Performance portable ice-sheet modeling with MALI

Jerry Watkins[1,*], Max Carlson[1], Kyle Shan[2,**], Irina Tezaur[1], Mauro Perego[3], Luca Bertagna[3], Carolyn Kao[4,**], Matthew J. Hoffman[5], and Stephen F. Price[5]

[1]Sandia National Laboratories, Quantitative Modeling & Analysis Department, Livermore, CA, USA.
[2]Micron Technology, Boise, ID, USA.
[3]Sandia National Laboratories, Center for Computing Research, Albuquerque, NM, USA.
[4]TSMC, Hsinchu, Taiwan.
[5]Los Alamos National Laboratory, Fluid Dynamics and Solid Mechanics Group, Los Alamos, NM, USA.
[*]Corresponding author; email: `jwatkin@sandia.gov`.
[**]All work completed as a student at Stanford University.

April 12, 2022

Abstract

High resolution simulations of polar ice-sheets play a crucial role in the ongoing effort to develop more accurate and reliable Earth-system models for probabilistic sea-level projections. These simulations often require a massive amount of memory and computation from large supercomputing clusters to provide sufficient accuracy and resolution. The latest exascale machines poised to come online contain a diverse set of computing architectures. In an effort to avoid architecture specific programming and maintain productivity across platforms, the ice-sheet modeling code known as MALI uses high level abstractions to integrate Trilinos libraries and the Kokkos programming model for performance portable code across a variety of different architectures. In this paper, we analyze the performance portable features of MALI via a performance analysis on current CPU-based and GPU-based supercomputers. The analysis highlights performance portable improvements made in finite element assembly and multigrid preconditioning within MALI with speedups between 1.26–1.82x across CPU and GPU architectures but also identifies the need to further improve performance in software coupling and preconditioning on GPUs. We also perform a weak scalability study and show that simulations on GPU-based machines perform 1.24–1.92x faster when utilizing the GPUs. The best performance is found in finite element assembly which achieved a speedup of up to 8.65x and a weak scaling efficiency of 82.9% with GPUs. We additionally describe an automated performance testing framework developed for this code base using a changepoint detection method. The framework is used to make actionable decisions about performance within MALI. We provide several concrete examples of scenarios in which the framework has identified performance regressions, improvements, and algorithm differences over the course of two years of development.

1 Introduction

1.1 Motivation

The Greenland and Antarctic ice-sheets contain the largest reserves of fresh water on Earth and have the greatest potential to cause changes in future sea level. The Special Report on the Ocean and Cryosphere in a Changing Climate (SROCC) [42] pointed to ice-sheets as one of the dominant contributors to rising and accelerating global mean sea level and stated that sea level will continue to rise for centuries due to mass loss from ice-sheets. Projections for future sea-level rise are dependent on the ability of ice-sheet models (ISMs) to simulate ice sheet mass loss, via a wide range of processes and instabilities, but major uncertainties in ice-sheet dynamics currently exist [42, 44].

In their 2007 Fourth Assessment Report, the Intergovernmental Panel on Climate Change (IPCC) defined clear deficiencies with ISMs ability to accurately capture dynamic processes and generally did not include these models when estimating future sea level rise [57]. Ice-sheet modeling improved dramatically with progress from many community supported ice-sheet models [13, 24, 35, 63, 75] and the IPCC's Fifth Assessment Report noted increased use of ISMs in climate models; however, major uncertainties remained [22]. Since then, there have been many studies which use multiple computational models to narrow the uncertainty but increasing computational cost remains a limiting factor [20, 66, 36, 25, 47]. The SROCC noted that high resolution simulations without model simplifications are ultimately needed to obtain accurate projections of future global mean sea level [42].

The increasing fidelity and resolution of ISMs pose significant computational challenges and demand their adoption of modern software techniques. This paper focuses on performance portable methods and optimizations that can be used to improve and maintain scalable performance in the presence of changing models, software and hardware.

1.2 Performance portability

High resolution simulations of ice-sheet dynamics require a massive amount of memory and computation from large High Performance Computing (HPC) clusters, which coincidentally have undergone a dramatic change over the past decade. The current list of fastest supercomputers [55] shows a diverse set of computing architectures, which typically include processors and accelerators from a variety of vendors. Software portability across these architectures is important for productivity, as the life cycle of a code base is typically much longer than the life cycle of individual supercomputers.

Heterogeneous compute nodes are also prevalent in new systems and will continue to be a challenge for software developers as the HPC community moves towards exascale [29]. The current strategy for HPC compute nodes utilizes two architectures, CPUs and GPUs, with fundamentally different design philosophies, to achieve high performance. CPUs are designed to minimize latency and are best utilized for sequential code performance while GPUs are designed to maximize memory and computational throughput in the presence of a sufficiently large and parallelizable work load. The latest CPUs also include vector processing units which can further improve computational throughput. In scientific computing, there are often many design choices that must be considered while developing software, and heterogeneous computing adds another design variable when choosing the best algorithms. Though it may be tempting to construct a highly optimized implementation for current HPC systems, this type of software development will become increasingly harder to maintain as future models, software and hardware become increasingly more complex. This motivates the need for fundamental abstractions to be present at the application level during

code development. In response to these ongoing challenges in HPC, performance portability has grown to become crucial for simulating physical phenomena at high resolutions.

Even with the urgency of the challenge, there is still no consensus on a clear definition for the term "performance portability" [40, 50]. In general, performance portability for an application means that a reasonable level of performance is achieved across a wide variety of computing architectures with the same source code. Here, "performance" and "variety" are also admittedly subjective. In [51, 52], performance portability is quantified through efficiencies based on both application and architecture performance for a given set of platforms. In [76], this is extended to include the "Roofline" model which captures a more realistic set of empirically determined performance bounds. Since this paper focuses more on application level improvements, herein performance portability will be characterized by application execution time and scalability efficiencies for multicore/manycore processors and GPUs.

There have been a number of approaches to performance portability for application developers including directives such as OpenMP and OpenACC, and frameworks such as Kokkos [19, 72], RAJA [32] and OCCA [39]. Performance portability for finite element analysis has also been executed on a variety of different software packages including Hiflow3 [2] and Firedrake/FEniCS/PyOP2 [60, 38, 61].

As ISM codes evolve to be more robust, accurate, performant and portable on the latest HPC systems, a heavier burden is placed on software developers to support and improve functionality. Maintaining developer productivity is crucial for delivering on scientific discovery. Unfortunately, productivity is difficult to quantify, with a wide range of possible metrics [23]. For scientific software development, version control and automated testing has been challenging to integrate [34, 49], but have shown to be effective methods for improving productivity.

Automated testing becomes even more important as performance portable libraries and frameworks improve and expand their capabilities. As discussed in [50], these libraries and frameworks strive to improve developer productivity by reducing programming complexity and the need for platform-specific tuning while maintaining and improving performance. Staying up-to-date with these libraries and frameworks becomes crucial for maintaining an active code base which utilizes the latest HPC machines; however, maintaining performance portability in the presence of active development can be a difficult task. Small changes within a compiler, library, architecture, or code can cause dramatic changes to performance and performance deficiencies can be difficult to identify retroactively. Automated performance testing offers a means to improve productivity by reducing the time it takes to identify performance regressions and improvements to performance portability.

1.3   Previous related work

Traditionally for HPC, ISM codes relied solely on Message Passing Interface (MPI) libraries to achieve performance on supercomputers. MPI focuses on distributed memory parallelism, where memory may need to be communicated across multiple compute nodes. In [24], a 60% weak scalability efficiency is computed on a set of Greenland ice-sheet meshes for the full Stokes solver in Elmer/Ice using 168 to 1092 cores. The computational component of the hybrid "SSA+SIA" model in PISM is found to scale well on up to 1024 cores in [17] but the I/O component is found to scale poorly. In [21], low-overhead performance instrumentation is developed for the "Blatter-Pattyn" model in ISSM and good scaling is found on up to 3072 cores. The study finds that matrix assembly and I/O begins to scale poorly and highlights the importance of continuous performance monitoring.

A code with MPI-only is not able to take advantage of the computational throughput available on shared memory architectures including compute nodes with dedicated GPUs. GPUs can pro-

vide substantial performance improvements to existing ISMs if properly utilized. In [8], a CUDA implementation of the "iSOSIA" approximation is used to show that higher-order ice flow models can be significantly accelerated with NVIDIA GPUs. FastICE is introduced in [58] as a parallel, GPU-accelerated full Stokes solver developed in CUDA, which utilizes a matrix-free method with pseudo-transient continuation. This is extended to a portable framework written in Julia in [59] and a parallel efficiency over 96% on 2197 GPUs is achieved.

In this work, the velocity solver in MALI, which uses the "Blatter-Pattyn" model formulation, is extended to be performance portable using a multigrid preconditioned, Newton-Krvylov method where extensive improvements have been made to matrix assembly performance. The performance and scalability of MALI and its velocity solver is analyzed on multiple architectures including Intel Knights Landing (KNL) and NVIDIA V100 GPUs. The testing framework in MALI is also extended to include performance monitoring with automated detection of performance regressions and improvements using a unique changepoint detection method.

MALI (MPAS-Albany Land Ice, [31]) is an ice-sheet model built on top of two main libraries: the MPAS (Model for Prediction Across Scales) library [62], written in Fortran and used for developing variable-resolution Earth system model components, and Albany, a C++ finite element code for solving partial differential equations [64]. The performance of MALI is dominated by the solution of the first-order approximation to the Stokes equations (hereafter simply first-order velocity or first-order, see Section 2 below); hence, the performance portability efforts described in this work have been mainly targeting the C++ implementation of these equations in Albany. We note that MALI can model several additional physical processes including the ice temperature evolution, subglacial hydrology and iceberg calving [31].

