, S. Price, A. Salinger, I. Tezaur, R. Tuminaro, C. Sockwell, J. Watkins, L. Bertagna, T. Zhang.*

Trilinos/DAKOTA collaborators*: M. Eldred, J. Jakeman, G. Stadler.*

Computing resources: *NERSC, OLCF.*

References

[1] M.A. Heroux *et al.* "An overview of the Trilinos project." *ACM Trans. Math. Softw.* **31**(3) (2005).

[2] A. Salinger, *et al.* "Albany: Using Agile Components to Develop a Flexible, Generic Multiphysics Analysis Code", *Int. J. Multiscale Comput. Engng*. 14(4) (2016) 415-438.

[3] **I. Tezaur**, M. Perego, A. Salinger, R. Tuminaro, S. Price. "*Albany/FELIX*: A Parallel, Scalable and Robust Finite Element Higher-Order Stokes Ice Sheet Solver Built for Advanced Analysis", *Geosci. Model Develop.* 8 (2015) 1-24.

[4] C. Edwards, C. Trott, D. Sunderland. "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", *J. Par. & Distr. Comput.* **74** (12) (2014) 3202-3216.

[5] R. Tuminaro, M. Perego, **I. Tezaur**, A. Salinger, S. Price. "A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling", *SIAM J. Sci. Comput.* 38(5) (2016) C504-C532.

[6] **I. Tezaur**, R. Tuminaro, M. Perego, A. Salinger, S. Price. "On the scalability of the *Albany/FELIX* first-order Stokes approximation ice sheet solver for large-scale simulations of the Greenland and Antarctic ice sheets", *Procedia Computer Science,* 51 (2015) 2026-2035.

[7] I. Demeshko, J. Watkins, **I. Tezaur**, O. Guba, W. Spotz, A. Salinger, R. Pawlowski, M. Heroux. "Towards performance-portability of the Albany finite element analysis code using the Kokkos library", E. van Brummelen, A. Corsini, S. Perotto, G. Rozza, eds. *Numerical Methods for Flows: FEF 2017 Selected Contributions*, Elsevier, 2019.

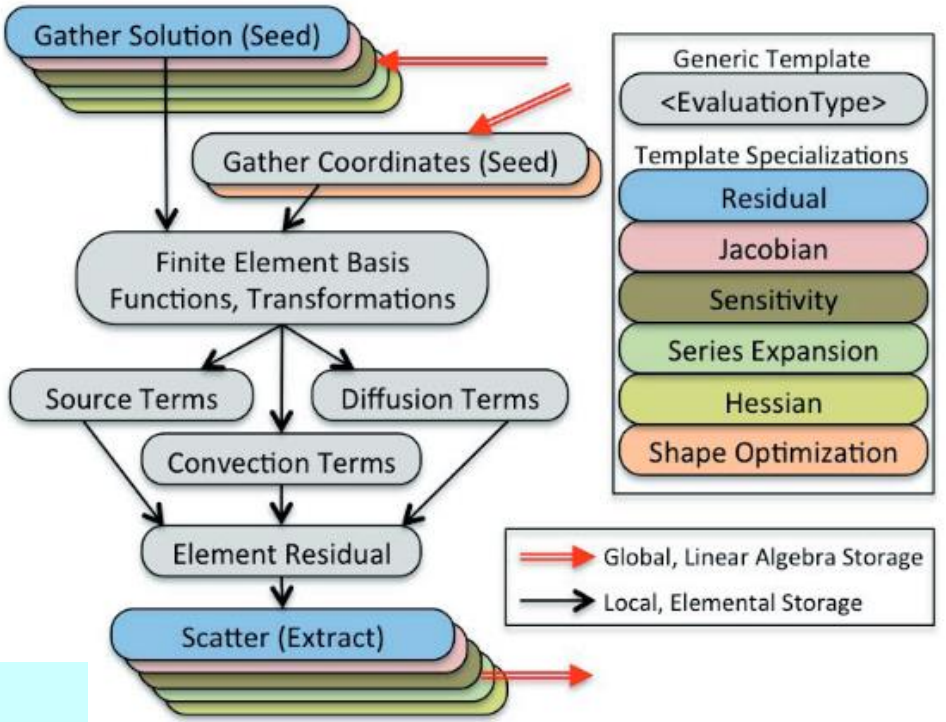
[8] S. Price, M. Hoffman, J. Bonin, T. Neumann, I. Howat, J. Guerber, **I. Tezaur**, J. Saba, J. Lanaerts, D. Chambers, W. Lipscomb, M. Perego, A. Salinger, R. Tuminaro. "An ice sheet model validation framework for the Greenland ice sheet", *Geosci. Model Dev.* 10 (2017) 255-270

Sandia National Laboratories

Start of Backup Slides

Albany/Land Ice Finite Element Assembly (FEA)

- ***Gather operation*** extracts solution values out of global solution vector.
- Physics ***evaluator*** functions operate on ***workset*** of elements, store evaluated quantities in local field arrays.
- FEA relies on ***template based generic programming*** + ***automatic differentiation*** for Jacobians, tangents, etc.
- ***Scatter operation*** adds local residual, Jacobian to global residual, Jacobian.



Performance-portability: focus on *FEA*.

- ***MPI-only FEA:***
  - Each MPI process has workset of cells & computes nested parallel for loops.
- ***MPI+X FEA:***
  - Each MPI process has workset of cells.
  - Multi-dimensional parallelism with +X (X=OpenMP, CUDA) for nested parallel for loops.

