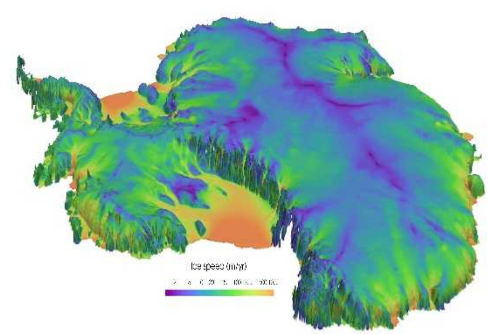
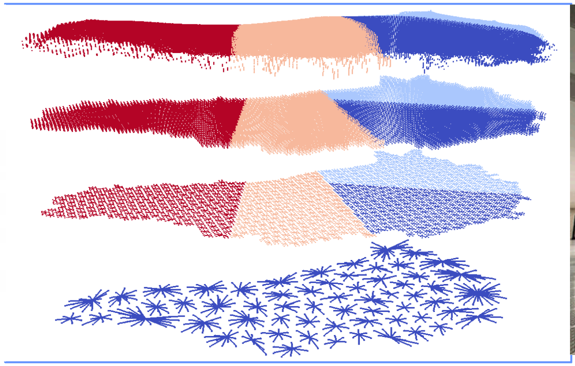
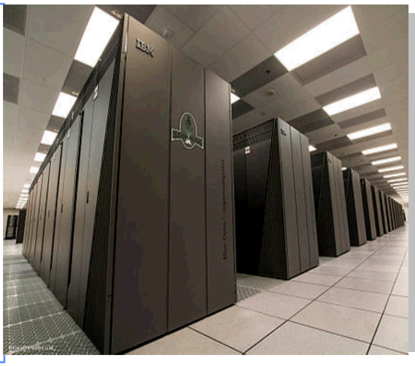
SAND2017-9362C







Performance & Performance Portability of the Albany/FELIX Finite Element Land-Ice Solver

I. Tezaur[1], J. Watkins[1], R. Tuminaro[1], I. Demeshko[2]

[1] Sandia National Laboratories, Albuquerque, NM and Livermore, CA, USA.
[2] Los Alamos National Laboratory, Los Alamos, NM, USA.

SIAM GS 2017 Erlangen, Germany September 11-14, 2017

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.

Outline

1. Background.
   - Motivation.
   - PISCEES project for land-ice modeling.
   - Albany/FELIX "First-Order Stokes" land-ice model.

2. Finite Element Assembly.
   - Performance-portability via Kokkos.

3. Linear Solve.
   - AMG preconditioning.

4. Summary & future work.

Outline

1. **Background.**
   - **Motivation.**
   - PISCEES project for land-ice modeling.
   - Albany/FELIX "First-Order Stokes" land-ice model.

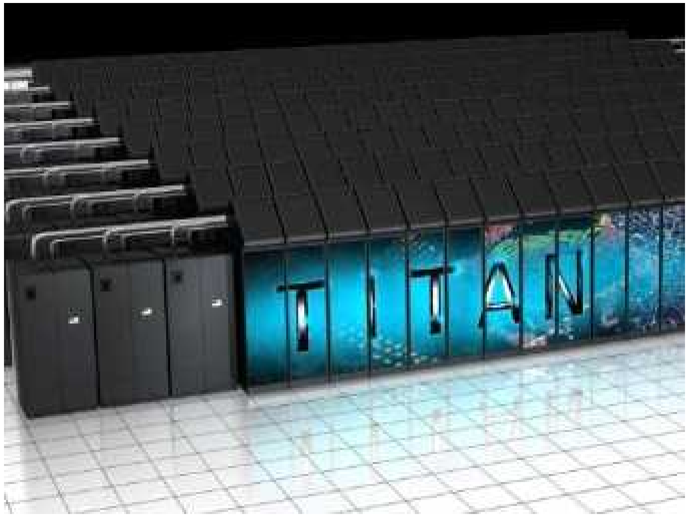
2. Finite Element Assembly.
   - Performance-portability via Kokkos.

3. Linear Solve.
   - AMG preconditioning.

4. Summary & future work.

Motivation

- Scientific models (e.g. climate models) 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***.



MPI+X Programming Model

- HPC architectures are rapidly changing, but ***trends*** remain the same.
    - ***Computations*** are ***cheap***, ***memory transfer*** is ***expensive***.
    - ***Single core cycle*** time has improved but stagnated.
    - Increased ***computational power*** achieved through ***manycore architectures***.

  → MPI-only is not enough to exploit emerging massively parallel architectures.

Approach: MPI+X Programming Model

Year	Memory Access Time	Single Core Cycle Time
1980s	~100 ns	~100 ns
Today	~50-100 ns	~1 ns

- MPI: inter-node parallelism.
- X: intra-node parallelism.

  → *Examples:* X = OpenMP, CUDA, Pthreads, etc.

Outline

1. **Background.**
   - Motivation.
   - **PISCEES project for land-ice modeling.**
   - Albany/FELIX "First-Order Stokes" land-ice model.
2. Finite Element Assembly.
   - Performance-portability via Kokkos.
3. Linear Solve.
   - AMG preconditioning.
4. Summary & future work.

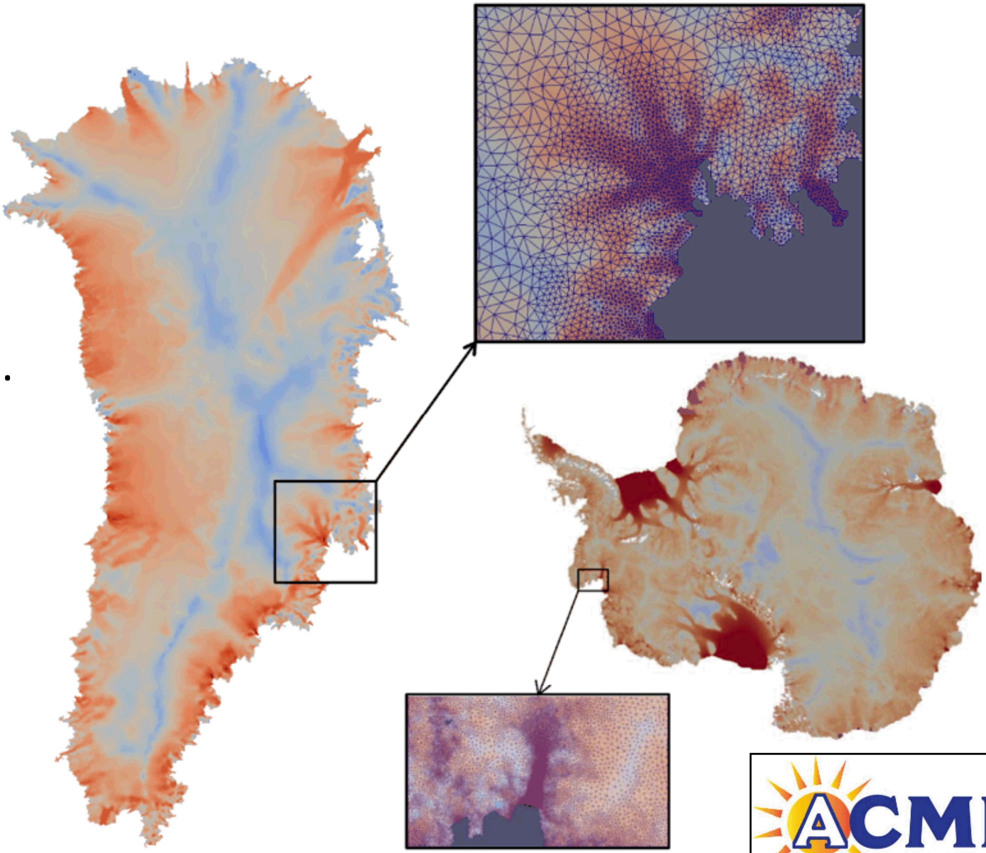
PISCEES Project for Land-Ice Modeling

"PISCEES" = Predicting Ice Sheet Climate Evolution at Extreme Scales
5 year SciDAC3 project currently coming to an end; follow-up 5 year continuation project funded under SciDAC4 and starting now!*

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

Requirements for *Albany/FELIX*:

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



As part of **ACME *DOE Earth System Model***, solver will provide actionable predictions of 21st century sea-level change (including uncertainty bounds).

* See talk by Steve Price in this MS. ** Finite Elements for Land Ice eXperiments

Outline

1. **Background.**
   - Motivation.
   - PISCEES project for land-ice modeling.
   - **Albany/FELIX "First-Order Stokes" land-ice model.**

2. Finite Element Assembly.
   - Performance-portability via Kokkos.

3. Linear Solve.
   - AMG preconditioning.

4. Summary & future work.

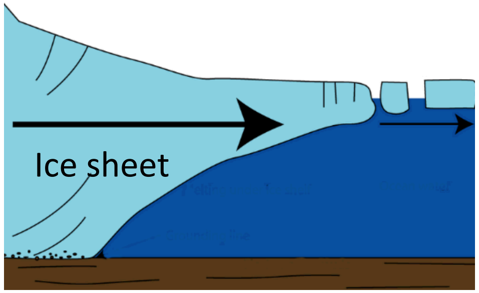
First-Order (FO) Stokes Model

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

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

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

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



First-Order (FO) Stokes Model

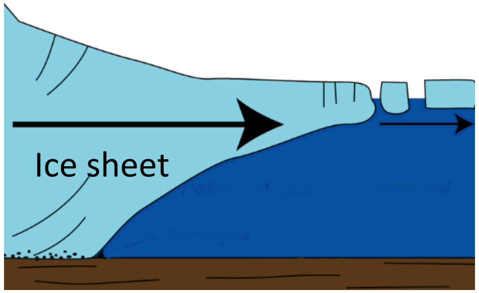
- Ice velocities given by the ***"First-Order" Stokes PDEs*** with nonlinear viscosity:

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

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

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

<u>Algorithmic choices for Albany/FELIX:</u>



First-Order (FO) Stokes Model

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

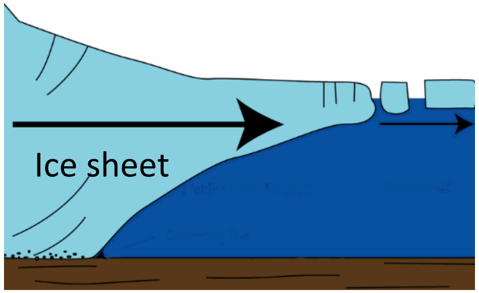
$$\begin{cases} -\nabla\cdot(2\mu\dot{\epsilon}_1) = -\rho g\dfrac{\partial s}{\partial x} \\ -\nabla\cdot(2\mu\dot{\epsilon}_2) = -\rho g\dfrac{\partial s}{\partial y} \end{cases}$$

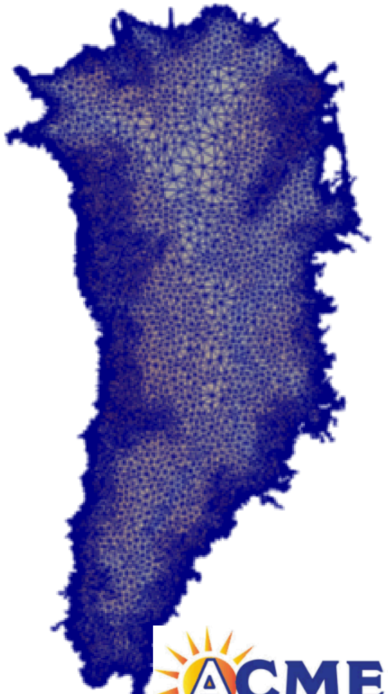
$$\mu = \frac{1}{2}A^{-\frac{1}{n}}\left(\frac{1}{2}\sum_{ij}\dot{\epsilon}_{ij}^{\,2}\right)^{\left(\frac{1}{2n}-\frac{1}{2}\right)}$$

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

Algorithmic choices for Albany/FELIX:

- ***3D unstructured grid*** FEM discretization.





First-Order (FO) Stokes Model

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

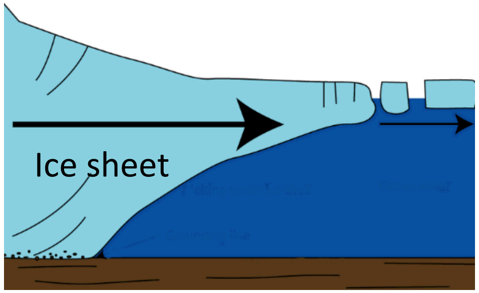
$$\begin{cases} -\nabla\cdot(2\mu\dot{\epsilon}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla\cdot(2\mu\dot{\epsilon}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}$$

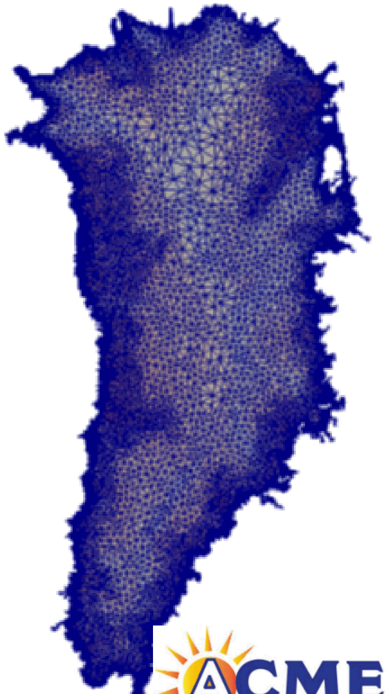
$$\mu = \frac{1}{2}A^{-\frac{1}{n}}\left(\frac{1}{2}\sum_{ij}\dot{\epsilon}_{ij}{}^{2}\right)^{\left(\frac{1}{2n}-\frac{1}{2}\right)}$$

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

<u>Algorithmic choices for Albany/FELIX:</u>

- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.