Albany uses high level abstractions to integrate Trilinos libraries [28] and the Kokkos programming model [19, 72] for performance portable code across a variety of different architectures. Albany follows an "MPI+X" programming model, where MPI is used for distributed memory parallelism and the Kokkos library is used for shared memory parallelism. Kokkos provides abstractions for parallel execution and data management of shared memory computations in order to obtain optimal data layouts and hardware features, reducing the complexity of the application code. The performance portable implementation in Albany is described in detail in [16] where the authors highlight finite element assembly performance for Aeras, the atmospheric dynamical core implemented in Albany.

Albany Land Ice (ALI) is first introduced in [68] under the name Albany/FELIX. In [69], the scalability of the multigrid preconditioned, velocity solver is analyzed on up to 1024 cores. An initial study on the performance portability of the finite element assembly showed deficiencies in distributed memory assembly on GPU architectures in [74] but performance and scalability was reasonable among different architectures.

In this paper, we highlight recent improvements to finite element assembly that eliminate previous deficiencies. We also begin analyzing a new, performance portable velocity solver in ALI and expand our performance analysis to MALI.

1.4   Main contributions

The MALI code was developed in response to the growing challenges in developing a more accurate and efficient ISM [31]. In this paper, the performance portable features of MALI are introduced and analyzed on the two supercomputing clusters: NERSC Cori and OLCF Summit. A changepoint detection method is also introduced and tested for automated performance testing on next generation architectures. The main contributions of this work are summarized as follows:

- Insights into the development of a performance portable, finite element code base using high-level abstractions from Trilinos libraries and the Kokkos programming model.

- A description of new, performance-enhancing features introduced in MALI and an analysis demonstrating the expected improvements on different HPC architectures, including Intel Knights Landing (KNL) and NVIDIA V100 GPUs.

- A weak scalability study and a demonstration of speedup over CPU-only simulations.

- Insights into the development of a changepoint detection method for automated performance testing and demonstrations of tracking performance regressions, improvements, and differences between algorithms.

The methods introduced focus on improving performance portable modeling in MALI, but are extensible to other applications targeting HPC.

The remainder of this paper is organized as follows. Section 2 introduces the ice-sheet equations relevant to our analysis. Section 3 gives a detailed overview of how these equations are implemented, solved and verified in MALI. In Section 4, the methods used to achieve, improve and maintain performance portability in MALI are introduced. Lastly, Section 5 provides three numerical examples that demonstrate the expected performance of MALI on HPC systems and the utility of automated performance testing.

2　The governing ice-sheet equations

In this section, the main equations governing the ice-sheet dynamics are briefly discussed. The section begins with a description of the first-order velocity equations and is followed by a description of the mass continuity equations. More information can be found in [31, 68]. In this work, we will always assume a "topologically extruded" ice-sheet geometry, meaning that the ice-sheet geometry can be obtained by vertically extruding the two-dimensional (2D) basal area, according to the local ice thickness. A consequence of this assumption is that the margin of the ice-sheet is always vertical, though the ice thickness is typically small at grounded glacier termini.

At the ice-sheet scale, ice behaves as a highly viscous, shear-thinning, incompressible fluid and can be modeled by nonlinear Stokes equations. In this paper, a first-order approximation [18, 65] of the Stokes equations is considered, often referred to as the "Blatter-Pattyn" model [6, 43] or the "first-order" model. The model is quasi-static with static velocity (momentum balance) equations coupled to a dynamic thickness (mass) equation. In conservative form, the three-dimensional (3D), first-order velocity equations are written as,

$$
\begin{aligned}
-\nabla \cdot (2\mu_e \dot{\boldsymbol{\epsilon}}_1) + \rho g \frac{\partial s}{\partial x} &= 0, \\
-\nabla \cdot (2\mu_e \dot{\boldsymbol{\epsilon}}_2) + \rho g \frac{\partial s}{\partial y} &= 0,
\end{aligned}
\tag{1}
$$

where x, y and z are spatial coordinates bounded by the ice domain Ω, ρ is the ice density, g is gravitational acceleration and $s \equiv s(x, y)$ is the upper surface of the domain. The strain rates in Equation (1) are defined as the vectors,

$$
\begin{aligned}
\dot{\boldsymbol{\epsilon}}_1 &= [2\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy}, \dot{\epsilon}_{xy}, \dot{\epsilon}_{xz}]^T, \\
\dot{\boldsymbol{\epsilon}}_2 &= [\dot{\epsilon}_{xy}, \dot{\epsilon}_{xx} + 2\dot{\epsilon}_{yy}, \dot{\epsilon}_{yz}]^T,
\end{aligned}
\tag{2}
$$

where the components of the approximate strain rate tensor can be written as,

$$\dot{\epsilon}_{xx} = \frac{\partial u}{\partial x}, \qquad \dot{\epsilon}_{yy} = \frac{\partial v}{\partial y}, \qquad \dot{\epsilon}_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right), \qquad \dot{\epsilon}_{xz} = \frac{1}{2}\frac{\partial u}{\partial z}, \qquad \dot{\epsilon}_{yz} = \frac{1}{2}\frac{\partial v}{\partial z}, \qquad (3)$$

and u and v are the ice velocity components in the direction of x and y, respectively.

The effective viscosity, μ_e, in Equation (1) is derived from Glen's flow law [14, 41] and is written as,

$$\mu_e = \frac{1}{2}A^{-\frac{1}{n}}\dot{\epsilon}_e^{\frac{1}{n}-1}, \qquad (4)$$

where n is Glen's power law exponent and $\dot{\epsilon}_e$ is the effective strain rate, given by

$$\dot{\epsilon}_e^2 = \dot{\epsilon}_{xx}^2 + \dot{\epsilon}_{yy}^2 + \dot{\epsilon}_{xx}\dot{\epsilon}_{yy} + \dot{\epsilon}_{xy}^2 + \dot{\epsilon}_{xz}^2 + \dot{\epsilon}_{yz}^2. \qquad (5)$$

The flow law rate factor in Equation (4) is strongly temperature dependent and is determined through an Arrhenius relation,

$$A = A_0 \exp\left(-\frac{Q}{RT^*}\right), \qquad (6)$$

where A_0 is a constant of proportionality, Q is the activation energy for ice creep, T^* is the ice temperature corrected for the pressure melting point and R is the universal gas constant.

The boundary conditions are best described by partitioning the surface of the 3D ice-sheet domain into upper, lower and lateral surfaces,

$$\Gamma = \Gamma_s \cup \Gamma_\beta \cup \Gamma_l, \qquad (7)$$

where Γ_s is the upper surface, Γ_β is the lower surface and Γ_l is the lateral surface. The boundary conditions can then be defined as:

1. a homogeneous boundary condition on the upper surface (atmosphere pressure is neglected),

$$\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} = 0, \qquad \dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} = 0, \qquad \text{on } \Gamma_s, \qquad (8)$$

   where $\boldsymbol{n}$ is the outwards facing normal vector;

2. a Robin boundary condition on the lower surface, representing a linear sliding law at the bed,

$$2\mu_e\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} + \beta u, \qquad 2\mu_e\dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} + \beta v, \qquad \text{on } \Gamma_\beta, \qquad (9)$$

   where the basal sliding coefficient $\beta \equiv \beta(x, y)$ is non-negative where the ice is grounded and zero where the ice is floating;

3. a dynamic Neumann boundary condition at the ice margin accounting for the back pressure from the ocean where the ice is submerged (note that by convention $z = 0$ represents the sea level),

$$\begin{aligned}
2\mu_e\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} - \rho g(s - z)\boldsymbol{n} &= \rho_w g \max(z, 0)\boldsymbol{n}, \\
2\mu_e\dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} - \rho g(s - z)\boldsymbol{n} &= \rho_w g \max(z, 0)\boldsymbol{n}, \qquad \text{on } \Gamma_l,
\end{aligned} \qquad (10)$$

   where ρ_w is the density of water and z is the elevation above sea level.

The steady velocity equations described above are coupled to a dynamic equation for the conservation of mass. Specifically, as the ice-sheet evolves in time, mass continuity is enforced through the following equation,

$$\frac{\partial H}{\partial t} + \nabla \cdot (H\bar{\boldsymbol{u}}) = \dot{a} + \dot{b}, \qquad (11)$$

where H is ice thickness, t is time, $\bar{\boldsymbol{u}}$ is depth-averaged velocity vector, $\dot{a}$ is surface mass balance and $\dot{b}$ is basal mass balance. The thickness equation is then used to evolve the geometry in time. The ice temperature is held constant in time.

3   Implementation in MALI

In this section we describe how the governing equations introduced in Section 2 are discretized and implemented in MALI, focusing in particular on the C++ velocity solver model in ALI. We first give a high-level overview of the implementation and then provide a detailed description for the two more computationally expensive components relevant to the paper: the finite element assembly and the preconditioned linear solver. A brief description of the MALI testing framework follows.

3.1   Overview

The ice thickness H in Equation (11) is discretized in MPAS with a upwind finite volume method on an unstructured, two-dimensional, Voronoi grid [31]. At every time step, the first-order velocity equations (Section 2) are solved in ALI [68]. Briefly, the equations are discretized using low-order nodal prismatic finite elements on a 3D mesh extruded from a triangulation dual to the MPAS Voronoi mesh. The discrete version of the velocity equations can be written in the compact form

$$\mathcal{F}(U;\ \{\phi_i\}, \{\nabla\phi_i\}, H, \beta, \ldots) = 0. \tag{12}$$

Here U is the solution vector, containing the values of ice velocity at the mesh nodes. $\{\phi_i\}$ and $\{\nabla\phi_i\}$ are the sets of the basis functions and their gradients. $\mathcal{F}$ is a vector function of the solution U. $\mathcal{F}$ also depends on the basis functions and on fields like ice thickness H and basal friction β. We refer to $\mathcal{F}(U;\ \cdot)$ as the *residual*.