Problem Type	% CPU time for FEA
Implicit	50%
Explicit	99%

MPI+X FEA via *Kokkos*

Sandia National Laboratories

- ***MPI-only*** nested for loop:

```
for (int cell=0; cell<numCells; ++cell)
   for (int node=0; node<numNodes; ++node)
      for (int qp=0; qp<numQPs; ++qp)
         compute A;
```

MPI process *n*

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
   for (int node=0; node<numNodes; ++node)
      for (int qp=0; qp<numQPs; ++qp)
         compute A;
```

MPI process *n*

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
  for (int node=0; node<numNodes; ++node)
    for (int qp=0; qp<numQPs; ++qp)
      compute A;
```

MPI process *n*

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

* Unified Virtual Memory.

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
  for (int node=0; node<numNodes; ++node)
    for (int qp=0; qp<numQPs; ++qp)
      compute A;
```

MPI process *n*

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

- ExecutionSpace defined at ***compile time***, e.g.

  typedef Kokkos::OpenMP ExecutionSpace; //MPI+OpenMP

  typedef Kokkos::CUDA ExecutionSpace; //MPI+CUDA

  typedef Kokkos::Serial ExecutionSpace; //MPI-only

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
  for (int node=0; node<numNodes; ++node)
    for (int qp=0; qp<numQPs; ++qp)
      compute A;
```

MPI process *n*

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

- ExecutionSpace defined at ***compile time***, e.g.

  typedef Kokkos::OpenMP ExecutionSpace; //MPI+OpenMP

  typedef Kokkos::CUDA ExecutionSpace; //MPI+CUDA

  typedef Kokkos::Serial ExecutionSpace; //MPI-only

- ***Atomics*** used to scatter local data to global data structures

  Kokkos::atomic_fetch_add

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
  for (int node=0; node<numNodes; ++node)
    for (int qp=0; qp<numQPs; ++qp)
      compute A;
```

MPI process *n*

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

- ExecutionSpace defined at ***compile time***, e.g.

  typedef Kokkos::OpenMP ExecutionSpace; //MPI+OpenMP

  typedef Kokkos::CUDA ExecutionSpace; //MPI+CUDA

  typedef Kokkos::Serial ExecutionSpace; //MPI-only

- ***Atomics*** used to scatter local data to global data structures

  Kokkos::atomic_fetch_add

- For MPI+CUDA, data transfer from host to device handled by ***CUDA UVM****.

* Unified Virtual Memory.

MPI+X FEA via *Kokkos*

- ***Multi-dimensional parallelism*** for nested for loops via *Kokkos:*

```
for (int cell=0; cell<numCells; ++cell)
  for (int node=0; node<numNodes; ++node)
    for (int qp=0; qp<numQPs; ++qp)
      compute A;
```

MPI process *n*

Kokkos parallelization in ALI master is only over **cells**.**

Thread *1* computes A for (cell,node,qp)=(0,0,0)

Thread *2* computes A for (cell,node,qp)=(0,0,1)

⋮

Thread *N* computes A for (cell,node,qp)=(numCells,numNodes,numQPs)

```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

- ExecutionSpace defined at ***compile time***, e.g.

```
typedef Kokkos::OpenMP ExecutionSpace; //MPI+OpenMP
typedef Kokkos::CUDA ExecutionSpace; //MPI+CUDA
typedef Kokkos::Serial ExecutionSpace; //MPI-only
```

- ***Atomics*** used to scatter local data to global data structures

```
Kokkos::atomic_fetch_add
```

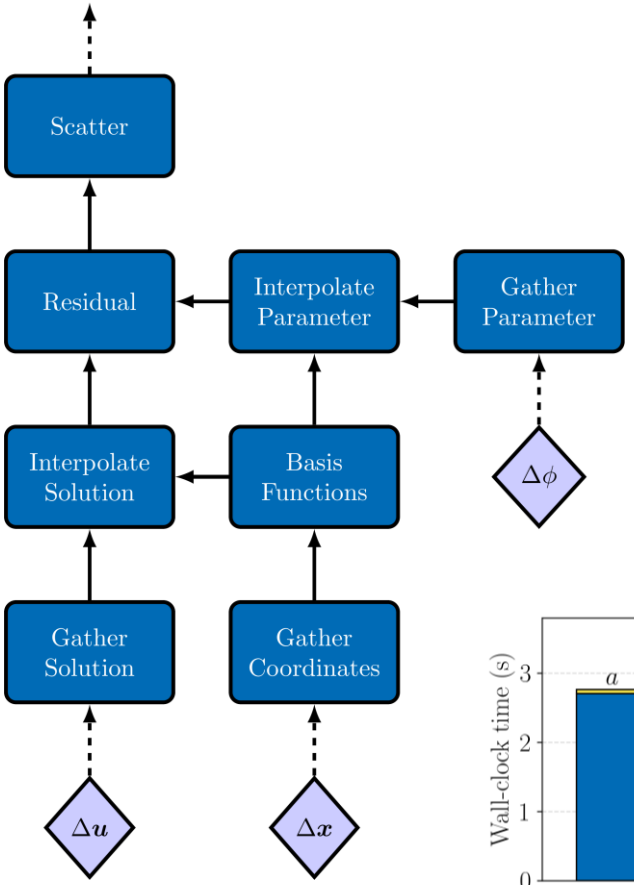
- For MPI+CUDA, data transfer from host to device handled by ***CUDA UVM****.

* Unified Virtual Memory. **Hierarchical parallelism can be up to 2x faster on GPU but adds code bloat & requires padding.