First-Order (FO) Stokes Model

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

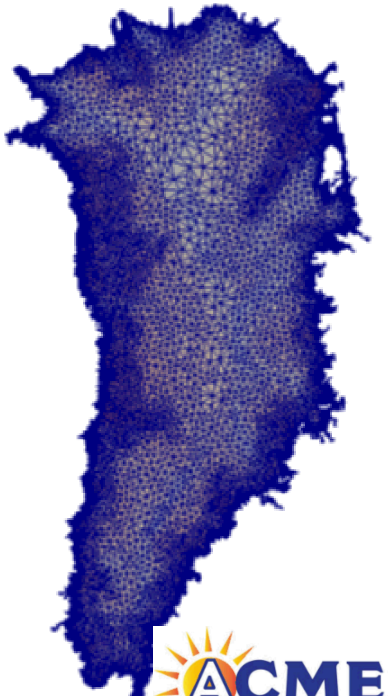
$$\begin{cases} -\nabla\cdot(2\mu\dot{\epsilon}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla\cdot(2\mu\dot{\epsilon}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}$$

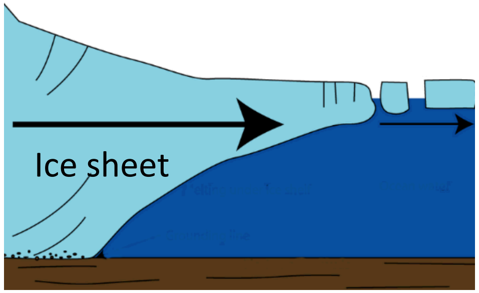
$$\mu = \frac{1}{2} A^{-\frac{1}{n}} \left(\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}^{\,2} \right)^{\left(\frac{1}{2n} - \frac{1}{2}\right)}$$

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

Algorithmic choices for Albany/FELIX:

- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- ***Algebraic-multigrid*** preconditioned Krylov linear solvers.





First-Order (FO) Stokes Model

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

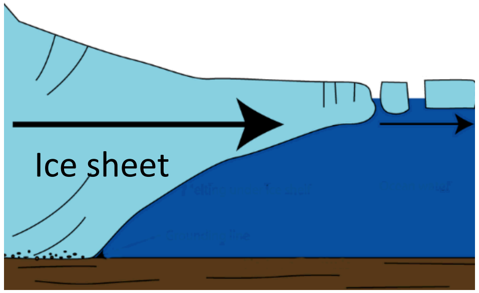
$$\begin{cases} -\nabla \cdot (2\mu \dot{\epsilon}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla \cdot (2\mu \dot{\epsilon}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}$$

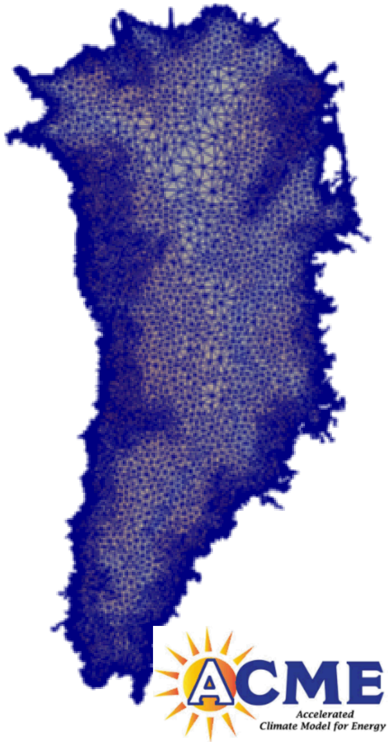
$$\mu = \frac{1}{2} A^{-\frac{1}{n}} \left(\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}^{\,2} \right)^{\left(\frac{1}{2n} - \frac{1}{2}\right)}$$

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

Algorithmic choices for Albany/FELIX:

- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- ***Algebraic-multigrid*** preconditioned Krylov 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:

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

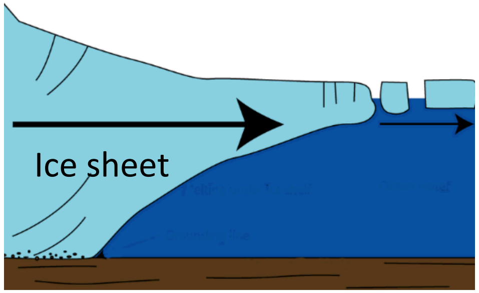
$$\mu = \frac{1}{2}A^{-\frac{1}{n}}\left(\frac{1}{2}\sum_{ij}\dot{\epsilon}_{ij}^{2}\right)^{\left(\frac{1}{2n}-\frac{1}{2}\right)}$$

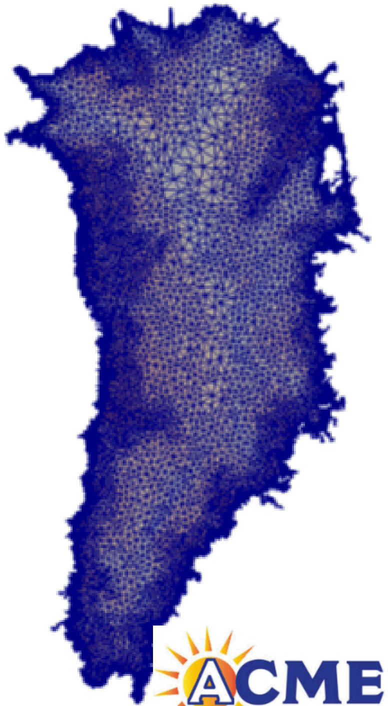
$$\dot{\epsilon}_1^{T} = (2\dot{\epsilon}_{11} + \dot{\epsilon}_{22}, \dot{\epsilon}_{12}, \dot{\epsilon}_{13})$$
$$\dot{\epsilon}_2^{T} = (2\dot{\epsilon}_{12}, \dot{\epsilon}_{11} + 2\dot{\epsilon}_{22}, \dot{\epsilon}_{23})$$
$$\dot{\epsilon}_{\mathrm{ij}} = \frac{1}{2}\left(\frac{\partial u_i}{\partial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

<u>Algorithmic choices for Albany/FELIX:</u>

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

Albany/FELIX implemented in open-source* multi-physics FE Trilinos-based code: Albany





* https://github.com/gahansen/Albany.

First-Order (FO) Stokes Model

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

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

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

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

Algorithmic choices for Albany/FELIX:

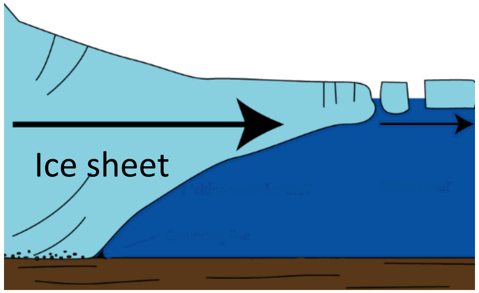
- ***3D unstructured grid*** FEM discretization.
- ***Newton method*** nonlinear solver with automatic differentiation Jacobians.
- ***Algebraic-multigrid*** preconditioned Krylov linear solvers.
- ***Advanced analysis capabilities***: deterministic inversion, calibration, UQ.

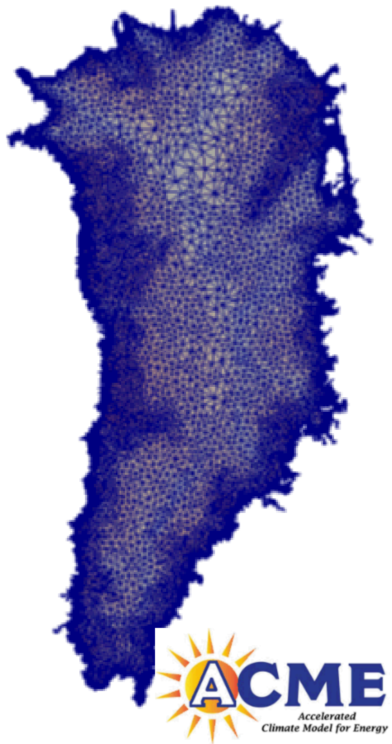
Albany/FELIX implemented in open-source* multi-physics FE Trilinos-based code: Albany

Implicit solver:

FEA** = 50% CPU-time

Linear solve = 50% CPU-time





* https://github.com/gahansen/Albany.   **Finite Element Assembly

Outline

1. Background.
   - Motivation.
   - PISCEES project for land-ice modeling.
   - Albany/FELIX "First-Order Stokes" land-ice model.

2. **Finite Element Assembly.**
   - **Performance-portability via Kokkos.**

FEA = 50% CPU-time

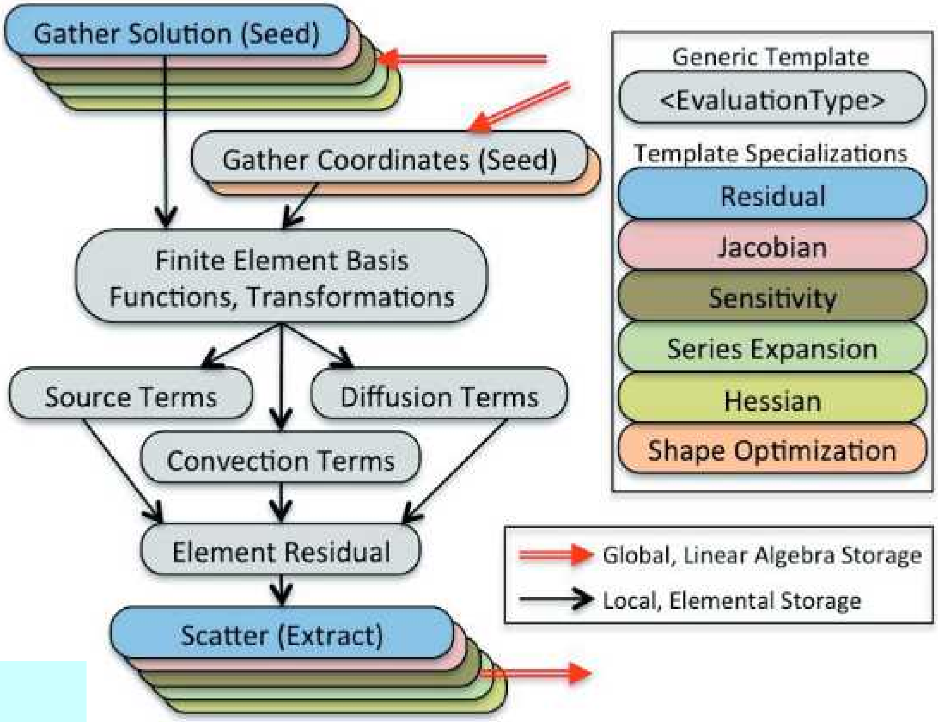
3. Linear Solve.
   - AMG preconditioning.

4. Summary & future work.

Albany/FELIX 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.

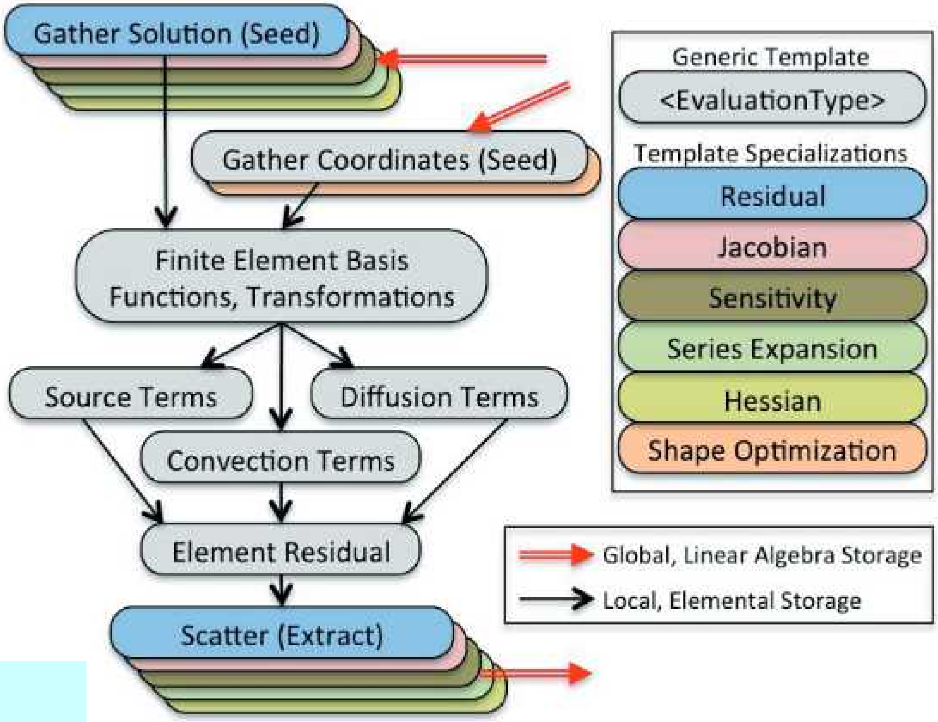
Performance-portability: focus on *FEA*.



Albany/FELIX 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*.



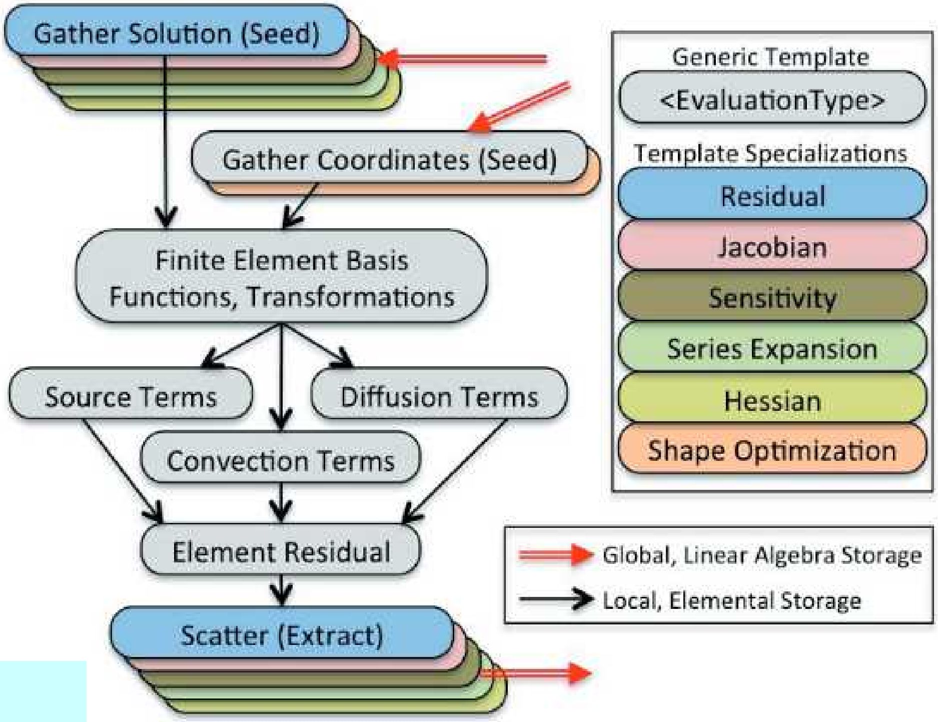
Problem Type	% CPU time for FEA
Implicit	50%
Explicit	99%

Albany/FELIX 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.