A damped Newton's method is used to solve the nonlinear discrete system (12):

$$\mathcal{J}_{\mathcal{F}}^k\, \delta_U^{k+1} = -\mathcal{F}(U^k), \qquad U^{k+1} = U^k + \alpha_k\, \delta_U^{k+1}. \tag{13}$$

Here $\mathcal{J}_{\mathcal{F}}^k := \dfrac{\partial\mathcal{F}}{\partial U}\bigg|_{U=U^k}$ is the *Jacobian* matrix and α_k is the damping factor. At each nonlinear iteration k, the linear system (13) is solved with the GMRES method using the "matrix-dependent semicoarsening-algebraic multigrid" (MDSC-AMG) [73] preconditioner.

Figure 1 shows a flow chart of ice-sheet dynamics in MALI, focusing on high-level components in the velocity solver in ALI. Each node of the flow chart is described below with references to relevant Trilinos packages:

- **Import**: Imports the ice velocity solution, U, from a nonoverlapping data structure where each MPI rank owns a unique part of the solution to an overlapping data structure where some data exists on multiple ranks. This gives each rank access to relevant solution data without any further communication. This is performed by Trilinos/Tpetra [3].

- **Gather**: Gathers solution values from an overlapping data structure to an element local data structure where data is indexed according to element and local node. This process also includes the gathering of geometry and field data from MPAS. This is constructed using Trilinos/Phalanx [45, 46, 64] and Trilinos/Kokkos [19, 72].

- **Interpolate**: Interpolates the solution and solution gradient from nodal points to quadrature points. Other field variables also require interpolation. This is constructed using Trilinos/Phalanx and Trilinos/Kokkos.

- **Evaluate**: Evaluates the residual, Jacobian and source terms of the first-order equations. These operators are templated in order to take advantage of automatic differentiation for analytical Jacobians using Trilinos/Sacado [54]. This also uses Trilinos/Phalanx and Trilinos/Kokkos.
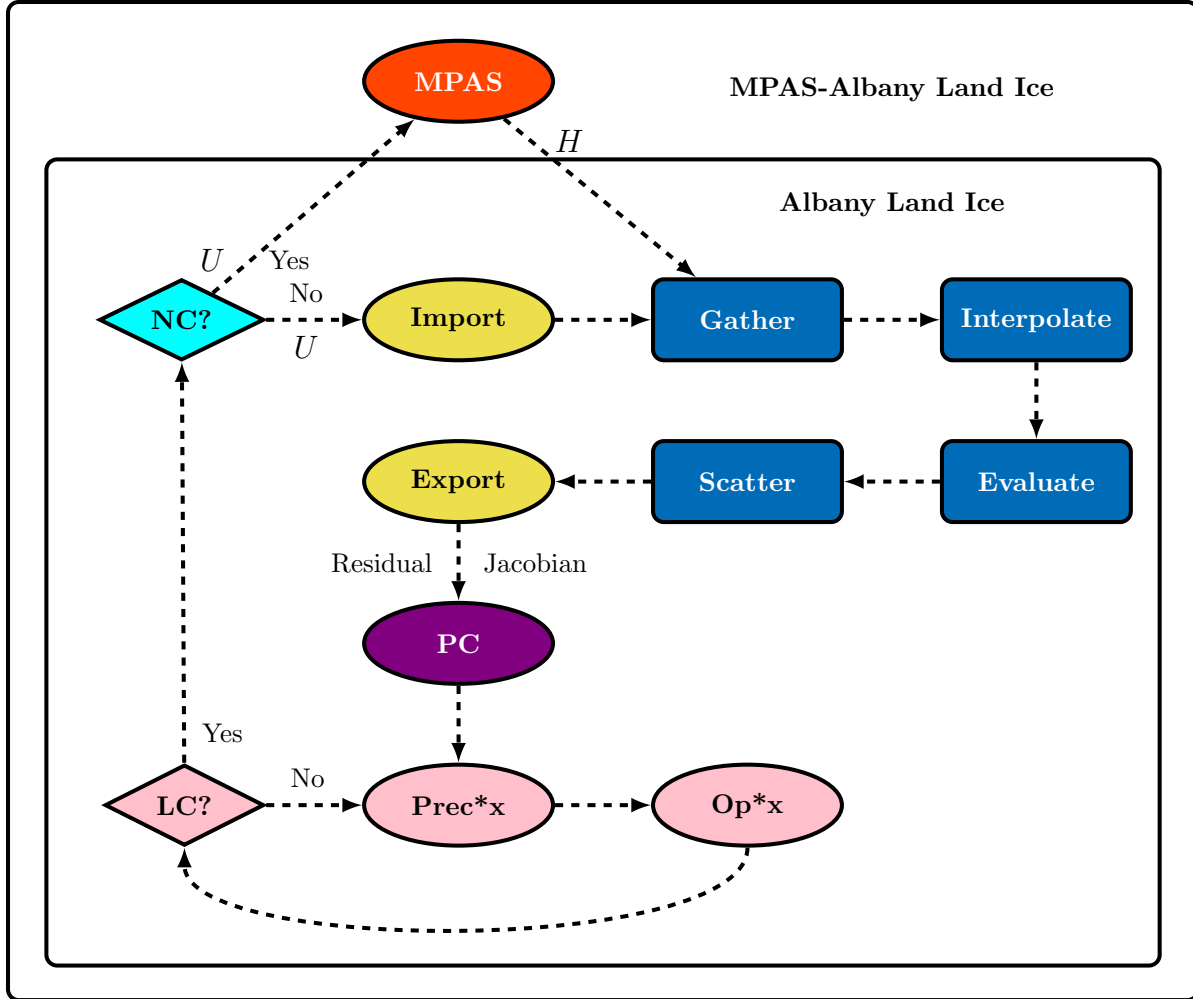
Figure 1: A flow chart depicting the workflow in MALI, focusing on high-level components in the velocity solver in ALI. The shapes of the nodes are constructed to show performance relevant characteristics of the code while the colors are used to differentiate high-level abstractions. In this case, ellipses represent portions of code with MPI communication and rounded rectangles represent portions of code which do not require MPI communication. The conditionals, LC (Linear solver converged) and NC (nonlinear solver converged), are represented with diamonds. The red-orange node represents the computation required for explicit time stepping of ice thickness, H, in MPAS, the purple node represents preconditioner construction (PC), and the pink nodes represent the linear solver required to solve (13). Finite element assembly begins and ends with distributed memory assembly colored in yellow but also performs shared memory processes colored in blue.

- **Scatter**: Scatters residual and Jacobian values from an element local data structure to an overlapping data structure. This is constructed using Trilinos/Phalanx and Trilinos/Kokkos.

- **Export**: Exports the residual and Jacobian from an overlapping data structure to a nonoverlapping data structure where information is updated across MPI ranks. This global structure allows for efficient use of linear solvers and is performed by Trilinos/Tpetra.

- **PC (Preconditioner Construction)**: Constructs the MDSC-AMG preconditioner from the Jacobian matrix and is performed by Trilinos/MueLu [5] and Trilinos/Ifpack2 [56].

- **Prec*x**: Applies the preconditioner to the solution vector of the linear system and is performed by Trilinos/Belos [4], Trilinos/MueLu, Trilinos/Ifpack2.

- **Op*x**: Applies the Jacobian matrix to the solution vector of the linear system and is performed by Trilinos/Belos.

- **LC? (Linear Solver Converged?)**: The linear solver loop is converged when a fixed linear tolerance or a maximum number of iterations is reached. This is managed by Trilinos/Belos.

- **NC? (Nonlinear Solver Converged?)**: The nonlinear solver loop is converged when a fixed nonlinear tolerance is reached. This is managed by Trilinos/NOX [70].

- **MPAS**: Once the ice velocity, U, has fully converged, it is interpolated to MPAS cell edges and used to update H using forward Euler. The new ice-sheet geometry is passed back into Albany Land Ice to re-compute U for the next time step. This process is performed until a final time step is reached. A more detailed description of thickness evolution in MPAS can be found in [31].

By constructing high-level abstractions for solving nonlinear partial differential equations and utilizing Trilinos packages as components, application developers are able to apply existing performance portable algorithms which are actively supported and improved by experts. Developers are also able to utilize the same algorithms on multiple applications, allowing for greater impact and increased sustainability.

3.2 Finite element assembly

The finite element approach implemented in Albany is designed to easily incorporate multiple physics models with graph-based evaluation using the Trilinos/Phalanx package [45, 46, 64]. The assembly is decomposed into a set of nodes called "evaluators". Evaluators have a specified set of inputs and outputs and are organized in a directed acyclic graph (DAG) based on dependencies. Figure 2 shows a simplified example of a DAG for the for finite element assembly in Albany. The inherent advantage of using a DAG is the increased flexibility, extensibility and usability of using modular evaluators when performing finite element assembly. The DAG also provides potential for task parallelism. The disadvantage of using a DAG is that there is a potential for performance loss through code fragmentation and a static graph can also lead to repetition of unneeded data movement and computation. This is discussed in more detail in Section 4.1.