Phalanx: DAG*-based assembly

Sandia National Laboratories

DAG Example



Advantages:

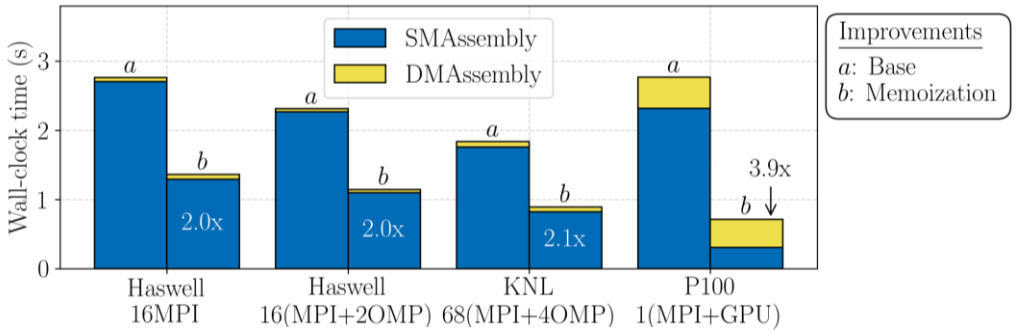
- Increased flexibility, extensibility, usability
- Arbitrary data type support
- Potential for task parallelism

Disadvantage:

- Performance loss through fragmentation

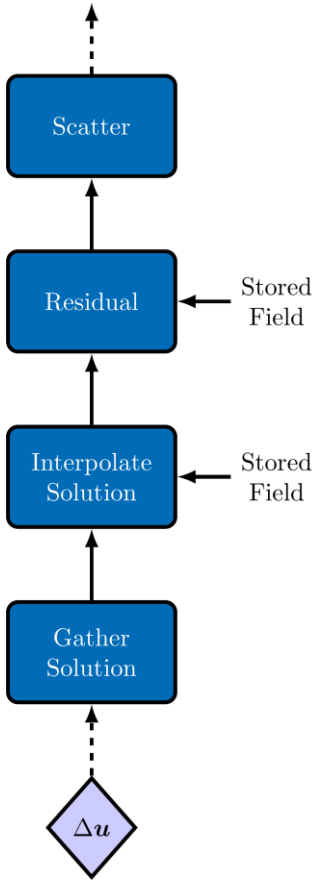
Extension:

- Performance gain through memoization



Single CPU socket or GPU

DAG Example (memoization)



* Directed acyclic graph.

Phalanx Evaluator: templated Phalanx node

Sandia National Laboratories

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 **ExeSpace** (e.g. Serial, OpenMP, CUDA)
- Inline functors are used as kernels
- MDField data layouts
  - Serial/OpenMP – **LayoutRight** (row-major)
  - CUDA – **LayoutLeft** (col-major)


```
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.;
   }
   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);
      }
   }
}
```

Sacado – Automatic Differentiation (AD)

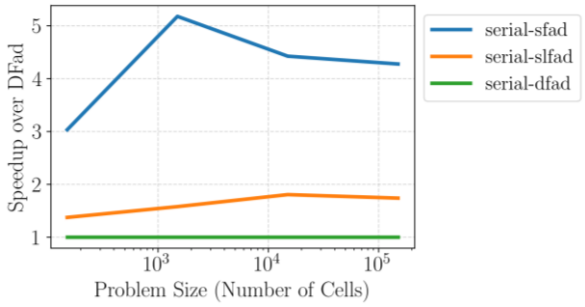
Sandia National Laboratories

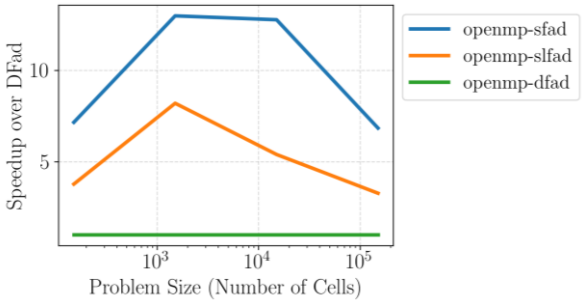
Sacado data types are used for derivative components (ND = # components)

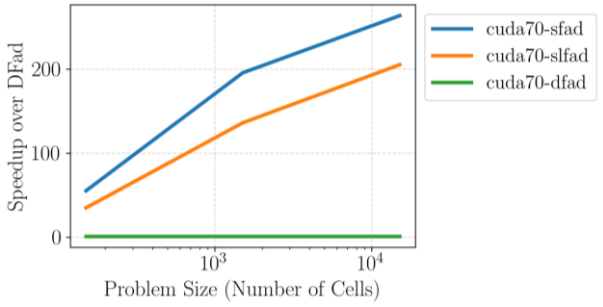
- **DFad** (most flexible) – ND is set at run-time
- **SLFad** (flexible/efficient) – maximum ND set at compile-time
- **SFad** (most efficient) – ND set at compile-time

ND Size Example: Tetrahedral elements (4 nodes), 2 equations, ND = 4*2 = 8

Fad Type Comparison for StokesFO<Jacobian> (Serial, OpenMP (12 threads), CUDA)



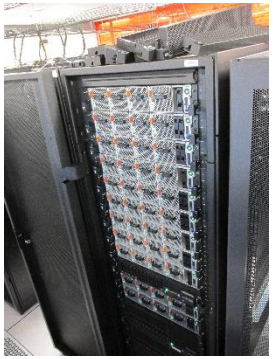




Performance Portability: a response to heterogeneity

Sandia National Laboratories

Generic Definition: For an application, a reasonable level of performance is achieved across a wide variety of computing architectures with the same source code.

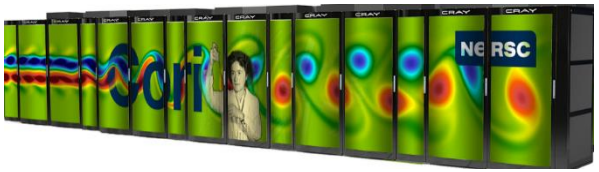


Let's be more specific:

- **Performance** quantified by **application execution time** while strong/weak scaling.
- **Portability** includes conventional CPU, Intel **KNL**, NVIDIA **GPU**.