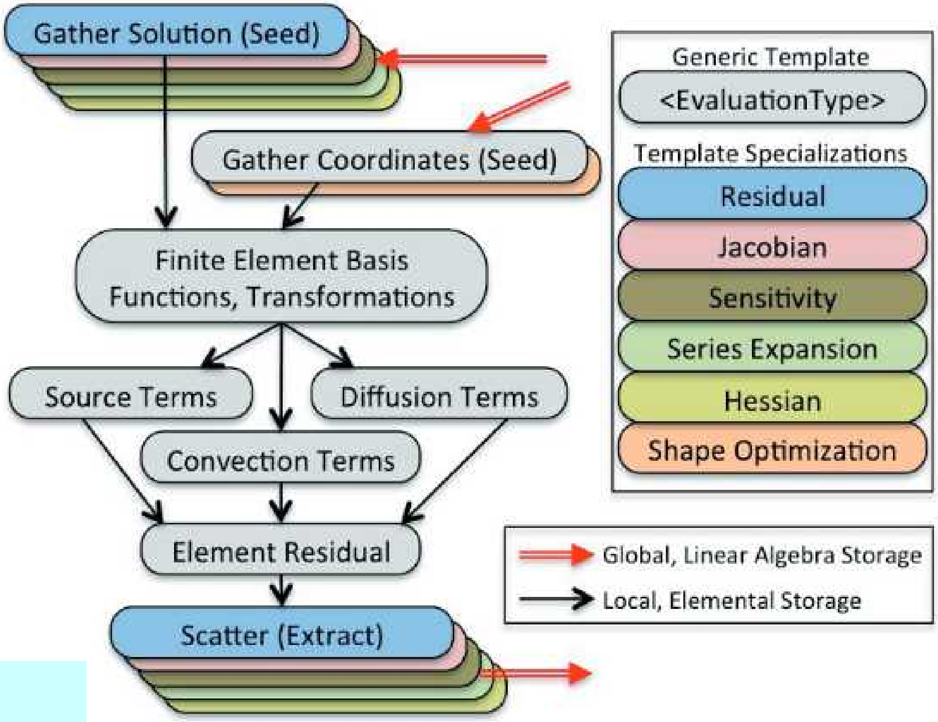


Problem Type	% CPU time for FEA
Implicit	50%
Explicit	99%

Albany/FELIX 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%

Performance-portability via *Kokkos*

We need to be able to run climate models on ***new architecture machines*** (hybrid systems) and ***manycore devices*** (multi-core CPU, NVIDIA GPU, Intel Xeon Phi, etc.) .

- In Albany/FELIX, we achieve performance-portability via ***Kokkos.***
  - ***Kokkos****:* C++ library and programming model that provides performance portability across multiple computing architectures.

    → *Examples:* Multicore CPU, NVIDIA GPU, Intel Xeon Phi, and more.
  - Provides ***automatic access*** to OpenMP, CUDA, Pthreads, etc.
  - Designed to work with the ***MPI+X*** programming model.
  - Abstracts ***data layouts*** for optimal performance ("array of strucs" 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 researcher to focus on ***application development*** for large heterogeneous architectures.

MPI+X FEA via *Kokkos*

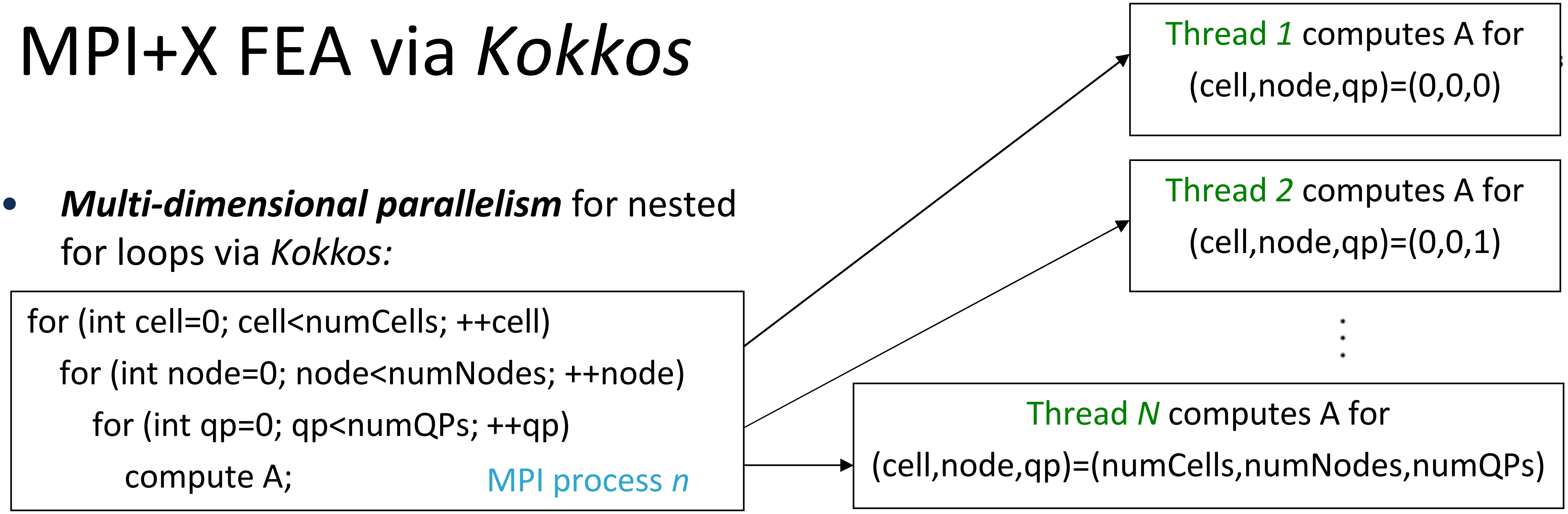
- ***MPI-only*** nested for loop:

```
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)

Single Threading
Task 1
Task 2
Task 3
Task 4
Thread 1
Core 1

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)

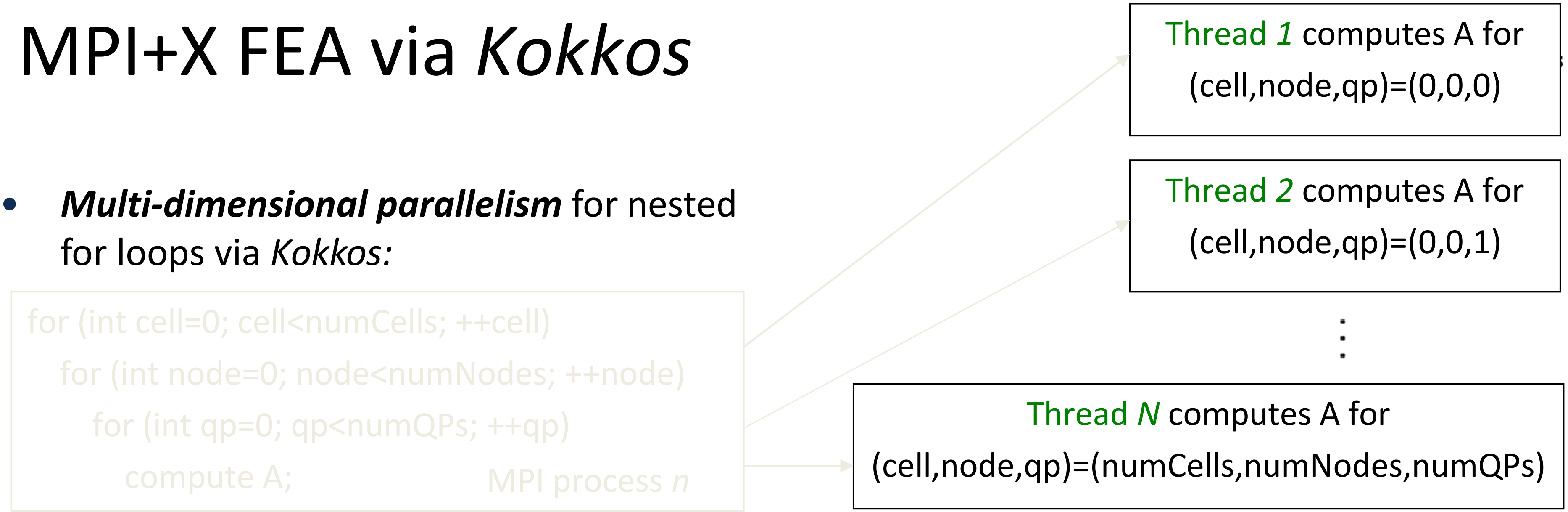
```
computeA_Policy range({0,0,0},{(int)numCells,(int)numNodes,(int)numQPs});
Kokkos::Experimental::md_parallel_for<ExecutionSpace>(range,*this);
```

Single Threading
Task 1
Task 2
Task 3
Task 4
Thread 1
Core 1

* 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)

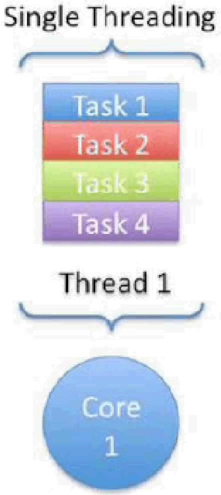
⋮

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

Single Threading
Task 1
Task 2
Task 3
Task 4
Thread 1
Core 1

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
```

Single Threading
Task 1
Task 2
Task 3
Task 4
Thread 1
Core 1

- ***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 FELIX 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
```

Single Threading
Task 1
Task 2
Task 3
Task 4
Thread 1
Core 1

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