Albany utilizes automatic differentiation to compute the Jacobian matrix and enable Newton-like methods for the solution of nonlinear partial differential equations. More in general, the finite element assembly in Albany is designed for embedded analysis using template-based generic programming [45, 46, 64]. Embedded analysis, such as derivative-based optimization and sensitivity
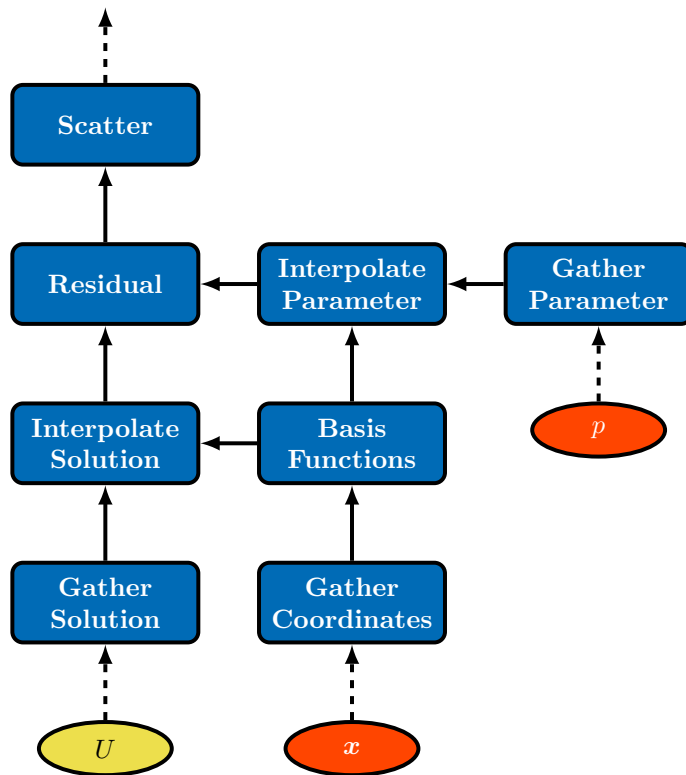
Figure 2: Albany uses a directed acyclic graph (DAG) for finite element assembly. In this example, a global residual is constructed by interpolating the solution and a parameter onto quadrature points. Basis functions are also needed to complete the interpolation which depend on physical coordinates. In the case of the first-order equations, the solution is the ice velocity. One example of a parameter could be the surface height.

analysis, for partial differential equations requires construction of mathematical objects such as Jacobians, Hessian-vector products and derivatives with respect to parameters. Albany utilizes C++ templates and operator overloading to perform automatic differentiation using the Trilinos/Sacado package [54]. Sacado provides multiple data types for storing the derivative components, each with their own relative advantages and disadvantages. The `DFad` options sets the number of derivative components at runtime, and is hence the most flexible but least efficient option. The `SLFad` options sets the maximum number of derivative components at compile time, making this option both flexible and relatively efficient. The most efficient but least flexible option is `SFad`. For this option, the number of derivative components is set at compile time. In MALI, we generally select the `SFad` type whenever possible, so as to achieve the best possible performance. The difference in performance with the various options is most profound in a GPU run, due to the substantial cost of performing dynamic allocation on the GPU.

Finite element evaluators also contain Kokkos parallel execution kernels for performance portability on shared memory architectures [16, 74]. Kokkos utilizes memory and execution spaces to determine where memory is stored and where code is executed. Phalanx evaluators utilize `MDField` with Kokkos `View` for memory management and evaluators can be used as a Kokkos functor to perform parallel operations. Sacado operators have also been designed to work with Kokkos. Figure 3 shows a simplified example of an evaluator with Kokkos.

```
1  template<typename EvalT, typename Traits>
2  void StokesFOResid<EvalT, Traits>::
3  evaluateFields(typename Traits::EvalData workset) {
4    Kokkos::parallel_for(
5        Kokkos::RangePolicy<ExeSpace>(0,workset.numCells),
6        *this);
7  }
8
9  template <typename EvalT, typename Traits>
10 KOKKOS_INLINE_FUNCTION
11 void StokesFOResid<EvalT, Traits>::
12 operator() (const int& cell) const {
13   for (int node = 0; node < numNodes; ++node) {
14     Residual(cell,node,0) = 0.0;
15     for (int qp = 0; qp < numQPs; ++qp) {
16       Residual(cell,node,0) +=
17           Ugrad(cell,qp,0,0)*wGradBF(cell,node,qp,0) +
18           Ugrad(cell,qp,0,1)*wGradBF(cell,node,qp,1) +
19           force(cell,qp,0)*wBF(cell,node,qp);
20     }
21   }
22 }
```

Figure 3: The first-order local residual computation is performed in a Phalanx evaluator that uses Kokkos for shared memory parallelism and Sacado for automatic differentiation. This is a simplified example of the computation of a single term in the residual. When called with a `double` `EvalT` type, the routine returns the residual; when called with a `Fad` `EvalT` type, automatic differentiation is applied and a Jacobian is returned.

The Phalanx evaluation type is passed via the template parameter `EvalT` and dictates whether a residual with a `double` type or a Jacobian with a `Fad` type is computed. A Kokkos `RangePolicy` is used to parallelize over cells over an execution space, `ExeSpace`. In this paper, only the `Serial` and `Cuda` executions spaces are used to differentiate between CPU and GPU execution but other

execution spaces are also available. The properties of each case are described in more detail in [16, 74]

3.3 Preconditioner for linear solve

A primary challenge in simulating ice-sheets at scale is solving the linear system associated with a thin, high-aspect ratio mesh. It has been shown [10, 33, 73] that multigrid methods can be used to address the challenges arising due to the anisotropic nature of the problem, although alternative methods have been proposed [12, 27]. The performance of the linear solver is thus primarily dictated by the efficacy of the multigrid preconditioner. The MDSC-AMG preconditioner introduced in [73] is specifically designed for ice-sheet meshes where a mesh is first constructed from 2D topological data and extruded in the vertical direction to construct a 3D mesh. The primary strategy for the multigrid hierarchy is to coarsen the fine mesh in the vertical direction until a single layer is reached and apply smoothed aggregation algebraic multigrid (SA-AMG) on the plane. Here, we are able to take advantage of the performance portable implementations of SA-AMG and point smoothers implemented in Trilinos/MueLu and Trilinos/Ifpack2.

3.4 Testing

Software quality tools are a central part of the Albany code base and are crucial for developer productivity [64]. Rather than using a fixed release of Trilinos, ALI is designed to stay up-to-date with Trilinos' version of the day, to ensure that the code inherits the most up-to-date additions and improvements to Trilinos. This requires a close collaboration between Albany and Trilinos developers and ensures rapid response to issues that might arise. The current nightly test harness includes unit, regression and performance tests on Intel and IBM multicore CPUs and NVIDIA GPUs and is monitored on a dedicated dashboard.

4 Methods for improving and maintaining performance portability

In this section, we discuss the major enhancements made to MALI to both improve and maintain performance portability. We provide three examples of finite element assembly optimizations, which improved performance on both CPU and GPU systems. Memoization is utilized to avoid unnecessary data movement and computation from the MALI workflow. Optimizations in matrix assembly and boundary condition computation led to significant speedups on both CPU and GPU and a large reduction in memory usage. We also provide a brief description of a new, performance portable MDSC-AMG preconditioner implemented in Trilinos/MueLu and tuned for ice-sheet modeling. Lastly, we provide a description of an automated performance testing framework for identifying regressions, improvements and performance differences between algorithms.

4.1 Memoization

A static DAG similar to the one shown in Figure 2 is executed when a new global residual or Jacobian is needed within the nonlinear solver. This leads to a repetition of unnecessary data movement and computation when input quantities do not change between calls of the DAG. A performance gain can be achieved by storing the results of expensive nodes in the DAG and returning the stored results when input quantities do not change. This process is known as memoization.

In Albany, memoization is implemented by constructing a new DAG which only follows changes caused by the solution. Figure 4 shows an example of the new DAG created by performing memoization in Figure 2. The first call executes the original DAG while storing all intermediate quantities.
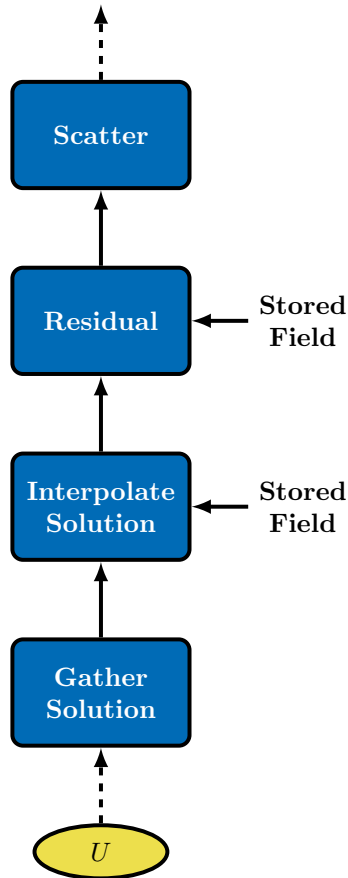

Figure 4: Albany uses memoization to create a new DAG which only depends on changes to the solution. This avoids unnecessary data movement and computations when parameters and coordinates do not change. For examples, in the case of first-order equations, the surface height does not change at each nonlinear iteration.

Then, the new DAG is called by default while the original DAG is only called when there is a change to a parameter or the coordinates. An initial speedup of roughly 1.4 on CPUs and 1.5 on GPUs was found when analyzing finite element assembly performance relative to the assembly without memoization.

4.2   Jacobian matrix

As discussed in Section 3, the finite element assembly of residual and Jacobian is performed on an "overlapped" distribution of the DOFs, while linear systems require a "unique" distribution of DOFs.