Approach: MPI+X Programming Model

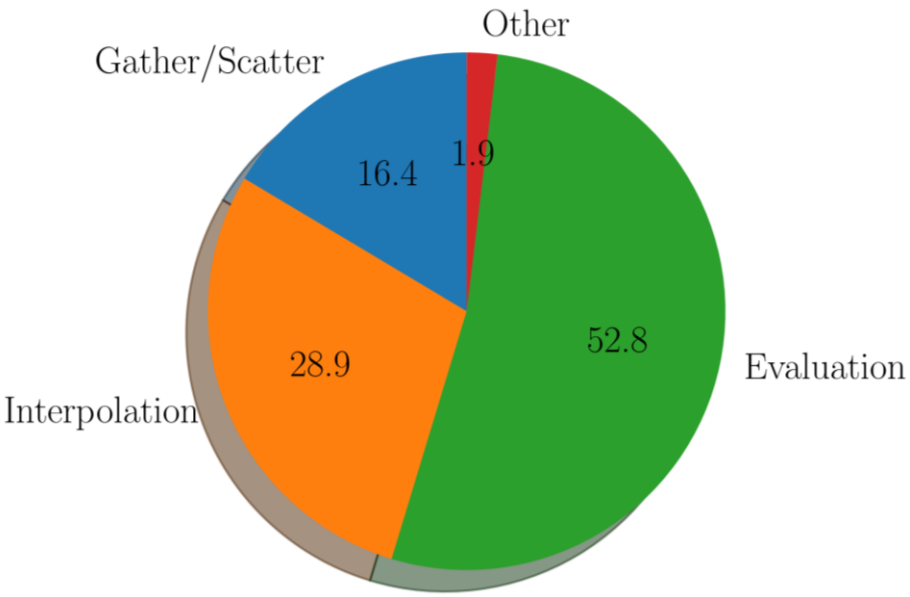
- MPI: **distributed memory** parallelism – Tpetra
- X: **shared memory** parallelism – Kokkos
  - Examples: OpenMP, CUDA
- Minimize **data movement** (efficient programming)
- Increase **arithmetic intensity** (improve compute to memory transfer ratio)
- Saturate **memory bandwidth** (expose more parallelism)



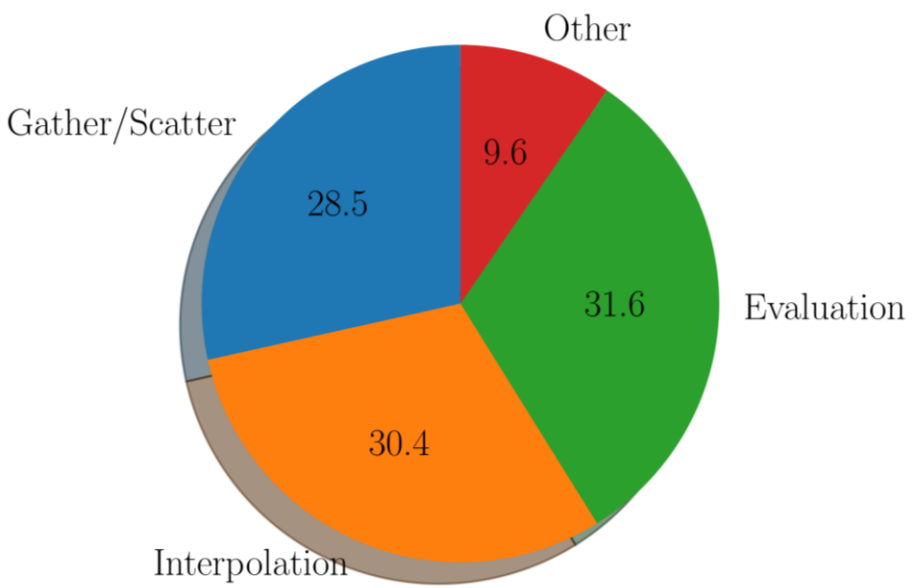
Single CPU/GPU shared memory profile

Sandia National Laboratories

SKX: 24-core



V100: 1 GPU



- Residual/Jacobian **Evaluation** most expensive
- **Gather/Scatter** becoming increasingly important…
- **Other**: some auxiliary routines are still expensive on the GPU (~10%)

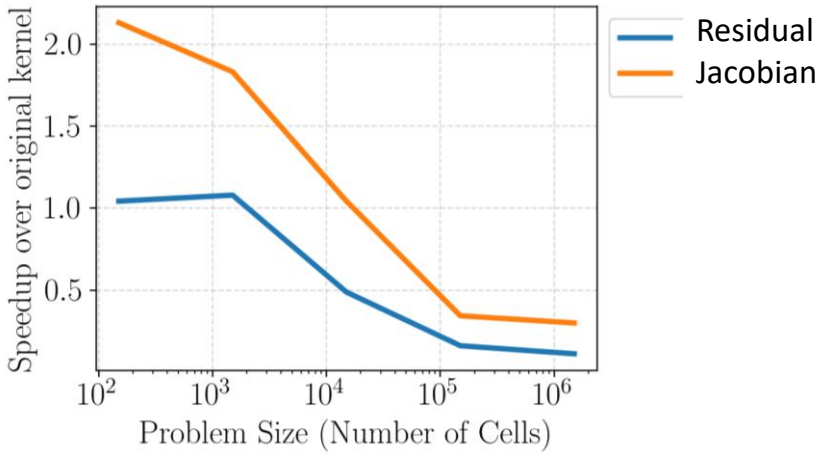
Hierarchical Parallelism

Sandia National Laboratories

Hierarchical parallelism is used to **expose more parallelism** when strong scaling

- Kokkos **TeamPolicy**, **TeamThreadRange** is used to parallelize over **cells** and **nodes**
- Kokkos **scratch space** is used to store node/quadrature values in **shared memory**
- ~2x **speedup** for **small** problem sizes on **GPU** (need padding for large problem sizes)
- **Slowdown** for **all** problem sizes on **CPU** (need different layout)