```
Kokkos::atomic_fetch_add
```

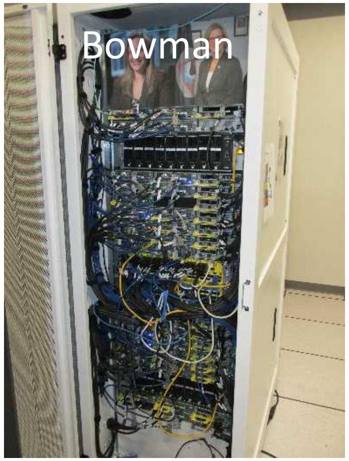
- For MPI+CUDA, data transfer from host to device handled by ***CUDA UVM****.

* Unified Virtual Memory.

Computer Architectures

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

- **Ride** (SNL) used for verification, performance tests
  12 nodes (dual-Power8 (16 cores) + P100 quad-GPU)
- **Bowman** (SNL) used for verification
  10 nodes (Intel Xeon Phi KNL (68 cores))
- **Cori** (NERSC) used for verification, performance tests
  9688 nodes (Intel Xeon Phi KNL (68 cores))
- **Summit** (ORLCF) is ultimate GPU target
  4600 nodes (dual-Power9 + 6 NVIDIA Volta)



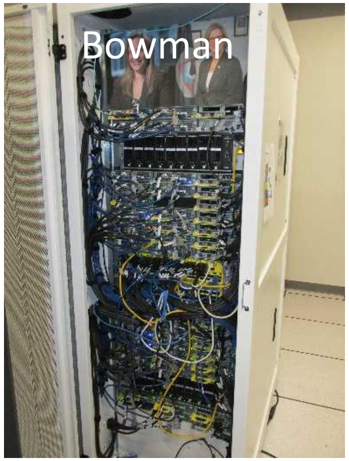
Computer Architectures

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

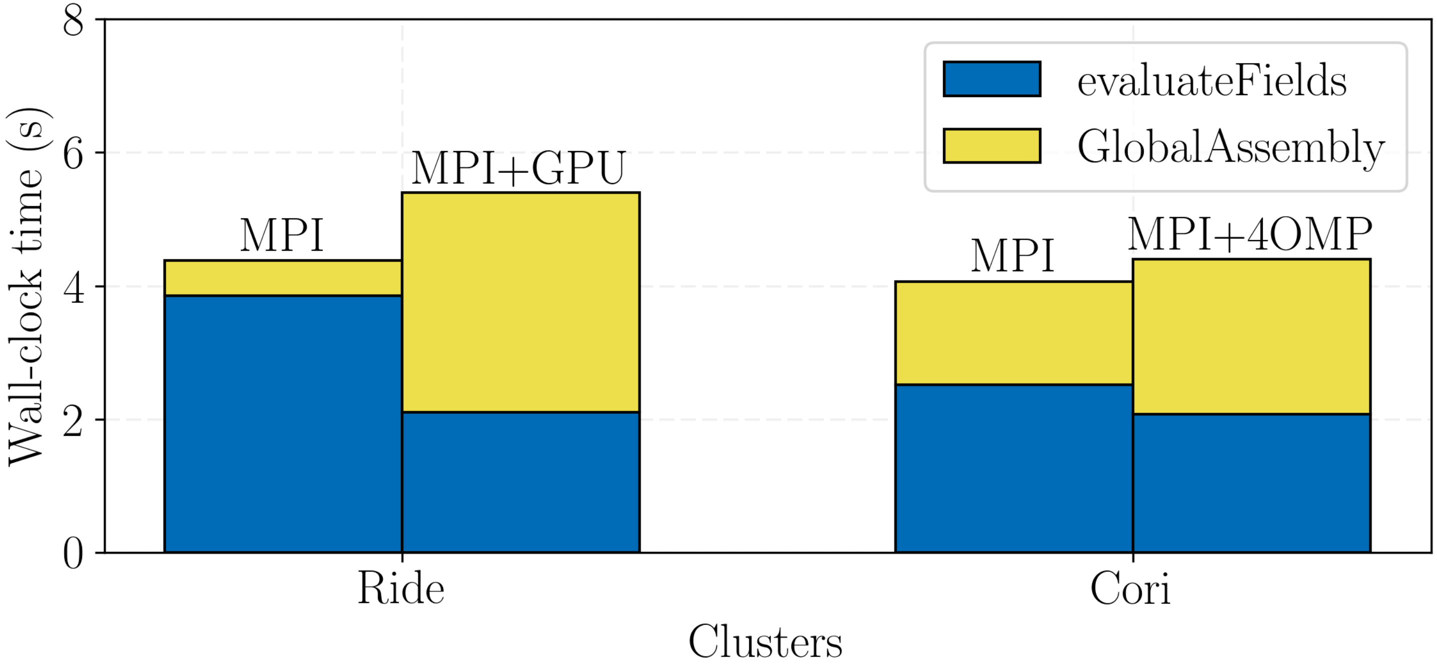
- **Ride** (SNL) used for verification, performance tests
  12 nodes (dual-Power8 (16 cores) + P100 quad-GPU)
- **Bowman** (SNL) used for verification
  10 nodes (Intel Xeon Phi KNL (68 cores))

Platforms utilized here.

- **Cori** (NERSC) used for verification, performance tests
  9688 nodes (Intel Xeon Phi KNL (68 cores))
- **Summit** (ORLCF) is ultimate GPU target
  4600 nodes (dual-Power9 + 6 NVIDIA Volta)



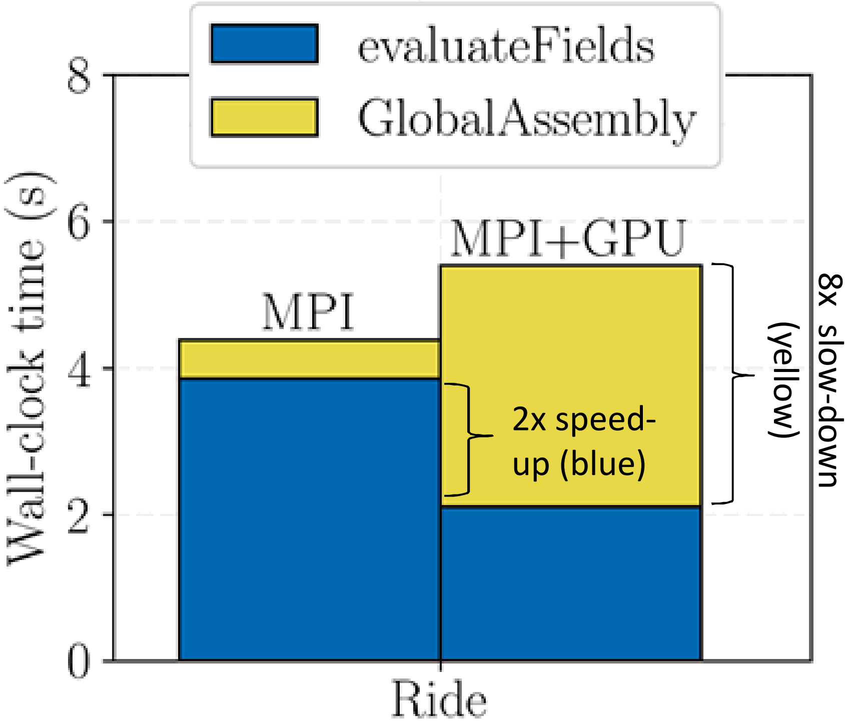
GIS Kokkos Execution Space Comparison



A single node comparison: 16MPI vs. 4(MPI+GPU) [*Left*], 68MPI vs. 68(MPI+4OMP) [*Right*]
(GIS 4km-20km unstructured mesh with 1.51M tet elements)

Blue (evaluateFields): mostly Residual/Jacobian computation + Gather/Scatter;
Yellow (GlobalAssembly): mostly communication + Trilinos operations.

GIS Kokkos 16 MPI vs. 4(MPI+GPU) Results



GlobalAssembly (yellow): mostly communication + Trilinos operations

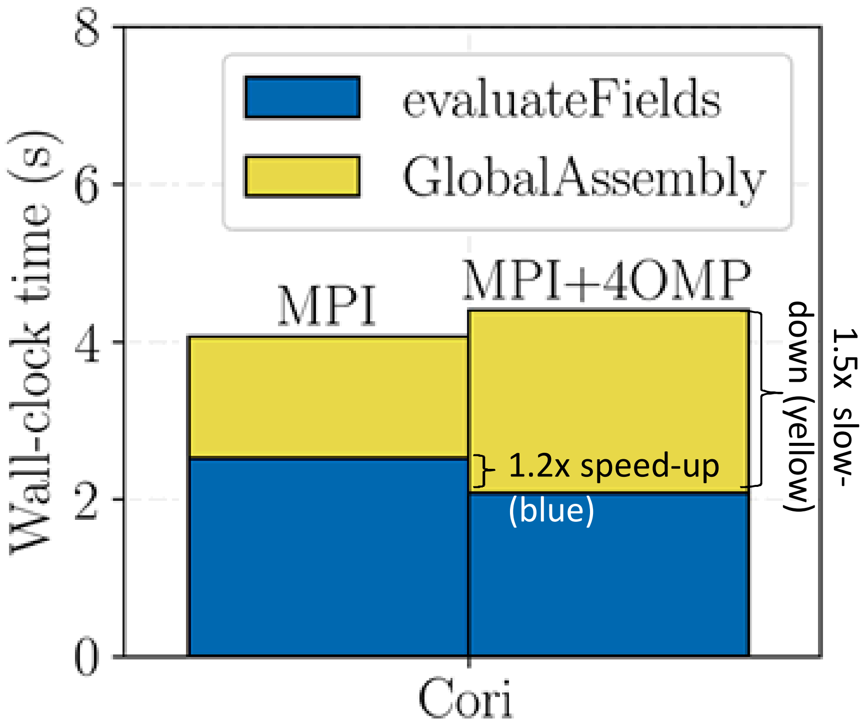
- 8x slow-down is not reasonable.
- Most slow-downs are in Trilinos (Tpetra) → Trilinos packages are currently being reworked using CUDA-aware MPI.

Summary: speed-ups on GPU are not yet as expected but may be improved by introducing padding, removing CudaUVM and unnecessary data movement, switching to CUDA-aware MPI.

evaluateFields (blue): mostly residual/Jacobian computation + Gather/Scatter

- 2x speedup not much considering 4 GPUs (desirable speedup: 10x or more).
- evaluateFields<Jacobian> much faster than evaluateFields<Residual> b/c there is more work in computing Jacobian.
- Data movement is lagging speedup – can be improved by removing CudaUVM, data padding to prevent data misalignment.
- A few kernels still for boundary conditions still need to be ported to Kokkos.

GIS Kokkos 68MPI vs. 68(MPI+4OMP) Results



1.2x speed-up (blue)

1.5x slow-down (yellow)

GlobalAssembly (yellow): mostly communication + Trilinos operations

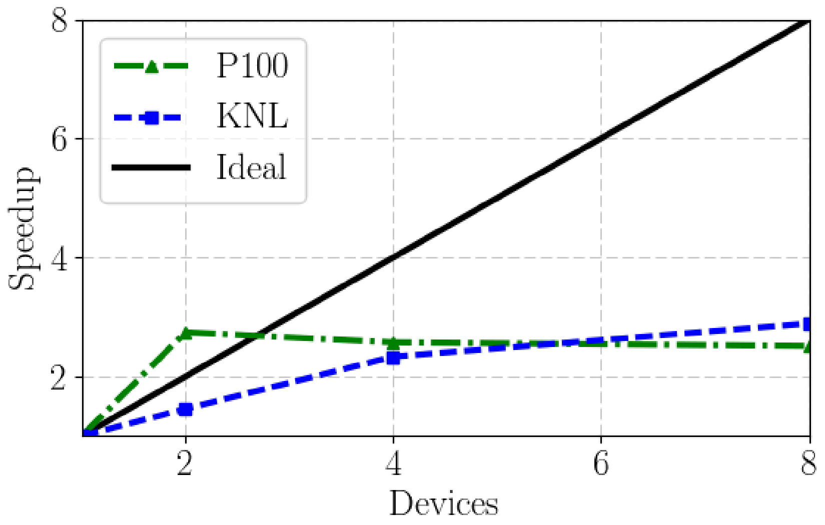
- 1.5x slowdown is in Jacobian assembly not Residual assembly → Trilinos team is investigating this now.

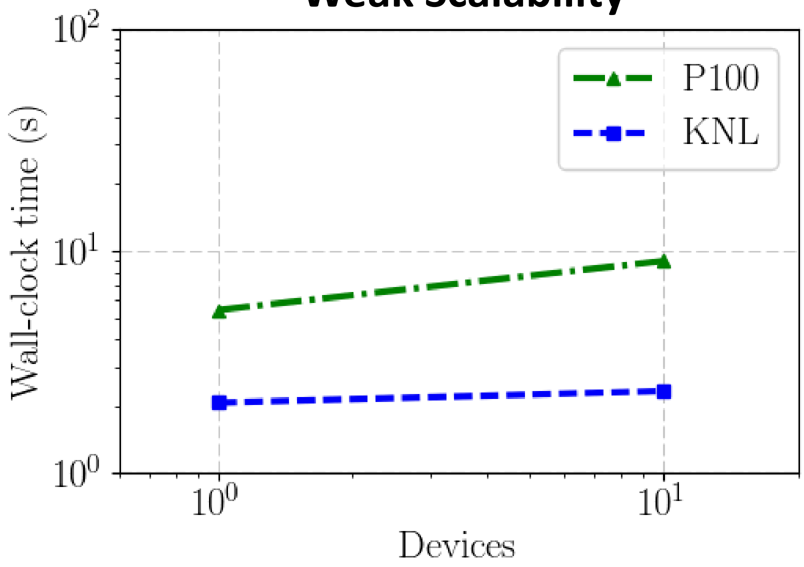
Summary: besides the global Jacobian assembly, results are promising. More studies are need once we profile nonlinear solver.

evaluateFields (blue): mostly residual/Jacobian computation + Gather/Scatter

- 1.2x speedup from hardware threads is reasonable (2x speedup expected; most likely limited by cache size in core)
- Speedup may be improved once we move towards using more OpenMP threads in nonlinear/linear solver (e.g. 4(MPI+68OMP)).
  - More OpenMP threads on cores in FEA reduces performance because it takes away from coarser grain MPI parallelism

GIS MPI+Device Scalability Study


Strong Scalability


Weak Scalability

- Scalability of **GlobalAssembly** is studied.
- **Strong Scalability** is for GIS 4km-20km tet mesh (1.51M elements).
- **Weak Scalability** is for GIS 4km-20km and 1km-7km (14.4M and 1.51M elements) tet meshes.
- KNL performs **better** in terms of scaling because of heavy use of MPI.