Until recently, Albany was using two separate Tpetra `CrsMatrix` objects for the Jacobian: an overlapped version for finite element assembly, and a non-overlapped version for linear solvers. An

Export operation (involving MPI communication) was used to copy data between the overlapped and the unique matrices, migrating off-processor rows to their owner.

We improved this portion of the library by switching to the new Tpetra `FeCrsMatrix` objects, which can store overlapped and non-overlapped matrix in a single object, by storing the "owned" rows first, followed by the off-processor ones. This arrangement allows to build the non-overlapped matrix as a "subview" of the overlapped one. The benefit is twofold: the memory footprint for the Jacobian is roughly halved, plus no copy is needed to transfer data for the local rows from the overlapped Jacobian to the non-overlapped Jacobian. This translated to a speedup of roughly 1.1 on CPUs and 2.1 on GPUs when analyzing finite element assembly performance relative to the old implementation.

4.3   Boundary conditions

In order to achieve high performance on GPUs, fields corresponding to boundary data needed to be aligned for coalesced access on the device. The process is described in detail in [11] for a different problem and is summarized in Figure 5. Originally, boundary data were stored using the same
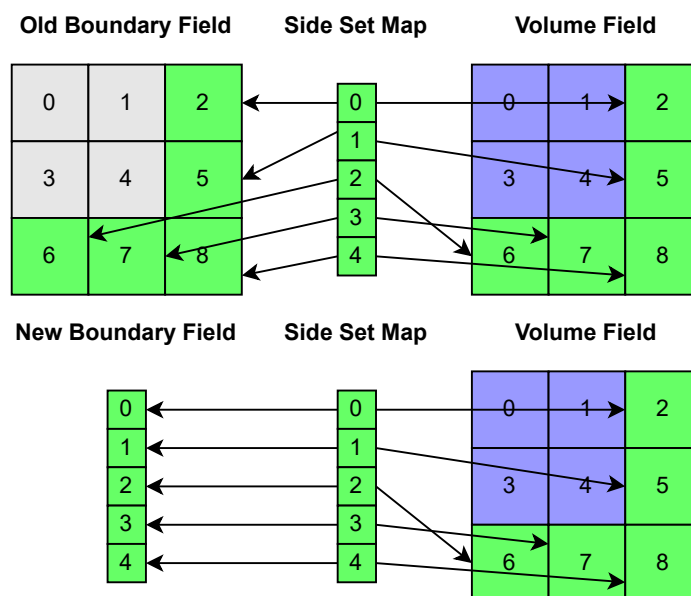

Figure 5: Boundary data were originally stored as a volume field combined with a mapping data structure for accessing appropriate boundary cells. It is now aligned to match one-to-one with the side set map and can be read in a coalesced fashion on the device.

layout as a volume field with a data structure that contained a list of indices corresponding to cells that belong to the boundary. This effectively meant that each thread in a device block was loading data from non-consecutive locations in memory which is a highly inefficient access pattern for GPUs. By aligning boundary data to match the layout of the side set mapping data structure, all boundary fields are now read efficiently within a device kernel. Modifying the boundary data layouts had the additional benefit of significantly reducing memory usage for both CPU and GPU. A speedup of roughly 1.2 and 8.7 was achieved on CPUs and GPUs, respectively, when analyzing finite element assembly performance relative to the old implementation.

4.4 Matrix dependent semicoarsening algebraic multigrid

Performance portable SA-AMG is provided by Trilinos/MueLu and is activated by using the Kokkos version of each component. This was extended to also include performance portable matrix dependent grid transfers for semicoarsening to complete the performance portable MDSC-AMG preconditioner introduced in Section 3.3.

The Kokkos version of MDSC (MDSC-Kokkos) uses Kokkos `View` as temporary data structures while assembling the prolongation matrix. Kokkos `parallel_for` is used to fill the contribution from each vertical line in parallel. A block tridiagonal system is assembled for each coarse layer in a vertical line on the fly. The system is then solved inline within the kernel using `KokkosBatched SerialLU` from Kokkos Kernels [71] where each thread performs an LU factorization without pivoting. The performance portable implementation is designed specifically as a batched direct solver for many small matrices and, in this case, additional optimizations are not needed. Once the solution is obtained, it is placed directly in the prolongation matrix.

Three performance portable point smoothers are provided by Trilinos/Ifpack2: MT Gauss-Seidel, Two-stage Gauss-Seidel, and Chebyshev. An autotuning framework using random search via Scikit-learn [48] is developed and used to determine the best smoother parameters for a small ALI test problem. We found that the performance portable smoothers did not outperform the serial line and point Gauss-Seidel smoothers in CPU simulations but Chebyshev smoothers performed the best in GPU simulations. Thus, the best CPU simulations continue to use the original line and point Gauss-Seidel smoothers while GPU simulations utilize the Chebyshev smoothers.

4.5 Automated performance testing

Performance tests are constructed as an extension of nightly regression tests. For example, a regression test might compute the steady-state solution of Equation (1) on a coarse Greenland mesh and compare computed surface velocities with known surface velocities. A performance test would perform the same calculation but also compare the end-to-end wall-clock time of the simulation to a specified value. Unfortunately, HPC clusters regularly exhibit large variations in performance causing performance tests to fail without any changes to the software. Thus, this method of performance testing is rarely used and changes in performance can go unnoticed for weeks or months.

A fundamental problem in maintaining performance tests is the ability to assess variations in performance on HPC systems. This requires a statistical approach to determine performance regressions and improvements. This is exemplified in [30] where the authors provide methods of measuring and reporting performance on HPC systems. Performance regressions, or performance degradation in software execution, can occur through various mechanism, including changes in compilers, third party libraries, hardware, or software. As the number of developers of a scientific software stack grows, the likelihood of performance regressions increase. This is addressed by developing a framework that automatically collects performance metrics and applies a changepoint detection method to the data to detect changes in performance during nightly testing.

Changepoint detection is well researched in many fields [1, 9, 67, 15] and is the process of finding abrupt variations in time series data. A changepoint detection method performs hypothesis testing between the null hypothesis, H_0, where no change occurs and the alternative hypothesis, H_A, where a change occurs. Given the performance metric time series,

$$X = \{x_1, x_2, \ldots, x_n\}, \tag{14}$$

where n is the number of historical samples collected while testing for a performance metric, x, a

subset of X can be defined as,
$$X_i^j = \{x_i, x_{i+1}, \ldots, x_j\}, \tag{15}$$
where i and j are the lower and upper limits of the time series, X_i^j. Two families of hypotheses are formulated as,
$$
\begin{aligned}
H_0 &: f_1^{\nu-1} = f_\nu^n, & \forall \nu \in \mathcal{K}, \\
H_A &: f_1^{\nu-1} \neq f_\nu^n, & \nu \in \mathcal{K},
\end{aligned}
\tag{16}
$$
where f_i^j is the probability density function $\forall x \in X_i^j$ and $\mathcal{K} = \{2, 3, \ldots, n\}$. This can be viewed as a generalized likelihood ratio test where H_0 states that all $x \in X$ belong to a single probability distribution while H_A states that there exists some ν such that all $x \in X_1^{\nu-1}$ and $x \in X_\nu^n$ belong to two separate probability distributions, respectively [26].

A two-sample t-test of $X_1^{\nu-1}$ and X_ν^n is performed to determine whether ν is a potential change-point. In order to perform multiple hypothesis tests, the Bonferroni correction [7] is used to adjust the significance level by α/k where α is the desired significance level and $k = n - 1$ is the number of tests. This correction is known to be overly conservative for large numbers of tests so only the largest changes in the time series are chosen. The pseudocode for this method is shown in Algorithm 1.

Algorithm 1 The single changepoint detection algorithm performs k hypothesis tests on the time series X_1^n using the significance level α. The time series is assumed to be roughly normal. The result is a set of changepoints, C, and t-statistics, S. ArgSortDesc() performs an indirect sort from largest to smallest values and returns an array of indices that would sort the array. This is used to find the k largest jumps.

1: **function** CHANGEPOINT(X_1^n, α, k)
2:    $C = []$
3:    $S = \{\}$
4:    $t^* = t_{n-2}^{1-\alpha/(2k)}$
5:    $\mathcal{K} = 1 + \text{ARGSORTDESC}(|X_2^n - X_1^{n-1}|)$
6:    **for** $\nu \in \mathcal{K}_1^k$ **do**
7:       $t = \text{T-TEST}(X_1^{\nu-1}, X_\nu^n)$
8:       **if** $|t| > t^*$ **then**
9:          $C \leftarrow [C, \nu]$
10:          $S \leftarrow S \cup \{\nu \rightarrow t\}$
      return C, S

A performance metric time series is likely to contain multiple changepoints. A sequential method is used to differentiate the time series based on previously identified changepoints. Once a change-point is detected, the method disregards any data prior to the changepoint. This ensures that changepoints are not retroactively changed as new data are introduced.

Due to the large variation in HPC systems, the time series data may also contain outliers which can be identified as changepoints. Multiple methods are used to ensure that the changepoints are accurate in the presence of outliers:

1. In any single t-test, outliers are identified on both distributions using the median absolute deviation [37] with a threshold comparable to three standard deviations. We remove at most 10% of the total data.

2. A minimum number of consecutive detections, m, of the same changepoint are needed before confirming a changepoint. This helps dilute the influence of an outlier.

3. As the time series is traversed sequentially, the sample size or "lookback window" for each test is limited by w observations. This helps avoid hypersensitivity where the smallest change in the time series becomes significant when the sample size is too large.