CUDA70

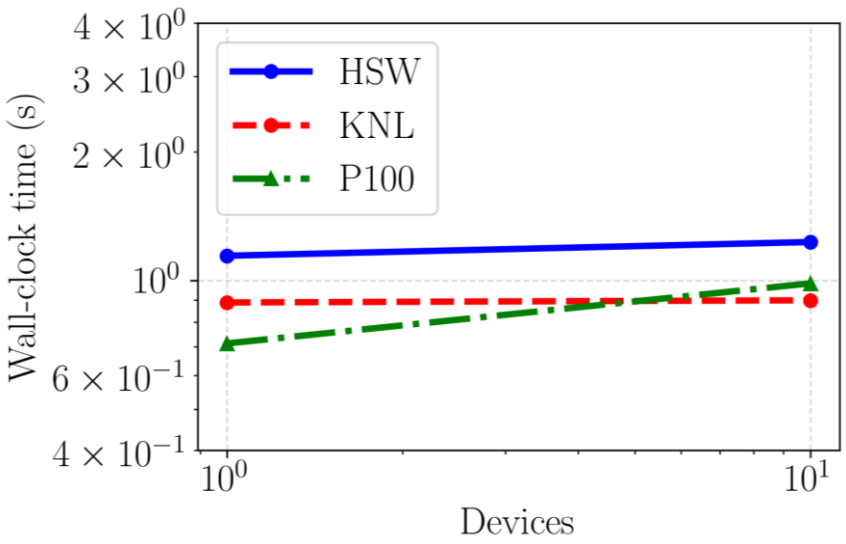


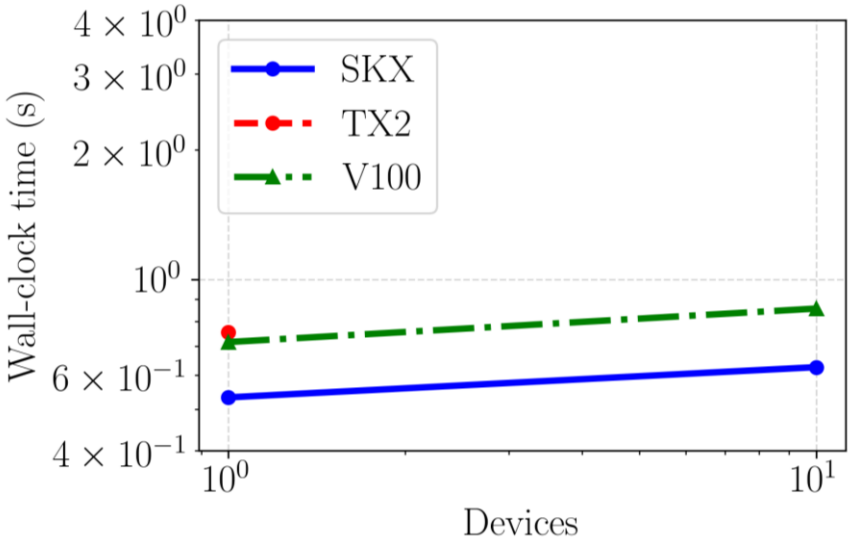

```
template<typename EvalT, typename Traits>
void StokesFOResid<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset) {
   Kokkos::parallel_for(
         Kokkos::TeamPolicy<ExeSpace>(workset.numCells,Kokkos::AUTO()),
         *this);
}

template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void StokesFOResid<EvalT, Traits>::
operator() (const Member& teamMember) const{
   const Index cell = teamMember.league_rank();
   // Allocate shared memory
   ScratchView qpVals(teamMember.team_shmem(), numQPs, fadSize);
   ScratchView nodeVals(teamMember.team_shmem(), numNodes, fadSize);
   // Zero nodeVals
   Kokkos::parallel_for(
   Kokkos::TeamThreadRange(teamMember, numNodes), [&] (const Index& node) {
      nodeVals(node) = 0; });
   // Fill Ugrad00
   Kokkos::parallel_for(
   Kokkos::TeamThreadRange(teamMember, numQPs), [&] (const Index& qp) {
      qpVals(qp) = Ugrad(cell,qp,0,0); });
   // Calc Ugrad00 contribution
   for (Index qp=0; qp < numQPs; ++qp) {
      Kokkos::parallel_for(
      Kokkos::TeamThreadRange(teamMember, numNodes), [&] (const Index& node)
         nodeVals(node) += qpVals(qp) * wGradBF(cell,node,qp,0); }); }
…
   // Copy to Residual0
   Kokkos::parallel_for(
   Kokkos::TeamThreadRange(teamMember, numNodes), [&] (const Index& node) {
      Residual(cell,node,0) = nodeVals(node); });
}
```

Performance results: weak scalability

Sandia National Laboratories

<u>Legend</u>: HSW, SKX=Haswell, Skylake CPU; KNL=Xeon Phi; TX2=ThunderX2; P100,V100=GPU