- KNL strong scaling likely hampered by **Jacobian assembly slow-downs**.
- P100 results hindered by **communication cost** (worse when scaling b/c no CUDA-aware MPI)
  - Scalability can likely be improved by removing CudaUVM.
- **Weak scaling** is comparable for P100 and KNL.

Outline

1. Background.
   - Motivation.
   - PISCEES project for land-ice modeling.
   - Albany/FELIX "First-Order Stokes" land-ice model.

2. Finite Element Assembly.
   - Performance-portability via Kokkos.

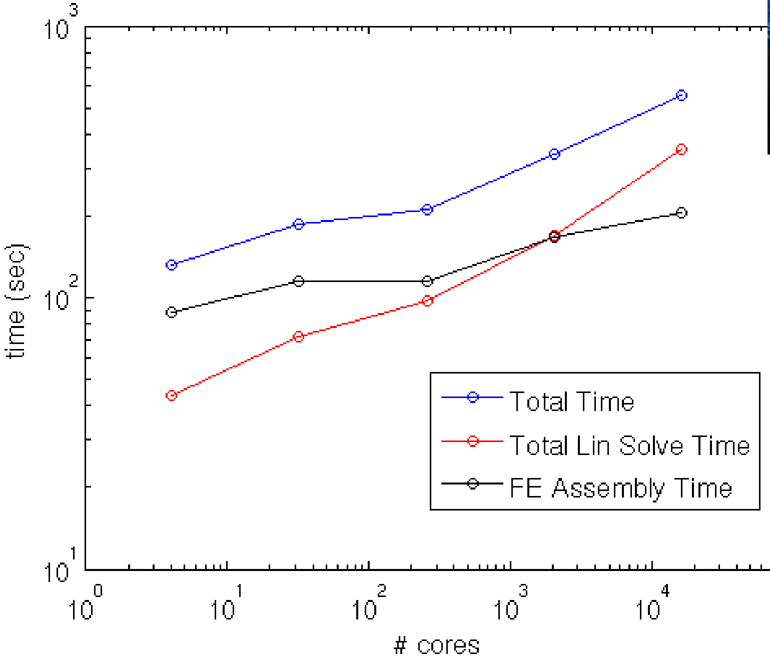
3. **Linear Solve.**
   - AMG preconditioning.

4. Summary & future work.

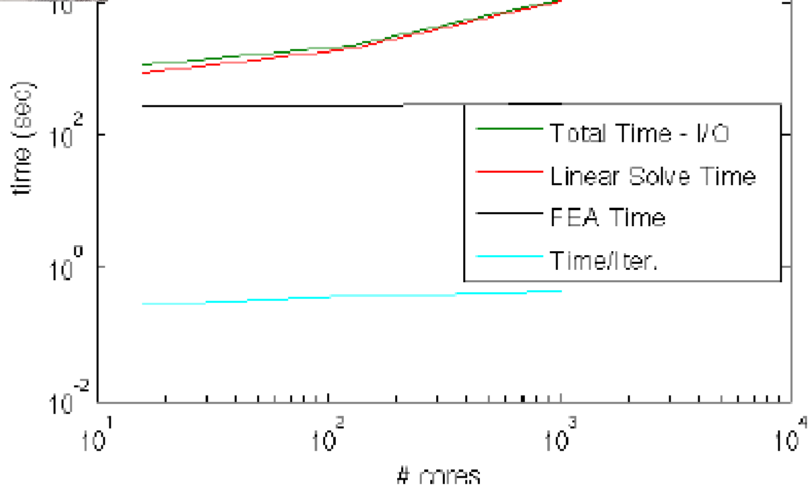
Linear Solve = 50% CPU-time

Initial Weak Scalability Study Using ILU

Greenland Ice Sheet



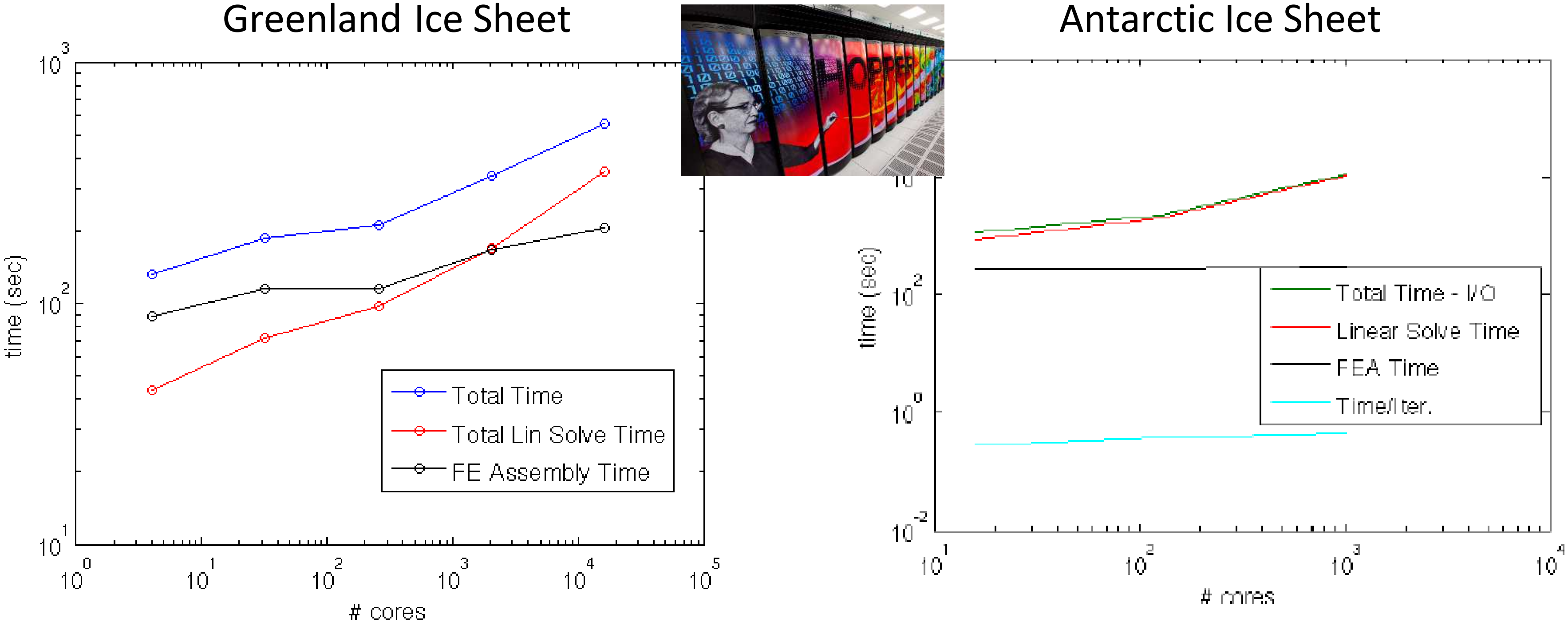
Antarctic Ice Sheet



Initial Weak Scalability Study Using ILU

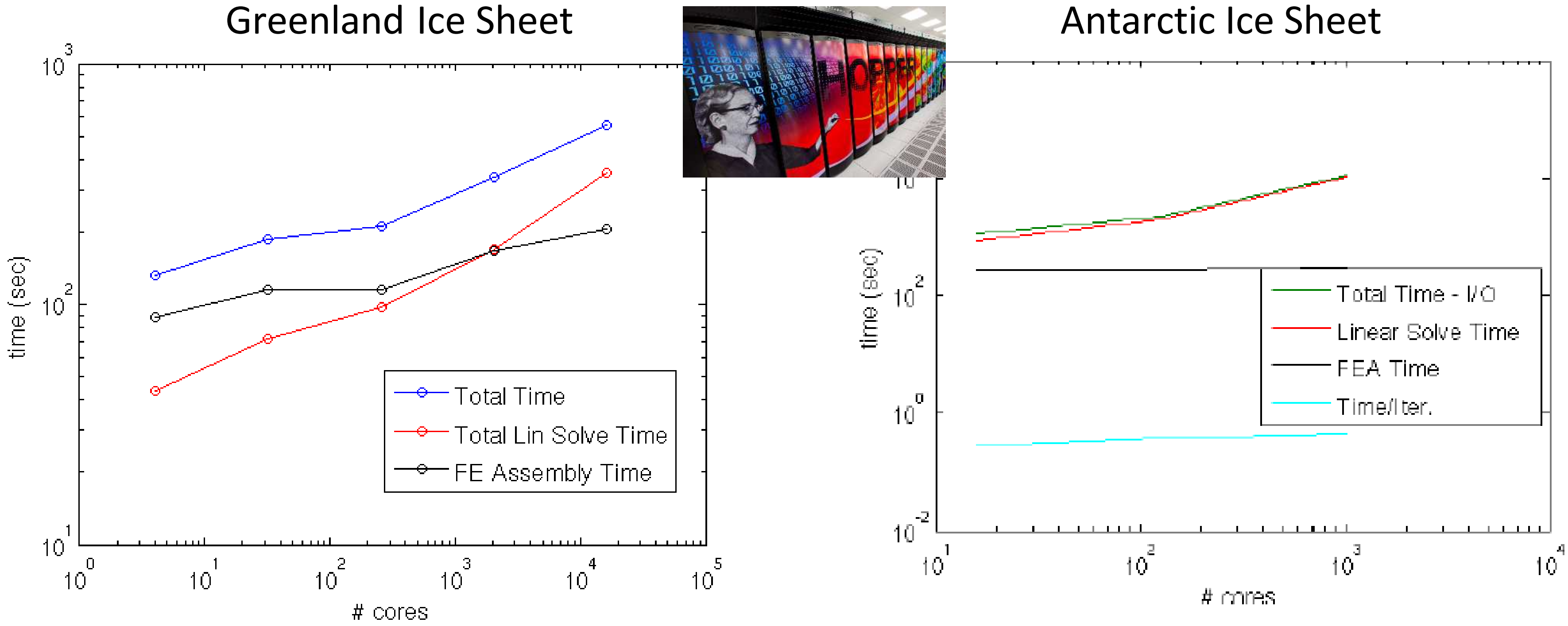
Greenland Ice Sheet

Antarctic Ice Sheet



Scalability results are ***not*** acceptable!

Initial Weak Scalability Study Using ILU



Greenland Ice Sheet

Antarctic Ice Sheet

Scalability results are ***not*** acceptable!

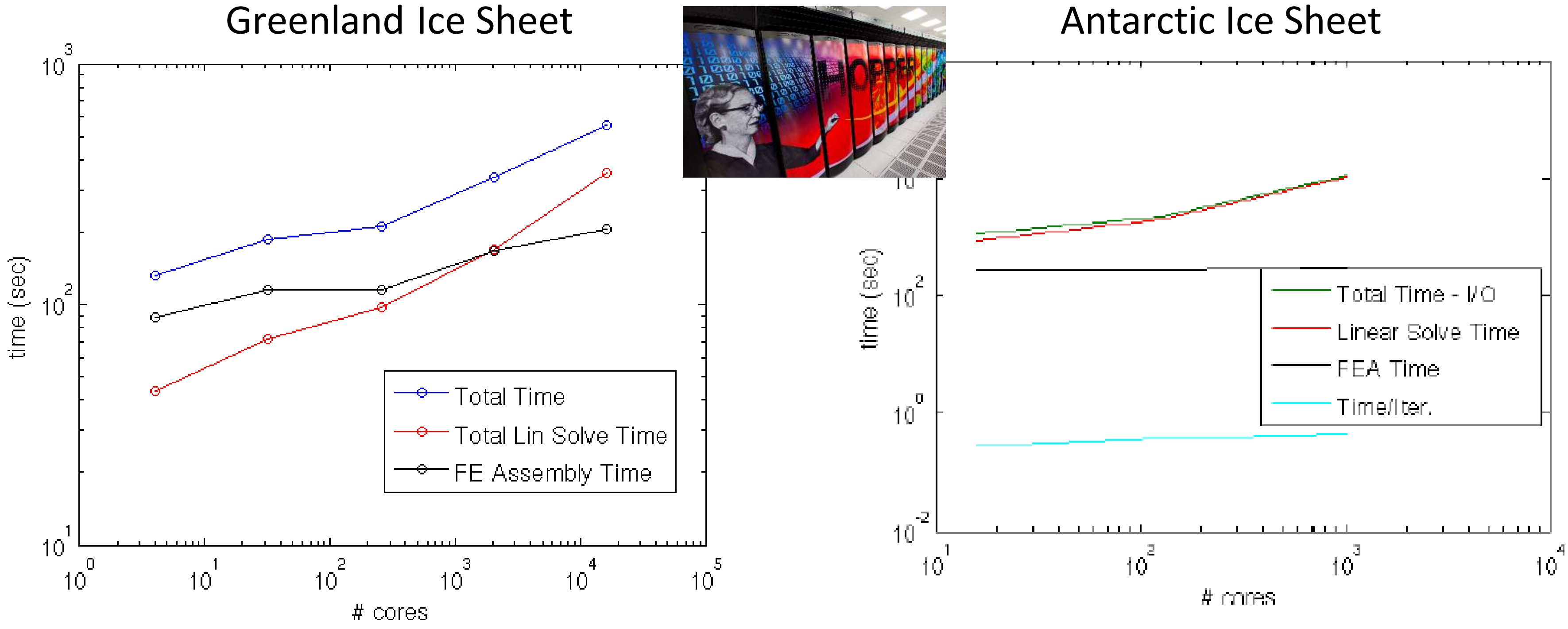
*Why is scalability so **bad** for out-of-the-box preconditioners?*

1. Ice sheet geometries have ***bad aspect ratios*** $(dx \gg dz)$.

2. ***Ice shelves*** give rise to ***severely ill-conditioned*** matrices.

3. ***Islands*** and ***hinged peninsulas*** lead to ***solver failures***.

Initial Weak Scalability Study Using ILU



Greenland Ice Sheet

Antarctic Ice Sheet

Scalability results are ***not*** acceptable!

Why is scalability so ***bad*** *for out-of-the-box preconditioners?*

1. Ice sheet geometries have ***bad aspect ratios*** $(dx \gg dz)$.
2. ***Ice shelves*** give rise to ***severely ill-conditioned*** matrices.
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.

Outline

1. Background.
   - Motivation.
   - PISCEES project for land-ice modeling.
   - Albany/FELIX “First-Order Stokes” land-ice model.

2. Finite Element Assembly.
   - Performance-portability via Kokkos.

FEA = 50% CPU-time

3. **Linear Solve.**
   - **AMG preconditioning.**

4. Summary & future work.

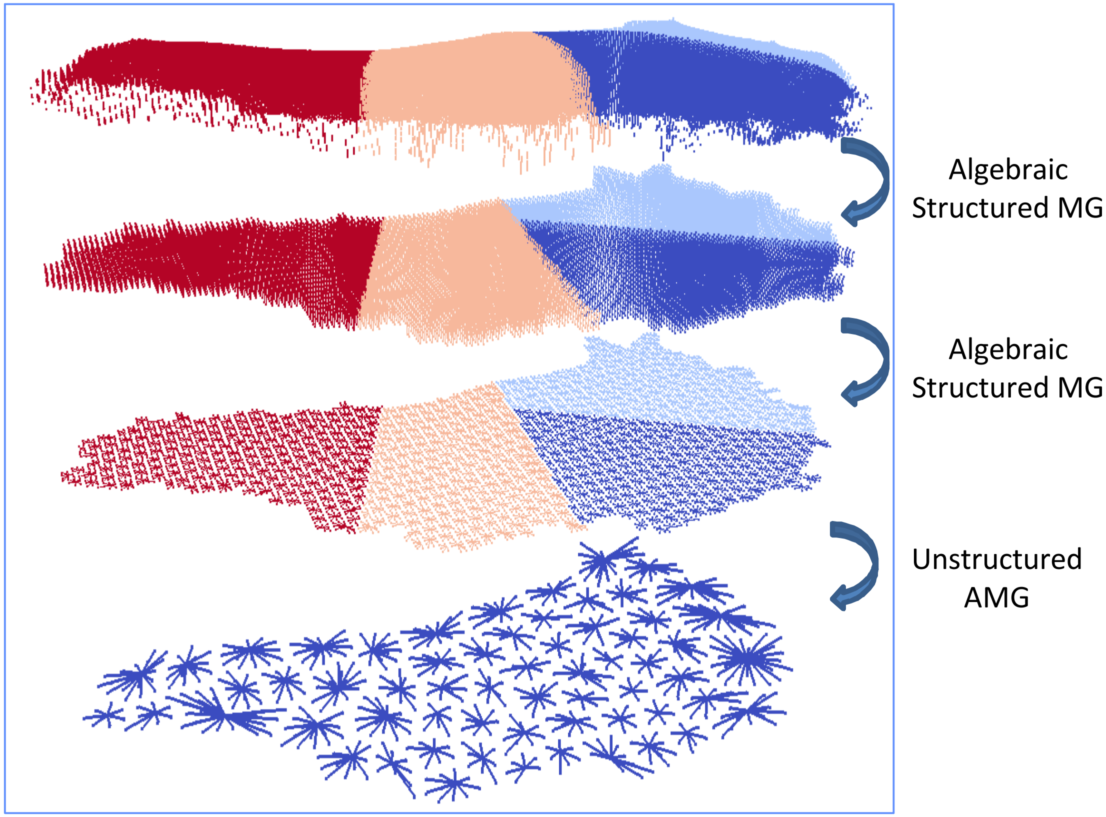
Scalability via Algebraic Multi-Grid Preconditioning with Semi-Coarsening

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*)



Algebraic Structured MG

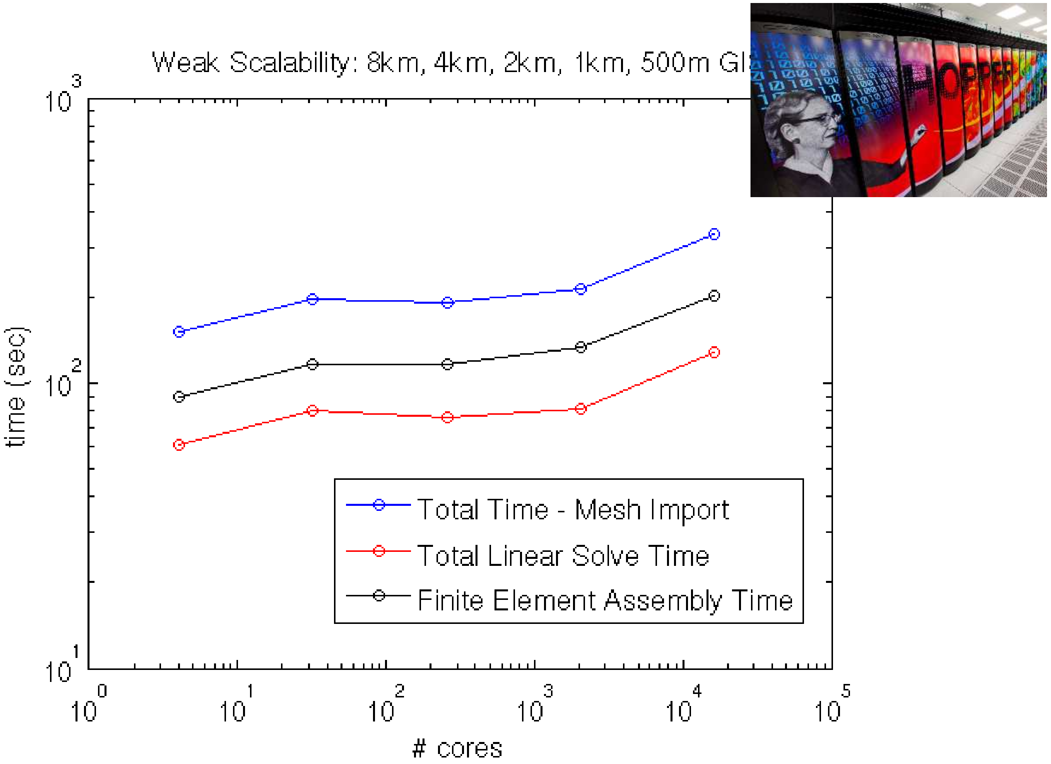
Algebraic Structured MG

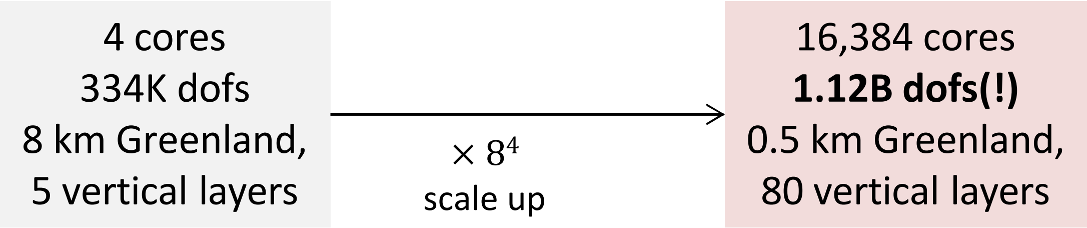
Unstructured AMG

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

Scaling studies (next slides):
New AMG preconditioner vs. ILU

Greenland Controlled Weak Scalability Study




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

× 8^4
scale up

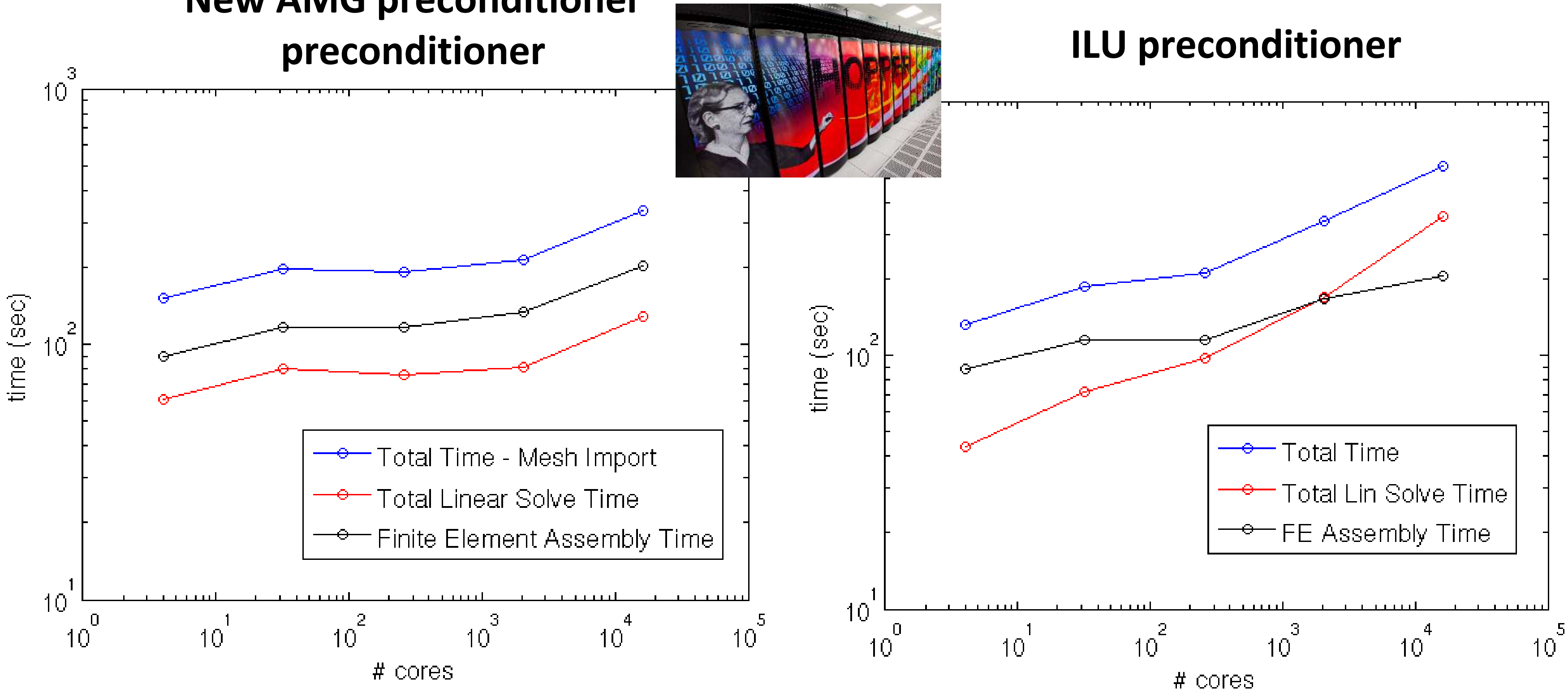
16,384 cores
1.12B dofs(!)
0.5 km Greenland,
80 vertical layers

- 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!

Greenland Controlled Weak Scalability Study

New AMG preconditioner preconditioner

ILU preconditioner



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

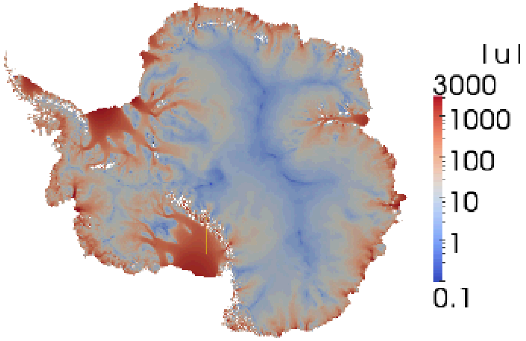
$\times\, 8^4$
scale up

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

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

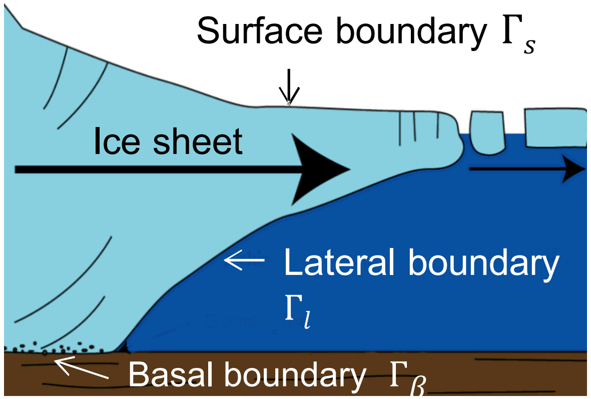
Moderate Resolution Antarctica Weak Scaling Study

Antarctica is fundamentally different than Greenland:
AIS contains large ice shelves (floating extensions of land ice).



- ***Along ice shelf front:*** open-ocean BC (Neumann).
- ***Along ice shelf base:*** zero traction BC (Neumann).

⇒ For vertical grid lines that lie within ice shelves, top and bottom BCs resemble Neumann BCs so sub-matrix associated with one of these lines is almost* singular.



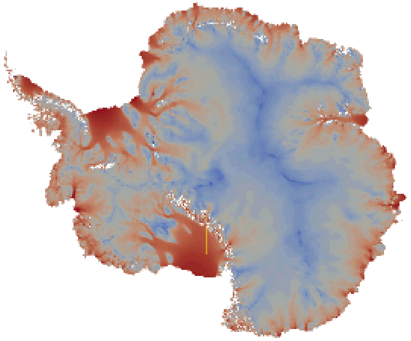
Surface boundary Γ_s
Ice sheet
Lateral boundary Γ_l
Basal boundary Γ_β

(vertical > horizontal coupling)
+
Neumann BCs
=