Algorithm 1 along with its modifications for multiple changepoint detection is implemented in python and executed during nightly performance testing. The log-transformed values are used because a log-normal distribution seems to fit the data slightly better than a normal distribution. A significance level of $\alpha = 0.005$ is chosen to ensure confidence and only the $k = 10$ largest changes are considered. $m = 3$ consecutive detections are needed to confirm a changepoint and a lookback window of $w = 30$ is chosen. This typically means that a minimum of three days are needed to detect a changepoint but this depends on data variability. Daily results are reported on an automated Jupyter notebook and posted online (see `https://sandialabs.github.io/ali-perf-data/`), and performance regressions are reported through an automated email report.

Performance regressions and improvements are quantified by utilizing changepoints to define subsets within the time series. Given a changepoint and two subsets, a 99% confidence interval (CI) for the difference in mean on log-transformed values is computed using a t-distribution. When transformed back, a relative performance ratio (speedup or slowdown) is given for the regression or improvement. Equation (17) shows an example of how the relative performance is computed,

$$\frac{\overline{\log\left(X_\nu^n\right)}}{\overline{\log\left(X_1^{\nu-1}\right)}} = \exp\left(\overline{\log\left(X_\nu^n\right)} - \overline{\log\left(X_1^{\nu-1}\right)}\right), \tag{17}$$

where the overline represents an arithmetic mean. A similar technique is also used in Sections 5.1 and 5.2 to compute speedups, proportions and efficiencies with 99% confidence intervals.

For performance comparisons, the difference in mean on log-transformed data between two performance tests needs to be established. A paired t-test is performed by taking the difference between the log-transformed data for the two tests where the dates intersect. The changepoint detection method is used on this data to identify subsets and a 99% confidence interval (CI) for the difference in mean on log-transformed data is computed on each subset. This is also computed as a relative performance ratio when transformed back.

5 Numerical results

In this section, the performance of MALI and standalone ALI is analyzed on two variable resolution Greenland ice-sheet meshes and a series of increasing higher resolution Antarctic ice-sheet meshes. In the first Greenland case, MALI is compared with and without the features described in Sections 4.1 and 4.4, and performance improvements are shown across all HPC architectures. In the second case, a weak scalability study of Antarctica shows that simulations perform best when utilizing the GPUs on modern HPC systems. In the last Greenland case, several examples are given on how the changepoint detection method described in Section 4.5 is used to identify performance regressions, improvements and differences in algorithm performance. What follows is a brief description of the experimental setup.

The MALI code base consists of three open-source software projects that are continuously updated through github repositories and tested nightly for performance and correctness on HPC machines. Table 1 shows where the three projects currently exist and the commit ids used for the performance experiments to follow. The code is compiled with the Kokkos `Serial` execution space for CPU-only simulations and `Cuda` for simulations utilizing GPUs. CPU-only simulations are executed with MPI ranks mapped to cores while GPU simulations are executed with MPI ranks

Table 1: MALI software repositories

Software	Repository	Git Branch	Commit Id
MPAS	`https://github.com/MALI-Dev/E3SM`	develop	d6309858d9
Albany	`https://github.com/sandialabs/Albany`	master	9d292d8f5
Trilinos	`https://github.com/trilinos/Trilinos`	develop	155e45e86c2

mapped to GPUs. In all experiments, CUDA-Aware MPI is turned off and `CUDA_LAUNCH_BLOCKING` is turned on.

The simulations are executed on the four architectures provided on the Cori and Summit supercomputers. A summary of each testing environment is provided in Table 2. Wall-clock time

Table 2: MALI simulations are executed on the three platforms or four architectures given below. A limited number of cores are utilized on some systems in order to keep a core idle for system operations. On Summit, an MPI-only simulation using only the CPU is tested along with an MPI+GPU simulation.

Name	Cori (HSW)	Cori (KNL)	Summit (PWR9,V100)
CPU	Intel Xeon E5-2698 v3 Haswell	Intel Xeon Phi 7250 Knights Landing	IBM POWER9
Number of Cores	16	68	22
GPU	None	None	NVIDIA Tesla V100
Node Arch.	2 CPUs	1 CPU	2 CPUs + 6 GPUs
Memory per Node	125 GiB	94 GiB	604 GiB + 15.7 GiB/GPU
CPU Compiler	Intel 19.0.3.199	Intel 19.0.3.199	gcc 9.1.0
GPU Compiler	None	None	nvcc 11.0.3
MPI	cray-mpich 7.7.10	cray-mpich 7.7.10	spectrum-mpi 10.4.0.3
Node Config.	32 MPI	64 MPI	PWR9: 42 MPI V100: 6 MPI

is captured by using MPAS and Trilinos/Teuchos timers to obtain an average time across MPI processes. The relevant timers and their descriptions are shown in Table 3.

5.1 MALI Greenland ice-sheet 1-to-10 kilometer variable resolution case

Here we consider a variable-resolution grid of the Greenland ice-sheet, which is finer in regions with a more complex flow structure, i.e. close to the margin and in regions where the observed surface velocity is higher. The 2D grid with cell spacing ranging from 1km to 10km is extruded in the vertical direction using 10 layers of variable thickness (thinner at the bed). The basal sliding condition and temperature are pre-computed using an initialization approach that matches the surface velocity observation while satisfying the first-order velocity equations coupled to a steady state enthalpy equation [53, 27]. In this case, MALI is used to perform an initial state calculation and a single time step, leading to two nonlinear solves using ALI. The temperature is held fixed during the time step. The nonlinear and linear solver tolerances are 1×10^{-5} and 1×10^{-8}, respectively. Simulations are compared with and without the features described in Sections 4.1

Table 3: The MALI timers described below are used to collect the average wall-clock time across MPI processes.

Timer	Description
Total Time	Total simulation time reported by MALI
Total Solve	Total nonlinear solve time reported by ALI
Total Fill	Total finite element assembly time starting from the ice velocity import and ending with the export of the residual and Jacobian
Preconditioner Construction	Total time constructing the MDSC-AMG preconditioner
Linear Solve	Total time in the linear solver including the application of the preconditioner

and 4.4. The cases are given in Table 4. Multiple samples are collected for each case using the

Table 4: The MALI Greenland ice-sheet 1-to-10 km variable resolution simulation is executed with and without the specific features described below.
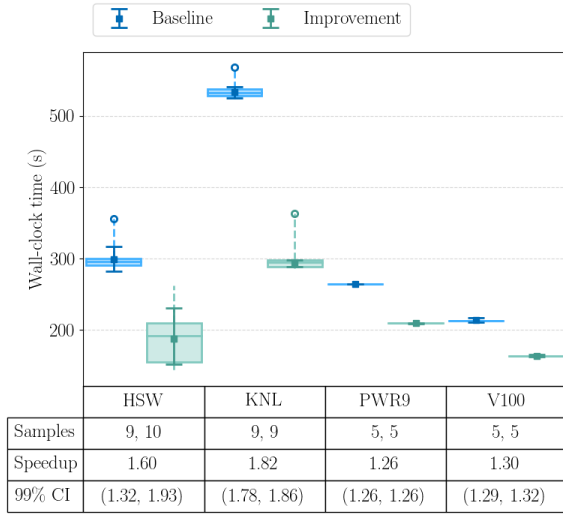
Case Name	Description
Baseline	Finite element assembly without memoization, serial MDSC, default smoother settings
Improvement	Finite element assembly with memoization, MDSC-Kokkos, optimal smoothers found through autotuning

same allocation and a mean error bar is computed using the method described in Section 4.5. A two-sample t-test of the mean difference of the log is also performed to ensure differences are statistically significant. This is then used to compute a confidence interval for the speedup of the improvement relative to the baseline. The results for each timer are shown in Figure 6.
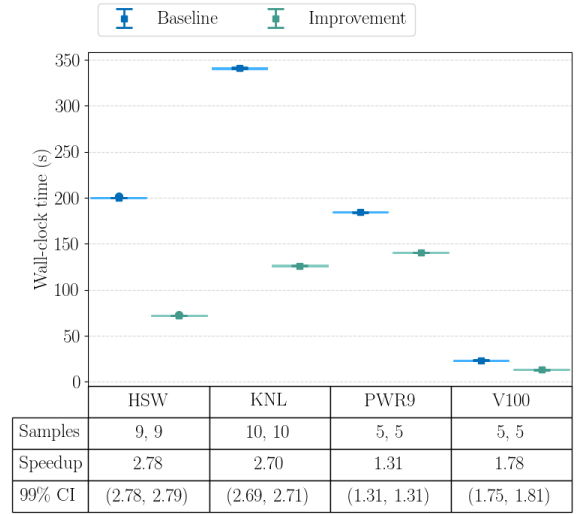
The **Total Fill** timer in Figure 6b is used to measure the improvement from memoization. The variation from this timer is small and the performance improvement is clear across all architectures. Larger improvements are seen on Cori. More variation is seen from the **Preconditioner Construction** timer in Figure 6c which is used to measure the improvement from MDSC-Kokkos. In this case, the speedup on Cori is not statistically significant but the speedup on POWER9 shows that there may be some benefit on CPU architectures. The speedup on V100 GPUs is larger and more significant. Lastly, the **Linear Solve** timer in Figure 6d is used to measure the improvement from tuning the GPU preconditioner. There is some variation and performance loss seen in the linear solve on Haswell CPUs but it is not very significant and the slowdown is not seen on the other CPU architectures. Again, there is a statistically significant speedup on V100 GPUs.