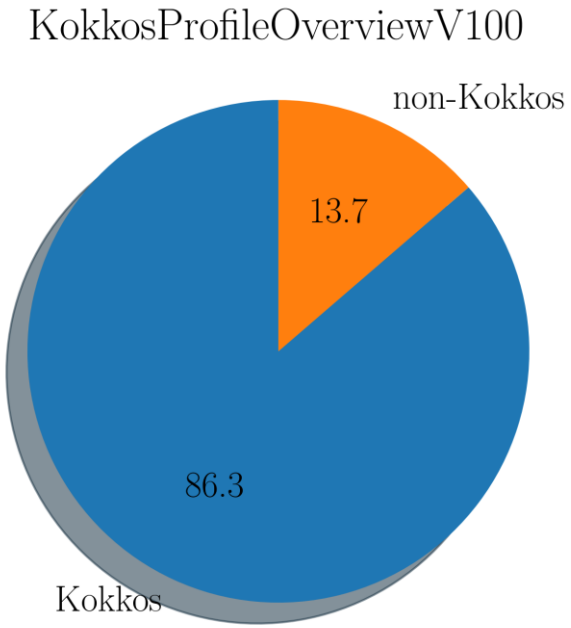


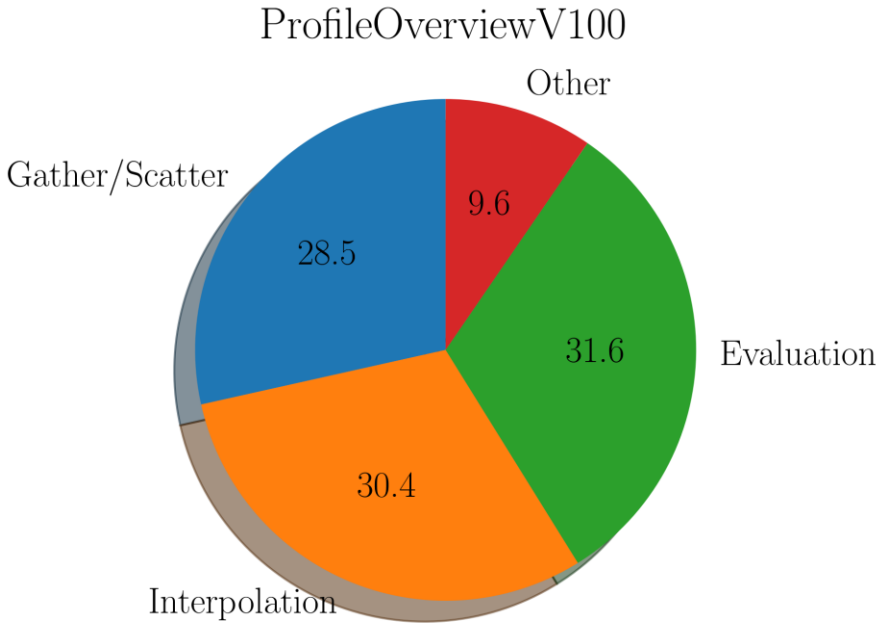


Reasonable scaling across all devices **w/o** machine-specific optimization in Albany

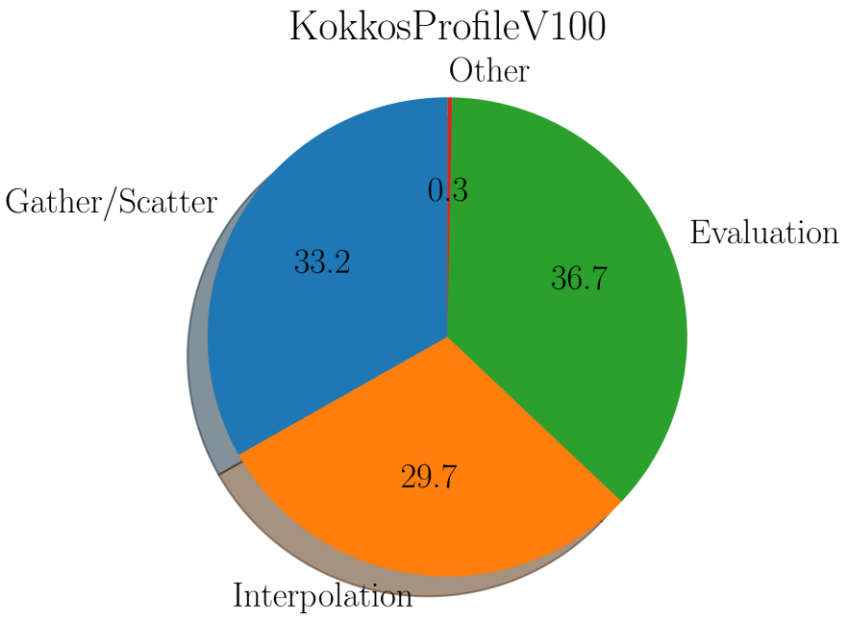
- Poor GPU scaling (Export WIP within Tpetra)
- Best case: Skylake at 10 devices (280 cores)