nearly singular submatrix associated with vertical lines

⇒ Ice shelves give rise to severe ill-conditioning of linear systems!

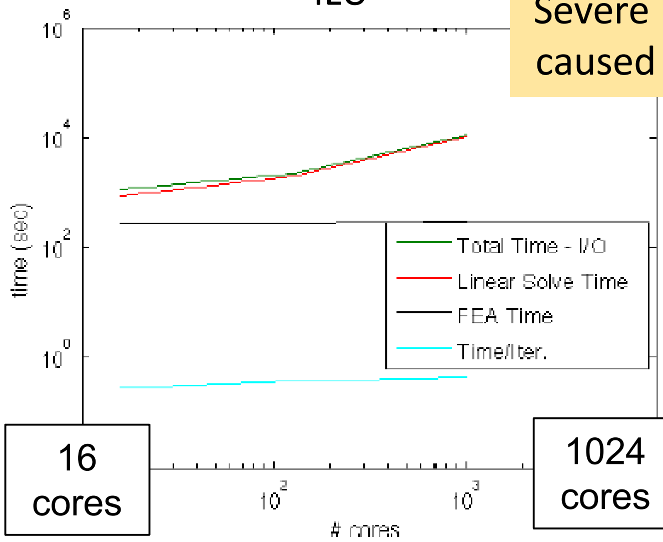
*Completely singular in the presence of islands and some ice tongues.

Moderate Resolution Antarctica Weak Scaling Study



- Weak scaling study on Antarctic problem (8km w/ 5 layers → 2km w/ 20 layers).
- 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.

ILU



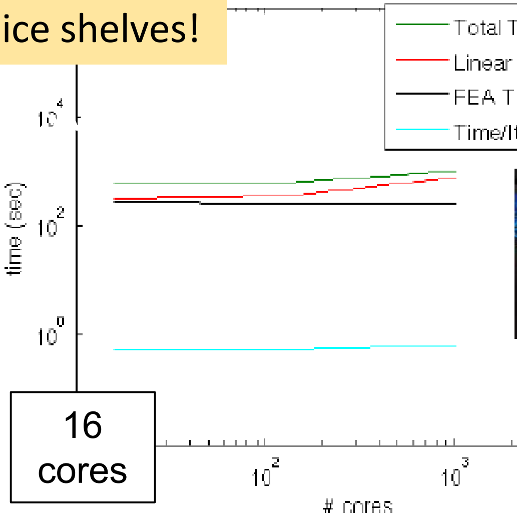
Severe ill-conditioning caused by ice shelves!

16 cores

1024 cores

# cores

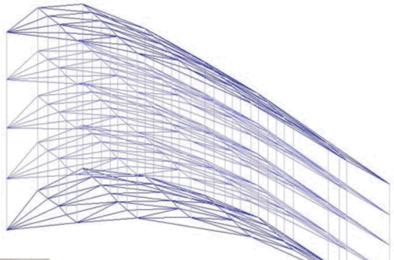
AMG



16 cores

1024 cores

# cores



AMG preconditioner

(vertical > horizontal coupling)
+
Neumann BCs
=
nearly singular submatrix associated with vertical lines

AMG preconditioner less sensitive than ILU to ill-conditioning (ice shelves → Green’s function with modest horizontal decay → ILU is less effective).

Towards Linear Solver Performance-Portability

- Jerry: please fill in info on Siva’s available preconditioners.

Outline

1. Background.
   - Motivation.
   - PISCEES project for land-ice modeling.
   - Albany/FELIX "First-Order Stokes" land-ice model.
2. Finite Element Assembly.
   - Performance-portability via Kokkos.
3. Linear Solve.
   - AMG preconditioning.
4. **Summary & future work.**

Summary & Conclusions

- A ***performance portable*** implementation of the FEA in the ***FELIX land-ice*** 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, NVIDIA GPU, Intel Xeon Phi, etc.).
  - Performance studies show that ***further optimization*** is needed to fully utilize all resources.

More on *performance-portability* of Albany using *Kokkos* can be found here: https://github.com/gahansen/Albany/wiki/Albany-performance-on-next-generation-platforms

Summary & Conclusions

- A ***performance portable*** implementation of the FEA in the ***FELIX land-ice*** 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, NVIDIA GPU, Intel Xeon Phi, etc.).
  - Performance studies show that ***further optimization*** is needed to fully utilize all resources.

More on *performance-portability* of Albany using *Kokkos* can be found here: https://github.com/gahansen/Albany/wiki/Albany-performance-on-next-generation-platforms

- ***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.

Summary & Conclusions

- A ***performance portable*** implementation of the FEA in the ***FELIX land-ice*** 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, NVIDIA GPU, Intel Xeon Phi, etc.).
  - Performance studies show that ***further optimization*** is needed to fully utilize all resources.

More on *performance-portability* of Albany using *Kokkos* can be found here: https://github.com/gahansen/Albany/wiki/Albany-performance-on-next-generation-platforms

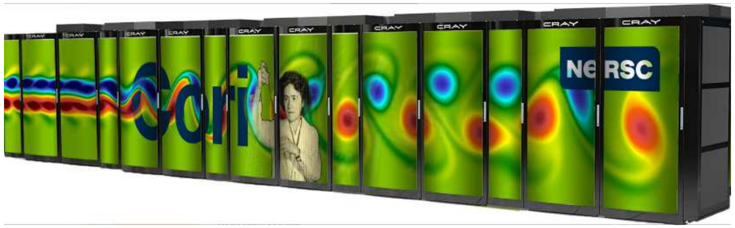
- ***Scalable, fast*** and ***robust*** linear solve is achieved via algebraic multigrid (AMG) preconditioner that takes advantage of layered nature of meshes.
  - Performance portability of linear solve is work in progress.

Heterogeneous HPC architectures can now be utilized for land-ice research using Albany/FELIX.

Ongoing/Future Work

Finite Element Assembly (FEA):

- ***Profiling*** using TAU and nvprof.
- Methods for ***improving performance***:
  - Reduce excess memory usage.
  - Utilize shared memory.
  - Replace CUDA UVM with manual memory transfer.
  - Improve performance of other sections of code besides FEA.
  - Parallelize over nodes and quadrature points in addition to cells for FELIX.
- ***Large-scale*** runs on Cori and Summit.



Linear Solve:

- Performance-portability of ***preconditioned iterative linear solve*** using *Kokkos* for implicit problems in Albany (e.g., FELIX).
  - Finish converting MueLu/Ifpack2 to use Kokkos.
  - Algorithm redesign may be necessary for GPUs.