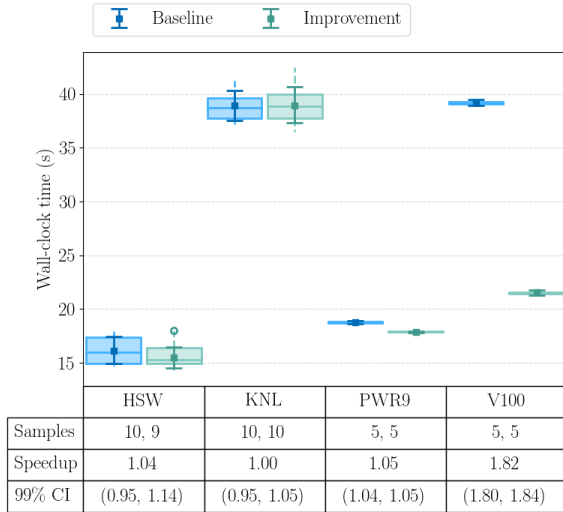The performance of the linear solver is highlighted in Table 5.

Figure 6a shows the overall performance improvement in MALI from the addition of memoization, MDSC-Kokkos and GPU preconditioner tuning. There is a statistically significant performance improvement across all architectures despite the large variation on Cori. Lastly, Figure 7 shows the proportions of total wall-clock for each architecture, case and timer. The plot shows that on CPU platforms, **Total Fill** remains a significant portion of total runtime. On GPU platforms, the finite element assembly is much less significant when compared to the linear solver and a large portion of runtime is in other portions of the code. This includes the initial setup of the data
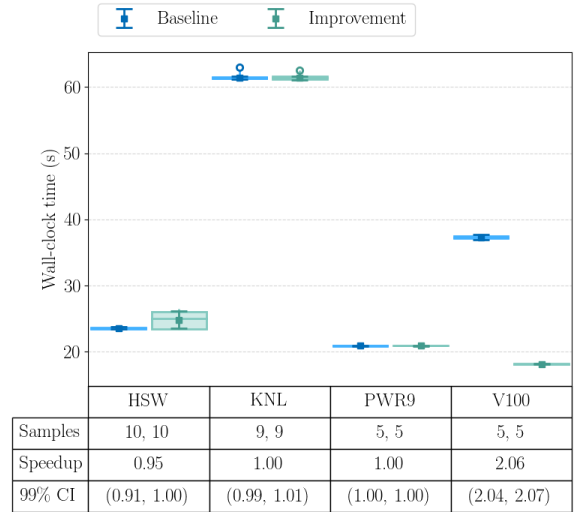
(a) Total Time

	HSW	KNL	PWR9	V100
Samples	9, 10	9, 9	5, 5	5, 5
Speedup	1.60	1.82	1.26	1.30
99% CI	(1.32, 1.93)	(1.78, 1.86)	(1.26, 1.26)	(1.29, 1.32)

(b) Total Fill

	HSW	KNL	PWR9	V100
Samples	9, 9	10, 10	5, 5	5, 5
Speedup	2.78	2.70	1.31	1.78
99% CI	(2.78, 2.79)	(2.69, 2.71)	(1.31, 1.31)	(1.75, 1.81)

(c) Preconditioner Construction

	HSW	KNL	PWR9	V100
Samples	10, 9	10, 10	5, 5	5, 5
Speedup	1.04	1.00	1.05	1.82
99% CI	(0.95, 1.14)	(0.95, 1.05)	(1.04, 1.05)	(1.80, 1.84)

(d) Linear Solve

	HSW	KNL	PWR9	V100
Samples	10, 10	9, 9	5, 5	5, 5
Speedup	0.95	1.00	1.00	2.06
99% CI	(0.91, 1.00)	(0.99, 1.01)	(1.00, 1.00)	(2.04, 2.07)

Figure 6: The MALI Greenland ice-sheet 1-to-10 km variable resolution simulation is executed multiple times on four architectures (Cori: HSW, KNL; Summit: PWR9, V100) and two cases in order to capture improvements across four timers. Architectures, timers and cases are defined in Table 2, 3 and 4, respectively. The lower/upper quartiles are shown along with the median in a box plot while a dashed line is used to show the full data range. The sample size is trimmed using the methods discussed in Section 4.5 and outliers are shown as circles. The trimmed sample sizes for the baseline and improvement are given in the table as a pair and a mean error bar is plotted. The speedup from the improvement relative to the baseline is also given along with a confidence interval (CI). CIs are reported as (LL, UL) where LL is the lower limit and UL is the upper limit.
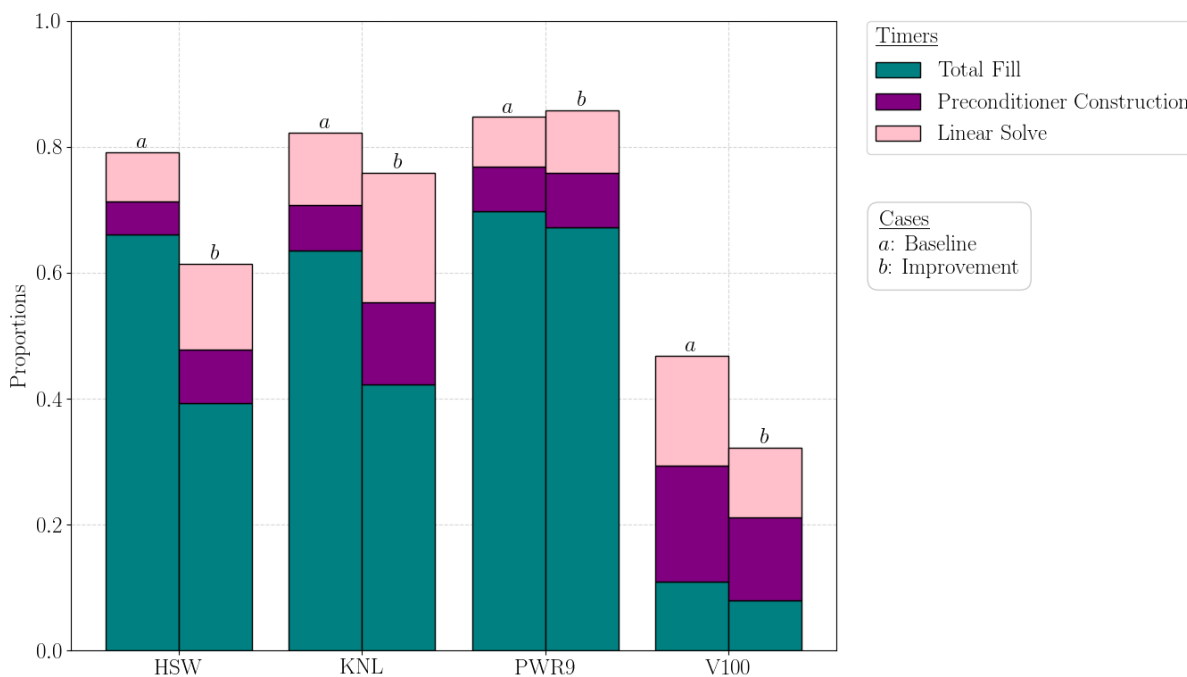
Table 5: The MALI Greenland ice-sheet 1-to-10 km variable resolution simulation is executed multiple times on four architectures (Cori: HSW, KNL; Summit: PWR9, V100) and two cases in order to capture improvements in the linear solve. Architectures and cases are defined in Table 2 and 4, respectively. The table below shows the average number of linear iterations and the total linear solve time across 26 nonlinear iterations (2 nonlinear solves) for all cases. A 99% confidence interval is reported (when statistically significant) as (LL, UL) where LL is the lower limit and UL is the upper limit.

	Case	Avg. Lin. Its.	Linear Solve Time (s)
HSW	Baseline	12.0	23.6 (23.4, 23.8)
	Improvement	12.0	24.8 (23.5, 26.1)
KNL	Baseline	12.0	61.4 (61.1, 61.6)
	Improvement	12.0	61.4 (61.1, 61.7)
PWR9	Baseline	11.7	20.9 (20.8, 21.0)
	Improvement	11.7	20.9
V100	Baseline	43.9 (43.5, 44.2)	37.3 (36.9, 37.7)
	Improvement	48.3 (48.2, 48.5)	18.1 (18.1, 18.2)

Figure 7: The MALI Greenland ice-sheet 1-to-10 km variable resolution simulation is executed multiple times on four architectures (Cori: HSW, KNL; Summit: PWR9, V100) and two cases. Architectures, timers and cases are defined in Table 2, 3 and 4, respectively. This plot shows the ratio of each timer compared to **Total Time**.

structures which only executes once but is significantly more expensive when compared to CPU platforms.

5.2 ALI Antarctica ice-sheet weak scalability study

The second case focuses on solving the first-order velocity equations in a weak scalability of ALI on a series of structured Antarctica ice-sheet meshes. This case has been used in a number of other papers including [69, 73, 27] where more detailed descriptions are given. In this case, the focus will be on how well the CPU+GPU solver performs over the CPU-only version. The five meshes vary in resolution from 1 to 16 kilometers and quadrilateral element counts vary from 51,087 to 13,413,740. The mesh is extruded by 20 layers during the setup phase, the equation is solved using the methods described in Section 3, and the mean value of the final solution is compared to a previously tested value using a relative tolerance of 1.0×10^{-5} to ensure the results remain consistent across runs and architectures. The basal sliding coefficient is predetermined using deterministic inversion from observed surface velocities [53] and a realistic temperature field is provided. Table 6 shows the number of compute nodes allocated and the total degrees of freedom for each case.