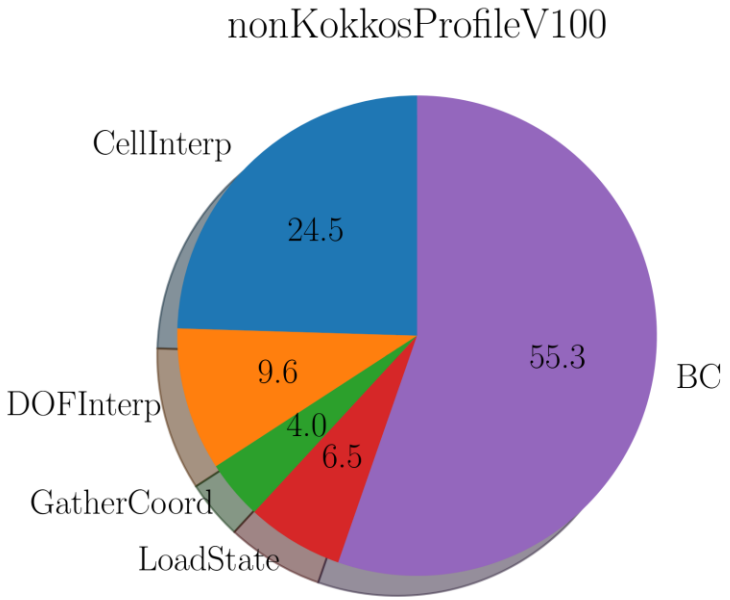
Single GPU: full profile





Single GPU: Kokkos and non-Kokkos





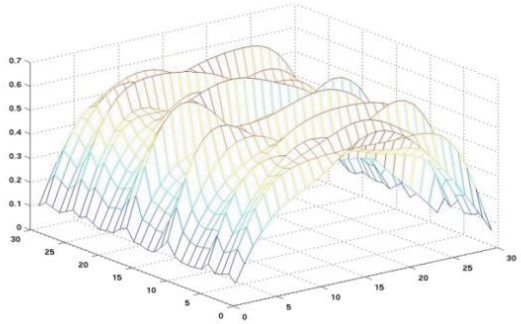
Solver challenge: Thin meshes

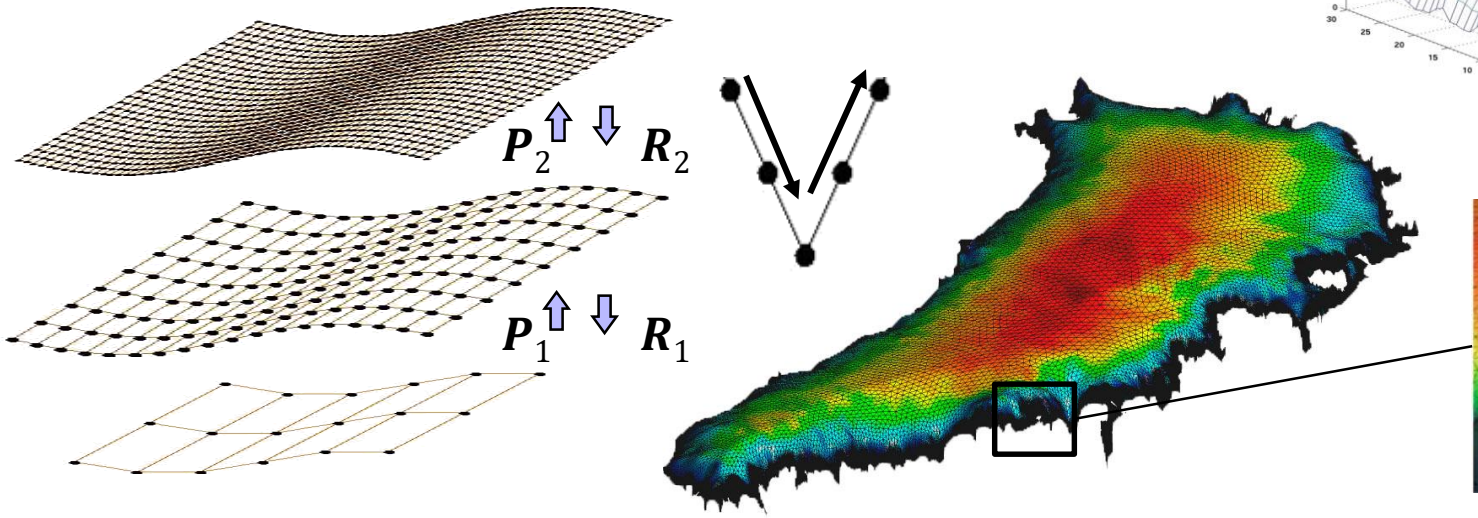
Sandia National Laboratories

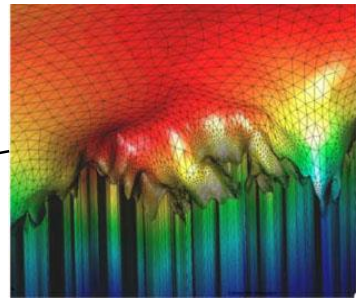
Meshes with anisotropy/bad aspect ratios: ice sheets are thin (thickness up to 4 km, horizontal extension of thousands km)

- Problem for ***multi-grid solvers***: if coarsening equally in all three coordinate directions, horizontal/vertical info gets "jumbled" and it is hard to smooth horizontal errors.
  - ***Point relaxation*** is inefficient in reducing errors in weak direction.

Right: point Jacobi error after 30 iterations. Errors are oscillatory in x-dimension. y-dimension is analogous to thin dimension in 3D land ice mesh.







Above left: illustration of multi-grid solver (V-cycle). Above right: thin extruded meshes

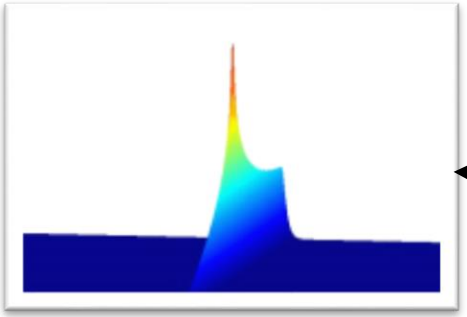
Antarctica solver challenge: Floating ice

Sandia National Laboratories

Ill-conditioning** associated with **floating ice boundary condition.

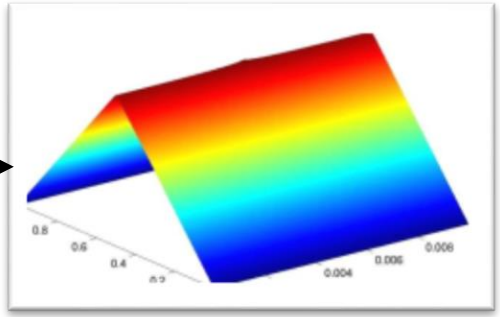
- **Grounded ice (GIS)**: Green's function shows ***rapid decay*** in ***horizontal direction***
  $\Longrightarrow$ preconditioner need not approximate long distance horizontal couplings
  - Vertical line solvers or ***ILU w/ layer-wise ordering*** + ***2D parallel DD*** (right) can work well (vertical coupling accurately captured)