  - Considering other solvers, e.g., hierarchical solvers, Shylu (FAST-ILU, multi-threaded GS).

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**) → **PISCEES SciDAC Application Partnership.**

PISCEES team members: *K. Evans, M. Gunzburger, M. Hoffman, C. Jackson, P. Jones, W. Lipscomb, M. Perego, S. Price, A. Salinger, I. Tezaur, R. Tuminaro, P. Worley.*

Trilinos/DAKOTA collaborators*: M. Eldred, J. Jakeman, E. Phipps, L. Swiler.*

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", submitted to *J. HPC Appl.*

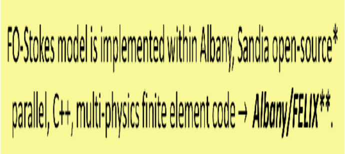
[8] S. Price, M. Hoffman, J. Bonin, T. Neumann, I. Howat, J. Guerber, **I. Tezaur**, J. Saba, J. Lanaerts, D. Chambers, W. Lipscomb, M. Perego, A. Salinger, R. Tuminaro. "An ice sheet model validation framework for the Greenland ice sheet", *Geosci. Model Dev.* 10 (2017) 255-270

Appendix: Parallelism on Modern Hardware

Year	Memory Access Time	Single Core Cycle Time
1980s	~100 ns	~100 ns
Today	~50-100 ns	~1 ns

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

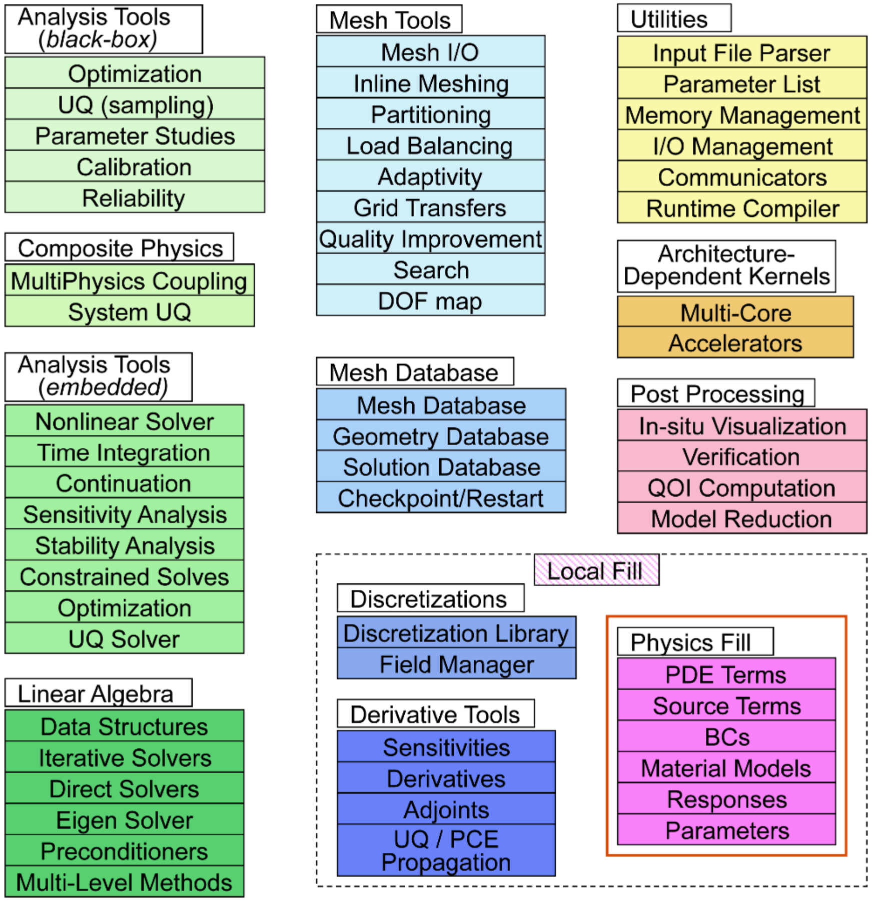
Appendix: Multiphysics Code



FO-Stokes model is implemented within Albany, Sandia open-source* parallel, C++, multi-physics finite element code → *Albany/FELIX***.

FO-Stokes model is implemented within Albany, Sandia open-source* parallel, C++, multi-physics finite element code → ***Albany/FELIX*****.

- ***Component-based*** design for rapid development of new physics & capabilities.
- Extensive use of libraries from the open-source ***Trilinos*** project:
  - Automatic differentiation.
  - Discretizations/meshes, mesh adaptivity.
  - Solvers, preconditioners.
  - Performance-portable kernels.
- ***Advanced analysis*** capabilities:
  - Parameter estimation.
  - Uncertainty quantification (DAKOTA).
  - Optimization.
  - Sensitivity analysis.



Analysis Tools (*black-box*)
- Optimization
- UQ (sampling)
- Parameter Studies
- Calibration
- Reliability

Composite Physics
- MultiPhysics Coupling
- System UQ

Analysis Tools (*embedded*)
- Nonlinear Solver
- Time Integration
- Continuation
- Sensitivity Analysis
- Stability Analysis
- Constrained Solves
- Optimization
- UQ Solver

Linear Algebra
- Data Structures
- Iterative Solvers
- Direct Solvers
- Eigen Solver
- Preconditioners
- Multi-Level Methods

Mesh Tools
- Mesh I/O
- Inline Meshing
- Partitioning
- Load Balancing
- Adaptivity
- Grid Transfers
- Quality Improvement
- Search
- DOF map

Mesh Database
- Mesh Database
- Geometry Database
- Solution Database
- Checkpoint/Restart

Utilities
- Input File Parser
- Parameter List
- Memory Management
- I/O Management
- Communicators
- Runtime Compiler

Architecture-Dependent Kernels
- Multi-Core
- Accelerators

Post Processing
- In-situ Visualization
- Verification
- QOI Computation
- Model Reduction

Local Fill

Discretizations
- Discretization Library
- Field Manager

Derivative Tools
- Sensitivities
- Derivatives
- Adjoints
- UQ / PCE Propagation

Physics Fill
- PDE Terms
- Source Terms
- BCs
- Material Models
- Responses
- Parameters

40+ packages; 120+ libraries

* https://github.com/gahansen/Albany. **Finite Elements for Land Ice eXperiments

Appendix: First-Order (FO) Stokes Model

- Ice behaves like a very ***viscous shear-thinning fluid*** (similar to lava flow).
- ***Quasi-static*** model with ***momentum balance*** given by ***"First-Order" Stokes PDEs***: "nice" elliptic approximation* to Stokes' flow equations.

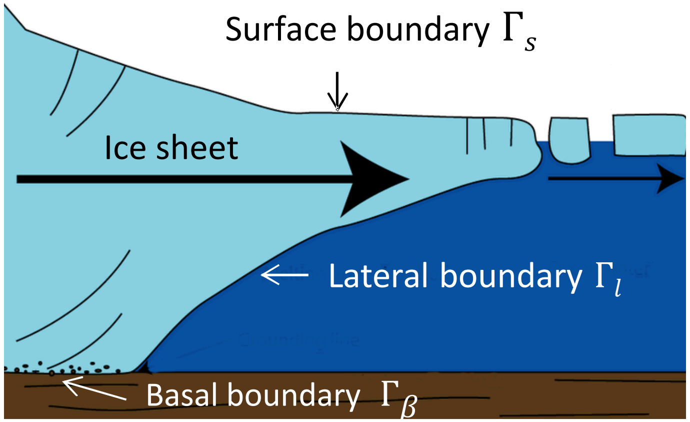
$$\begin{cases} -\nabla \cdot (2\mu \dot{\boldsymbol{\epsilon}}_1) = -\rho g \dfrac{\partial s}{\partial x} \\ -\nabla \cdot (2\mu \dot{\boldsymbol{\epsilon}}_2) = -\rho g \dfrac{\partial s}{\partial y} \end{cases}, \quad \text{in } \Omega$$

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

- Viscosity μ is nonlinear function given by "***Glen's law***":

$$\mu = \frac{1}{2} A(T)^{-\frac{1}{n}} \left(\frac{1}{2}\sum_{ij} \dot{\boldsymbol{\epsilon}}_{ij}{}^2\right)^{\left(\frac{1}{2n}-\frac{1}{2}\right)} \quad (n = 3)$$

- Relevant boundary conditions:
  - ***Stress-free BC:*** $2\mu\dot{\boldsymbol{\epsilon}}_i \cdot \boldsymbol{n} = 0$, on Γ_s
  - ***Floating ice BC:*** $2\mu\dot{\boldsymbol{\epsilon}}_i \cdot \boldsymbol{n} = \begin{cases} \rho g z \boldsymbol{n}, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$, on Γ_l
  - ***Basal sliding BC:*** $2\mu\dot{\boldsymbol{\epsilon}}_i \cdot \boldsymbol{n} + \beta(x,y)u_i = 0$, on Γ_β


Surface boundary Γ_s
Ice sheet
Lateral boundary Γ_l
Basal boundary Γ_β

$\beta(x,y) =$ basal sliding coefficient

***Assumption:** aspect ratio δ is small and normals to upper/lower surfaces are almost vertical.

Appendix: *Kokkos*-ification of Finite Element Assembly (Example)

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

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

Appendix: Ice Sheet Dynamic Equations

- Model for ***evolution of the boundaries*** (thickness evolution equation):

$$\frac{\partial H}{\partial t} = -\nabla \cdot (\bar{\boldsymbol{u}}H) + \dot{b}$$

where $\bar{\boldsymbol{u}}$ = vertically averaged velocity, $\dot{b}$ = surface mass balance (conservation of mass).

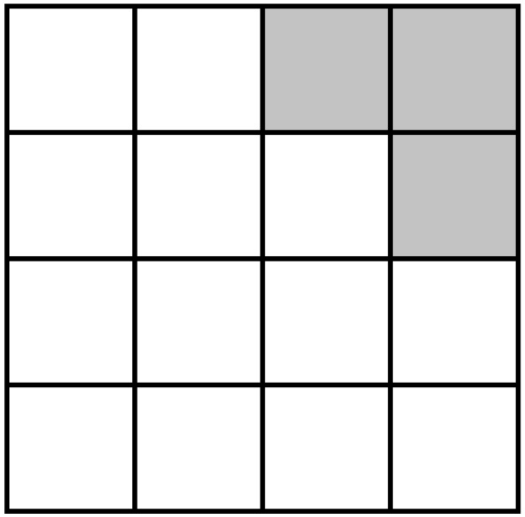
- ***Temperature equation*** (advection-diffusion):

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) - \rho c \boldsymbol{u} \cdot \nabla T + 2\dot{\boldsymbol{\epsilon}}\boldsymbol{\sigma}$$

(energy balance).

- ***Flow factor*** A in Glen's law depends on temperature T: $A = A(T)$.

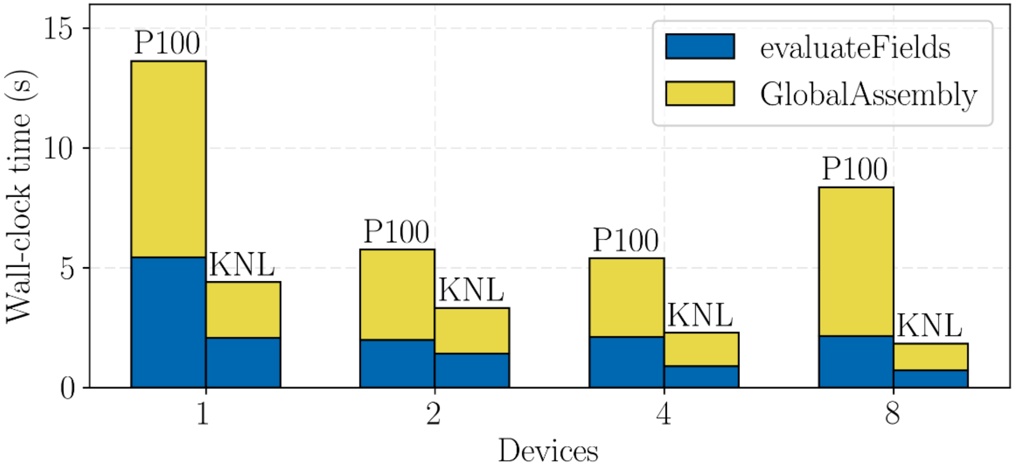
- Ice sheet ***grows/retreats*** depending on thickness H.



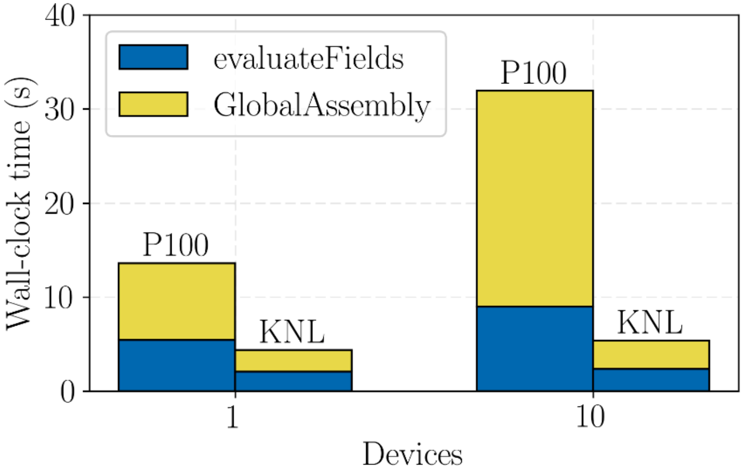
Ice-covered ("active") cells shaded in white ($H > H_{min}$)

Appendix: MPI+Device Scalability Study

Strong Scalability



Weak Scalability



Device Comparison, P100 vs. KNL (GIS 4km-20km mesh)

1. Blue: mostly Residual/Jacobian computation, Yellow: mostly communication.

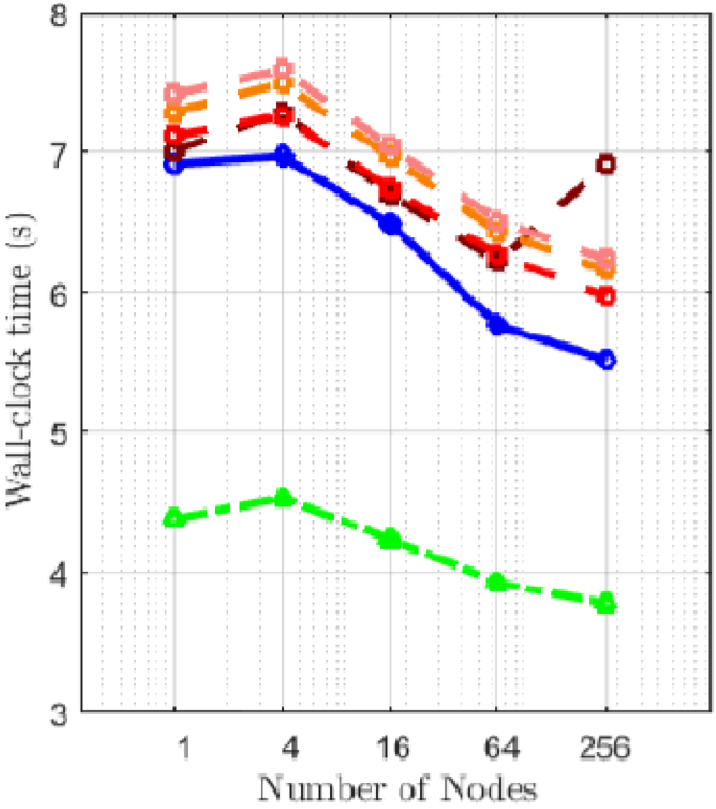
2. KNL performs better because of heavy use of MPI

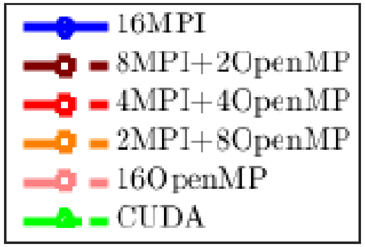
3. P100 performance is hindered by communication cost
(worse when scaling because of lack of CUDA aware MPI)

Appendix: Greenland Weak Scalability on *Titan*

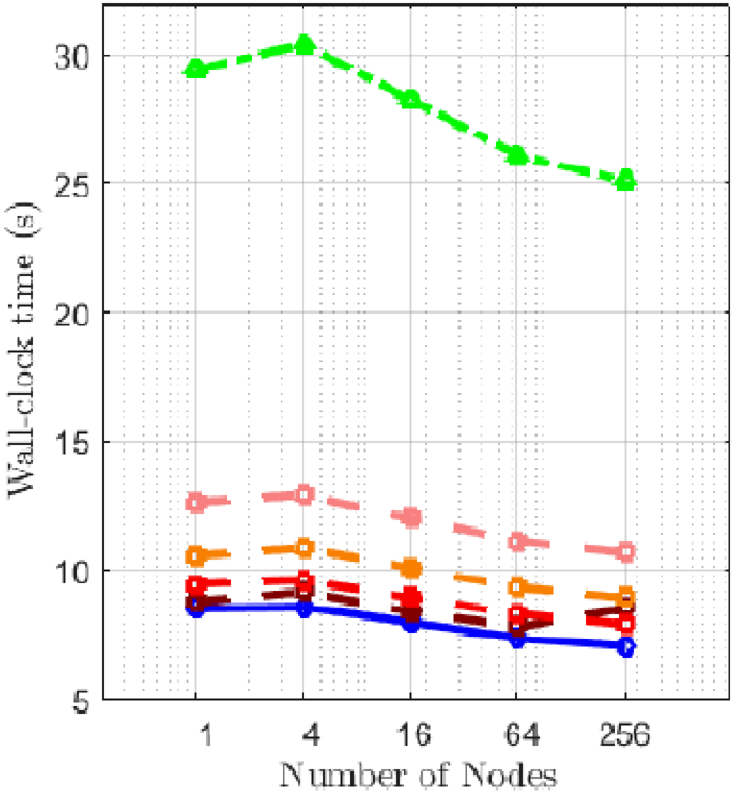
Weak scalability on *Titan* (16km, 8km, 4km, 2km, 1km Greenland)

Wall-clock Time: FEA





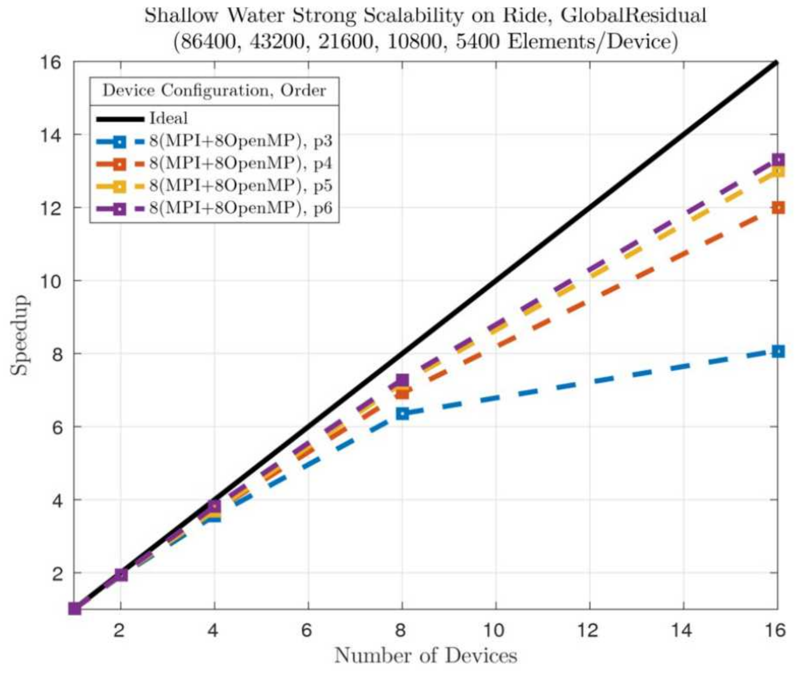
Wall-clock Time:
Total Time – Setup Time

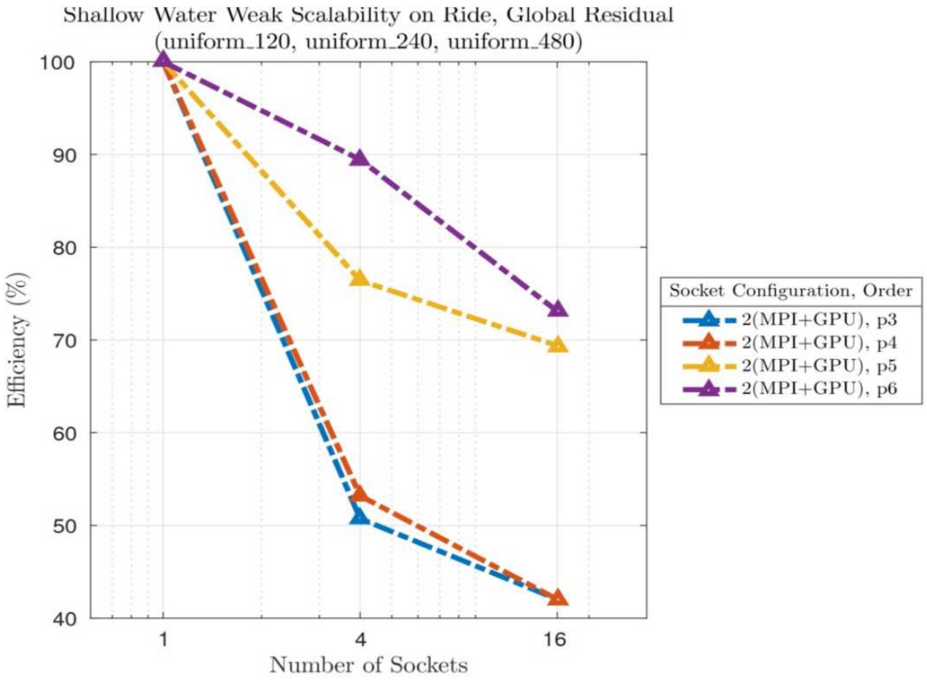


Titan: 18,688 AMD Opteron nodes

- 16 cores per node
- 1 K20X Kepler GPUs/ node
- 32GB + 6GB memory/ node

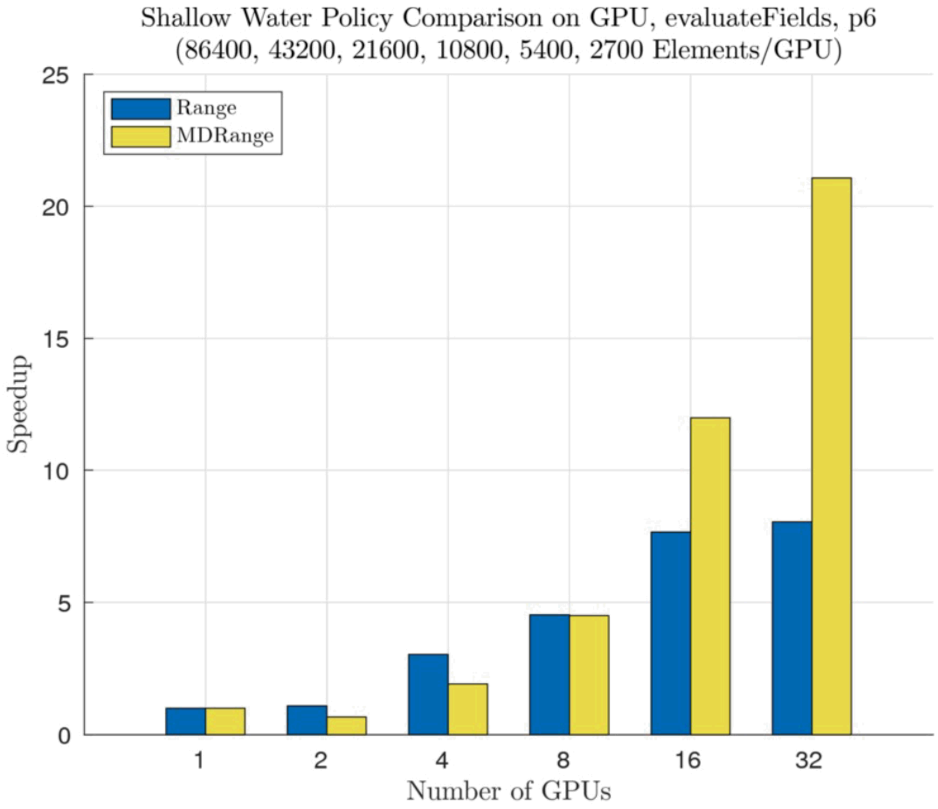
Appendix: Scalability with Increasing Order Elements





- ***Left:*** speedup plot shows benefit of using higher orders to obtain better strong scalability for MPI+OpenMP for Aeras atmosphere dycore shallow water test case.
- ***Right:*** weak scalability for MPI + GPU on the Ride for Aeras atmosphere dycore shallow water test case. Efficiency drops significantly for lower order elements, but GPU is better able to maintain weak scaling for higher order p6 spectral element.

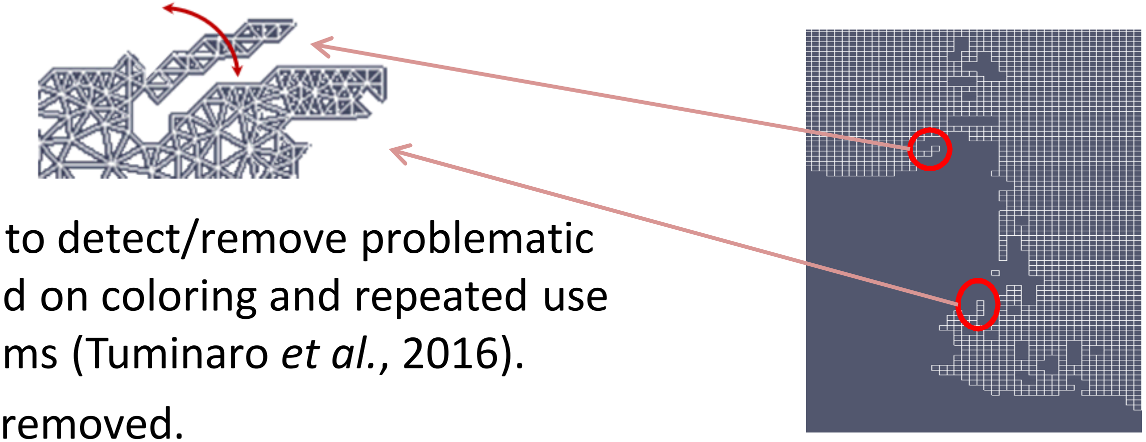
Appendix: Kokkos Range vs. MDRange Policy



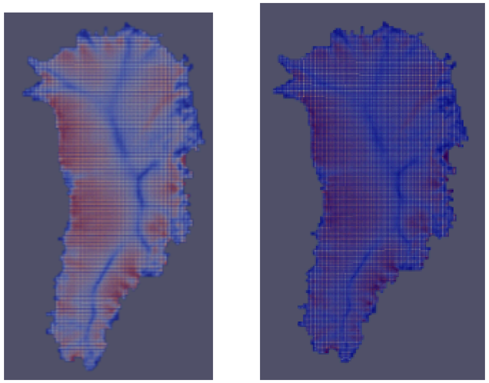
- Range vs. MDRange policy for shallow water test case in Aeras atmosphere dycore with p6 spectral element for MPI + GPU.

Appendix: Improved Linear Solver Performance through Hinge Removal

Islands and certain hinged peninsulas lead to **solver failures**



- 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.*, 2016).
- Solves are ***~2x faster*** with hinges removed.
- Current implementation is MATLAB, but working on C++ implementation 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