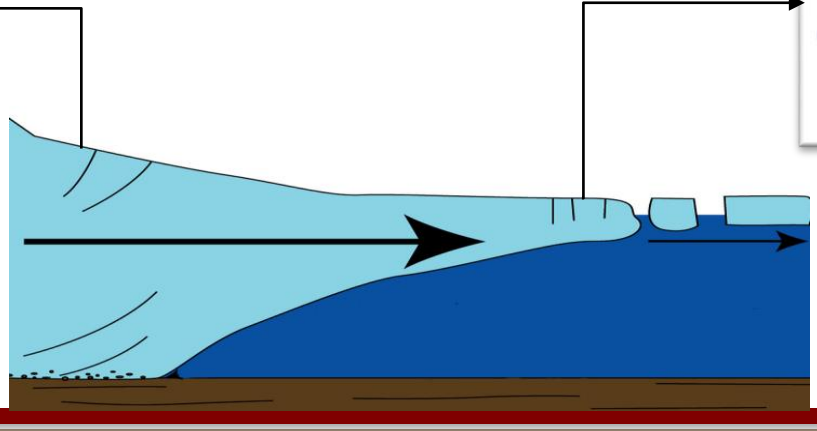
- **Floating ice (AIS)**: Green's function is ***nearly constant*** in ***thin direction***
  - ***ILU will not work well***: large Krylov space is needed to capture Green's function, preconditioner with spatially global character is insufficient



Thin grounded ice: ILU can work well with proper ordering

Horizontal Green's function decay
(Tuminaro *et al., SISC* 2016)

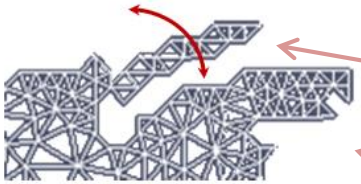




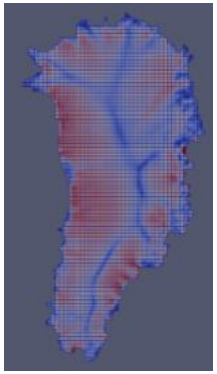
Solver challenge: Islands hinged peninsulas

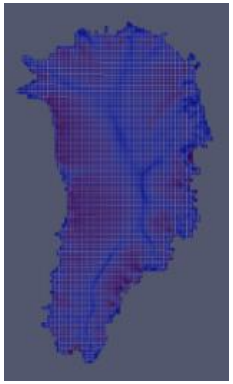
Sandia National Laboratories

Islands and certain hinged peninsulas lead to solver failures



- Rigid body translations and x-y plane rotations of islands/ peninsulas are correspond to ***nullspace components.***
- We have developed an algorithm to detect/remove problematic ***hinged peninsulas*** & ***islands*** based on coloring and repeated use of connected component algorithms (Tuminaro *et al. SISC*, 2016).
- Solves are ***~2x faster*** with hinges removed.
- ***WIP:*** C++ implementation within Trilinos for integration into dycores.





Greenland Problem

Resolu-tion	ILU – hinges	ILU – no hinges	ML – hinges	ML – no hinges
8km/5 layers	878 sec, 84 iter/solve	693 sec, 71 iter/solve	254 sec, 11 iter/solve	220 sec, 9 iter/solve
4km/10 layers	1953 sec, 160 iter/solve	1969 sec, 160 iter/solve	285 sec, 13 iter/solve	245 sec, 12 iter/solve
2km/20 layers	10942 sec, 710 iter/solve	5576 sec, 426 iter/solve	482 sec, 24 iter/solve	294 sec, 15 iter/solve
1km/40 layers	--	15716 sec, 881 iter/solve	668 sec, 34 iter/solve	378 sec, 20 iter/solve

Summary and outlook

Sandia National Laboratories

- A ***performance portable*** implementation of the FEA in the ***ALI*** model was created using ***Kokkos*** within the Albany code base.
  - With this implementation, the ***same code*** can run on devices with drastically ***different memory models*** (many-core CPU, GPU, Intel Xeon Phi, etc.).
  - Only "optimization" we have done for ***portability*** involved ***minimizing data movement*** (via memoization), which improved code performance on ***all architectures***.
  - ***Further optimization*** can be done to improve resource utilization.

See (Demeshko *et al., J. HPC. Appl.*, 2018) and (Watkins *et al., LNCSE,* 2020) for more **details** on our **performance portability efforts** in Albany using Kokkos.

- ***Scalable, fast*** and ***robust*** linear solve is achieved via algebraic multigrid (AMG) preconditioner that takes advantage of layered nature of meshes.
  - Performance portability of linear solve is work in progress.

See (Tezaur *et al., Procedia CS*, 2015) and (Tuminaro *et al., SISC,* 2016) for more **details** on our **AMG preconditioner/linear solver** work.

We are making **progress** towards running *Albany/Land Ice* on **heterogeneous HPC architectures** with the help of **Kokkos** and **Trilinos!**

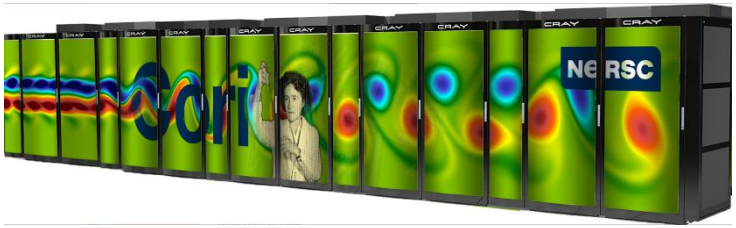
Ongoing and future work

Sandia National Laboratories

Finite Element Assembly (FEA):

- ***Profiling*** on CPUs and GPUs.
- Methods for ***improving performance***:
  - Reduce excess memory usage.
  - Replace CUDA UVM with manual memory transfer.
  - Further research into portable hierarchical parallelism.
  - Improve matrix export (FECrsMatrix in Tpetra).
- ***Large-scale*** runs on Cori and Summit.

Albany



Linear Solve:

- Performance-portability of ***preconditioned iterative linear solve*** using *Kokkos* for implicit problems in Albany (e.g., ALI).
  - All the pieces are there in Belos/Ifpack2/MueLu for us to try running on GPUs and other advanced architectures
  - We are also looking at other solvers, e.g., hierarchical solvers, Shylu (FAST-ILU, multi-threaded GS).