Table 6: A series of increasingly higher resolution Antarctic ice-sheet simulations are executed in a weak scalability study on four architectures. The table below shows the computing resources and degrees of freedom (DOF) associated with each case.
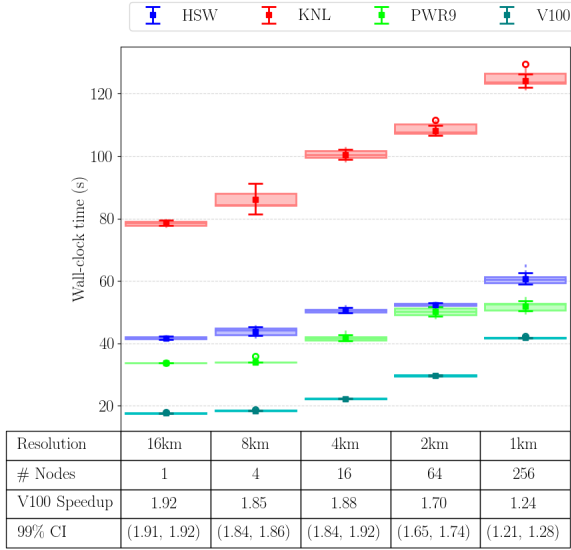
Resolution	Nodes	DOF
16km	1	2.20×10^6
8km	4	8.83×10^6
4km	16	3.53×10^7
2km	64	1.41×10^8
1km	256	5.66×10^8

In this case, standalone ALI is used to perform a single nonlinear solve where the tolerances for the nonlinear and linear solvers are set to 1×10^{-5} and 1×10^{-6}, respectively. Simulations are executed with all improvements described in Section 4. Similar to the Greenland case in Section 5.1, multiple samples are collected for each case using the same allocation and a mean error bar is computed. Two-sample t-tests of the mean difference of the log between the PWR9 and V100 cases are also performed and a 99% confidence interval for the speedup of the GPU relative to the CPU-only simulation is given in Figure 8. The first notable result is that the **Total Fill** is around 8 times faster when utilizing the V100 GPUs. The same cannot be said about the **Preconditioner Construction** and **Linear Solve** where the performance is worse. Despite this performance loss, **Total Solve** is faster when utilizing the GPU. It is important to note that the PWR9, CPU-only **Linear Solve** performed exceptionally well compared to the other architectures.
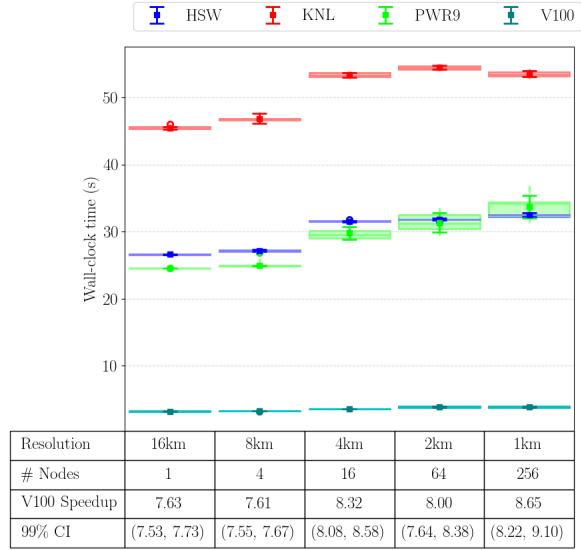
Weak scaling is used to determine how well a code is able to maintain the same wall-clock time when simulating larger problems with a proportionally larger amount of resources. In this case, the problem size is not exactly proportional to the resource size so the following formula is used to compute the weak scaling efficiency in terms of percentages,

$$\eta = \frac{(t_1/N_1)/(t_n/N_n)}{n} \times 100\%, \qquad 0 < \eta < 100\%, \tag{18}$$

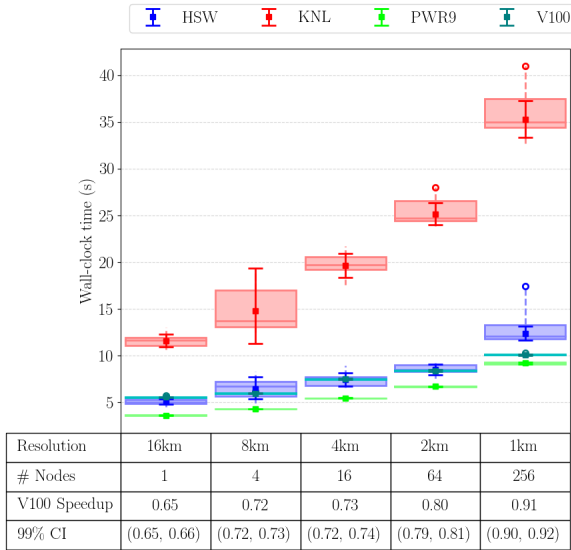where t is the wall-clock time, n is the number of compute nodes, N is the number of degrees of freedom, and larger values are better. Confidence intervals for the efficiency are computed by using
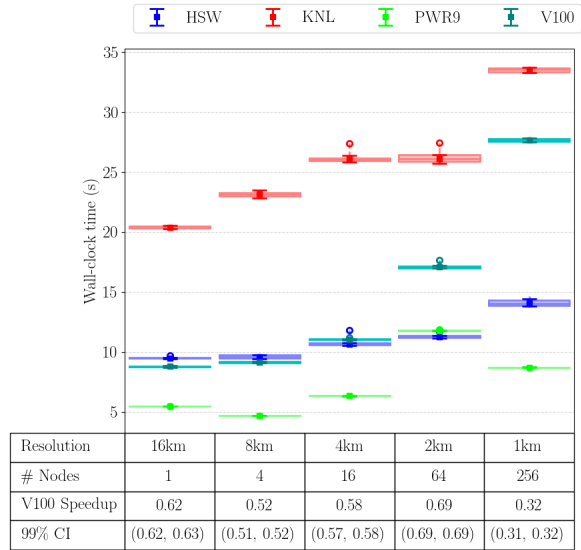
(a) Total Solve

Resolution	16km	8km	4km	2km	1km
# Nodes	1	4	16	64	256
V100 Speedup	1.92	1.85	1.88	1.70	1.24
99% CI	(1.91, 1.92)	(1.84, 1.86)	(1.84, 1.92)	(1.65, 1.74)	(1.21, 1.28)

(b) Total Fill

Resolution	16km	8km	4km	2km	1km
# Nodes	1	4	16	64	256
V100 Speedup	7.63	7.61	8.32	8.00	8.65
99% CI	(7.53, 7.73)	(7.55, 7.67)	(8.08, 8.58)	(7.64, 8.38)	(8.22, 9.10)

(c) Preconditioner Construction

Resolution	16km	8km	4km	2km	1km
# Nodes	1	4	16	64	256
V100 Speedup	0.65	0.72	0.73	0.80	0.91
99% CI	(0.65, 0.66)	(0.72, 0.73)	(0.72, 0.74)	(0.79, 0.81)	(0.90, 0.92)

(d) Linear Solve

Resolution	16km	8km	4km	2km	1km
# Nodes	1	4	16	64	256
V100 Speedup	0.62	0.52	0.58	0.69	0.32
99% CI	(0.62, 0.63)	(0.51, 0.52)	(0.57, 0.58)	(0.69, 0.69)	(0.31, 0.32)

Figure 8: A series of increasingly higher resolution Antarctic ice-sheet simulations are executed in a weak scalability study on four architectures (Cori: HSW, KNL; Summit: PWR9, V100). Four timers are captured. These are defined in Table 3. The mean error bar, median and lower/upper quartiles of each case are given along with outliers shown as circles. The speedup from the V100 CPU-GPU simulation relative to the POWER9 CPU-only simulation is also shown along with a confidence interval (CI). CIs are reported as (LL, UL) where LL is the lower limit and UL is the upper limit.

the same mean difference of the log between the single compute node case and the 256 node case. The results are shown in Table 7. In this study, the Haswell CPU performed the best when looking

Table 7: A series of increasingly higher resolution Antarctic ice-sheet simulations are executed in a weak scalability study on up to 256 compute nodes on Cori and Summit. Four architectures (Cori: HSW, KNL; Summit: PWR9, V100) are tested and four timers are captured as defined in Table 3. A weak scalability efficiency is computed for each case where one compute node is used as the reference. Larger values are better. A 99% confidence interval is reported as (LL, UL) where LL is the lower limit and UL is the upper limit.

	Total Solve	Total Fill	Preconditioner Construction	Linear Solve
HSW	68.9% (67.0, 70.9)	82.2% (81.5, 82.9)	41.2% (38.2, 44.5)	67.5% (66.2, 68.8)
KNL	63.5% (62.3, 64.6)	85.3% (84.5, 86.0)	33.0% (30.8, 35.5)	61.1% (60.6, 61.6)
PWR9	65.1% (63.3, 66.9)	73.1% (70.0, 76.4)	39.5% (39.0, 40.0)	63.0% (62.9, 63.1)
V100	42.2% (42.0, 42.4)	82.9% (80.5, 85.4)	55.2% (54.7, 55.8)	31.9% (31.6, 32.2)

at the **Total Solve** while the CPU+GPU case performed the worst. The **Total Fill** performed well across all architectures and there's a noticeable improvement on the GPU compared to previous studies [74]. In **Preconditioner Construction**, the CPU+GPU case performed best. The main cause for poor scaling on GPU platforms is visible in the **Linear Solve**. This can be explained by looking at the performance of the linear solver in Table 8. The average number of linear iterations from the GPU linear solve is much larger (reaching the maximum iteration constraint) at 256 compute nodes which contributed to worse scaling. The PWR9, CPU-only case also has much smaller linear solve times compared to the other architectures.

Figure 9 shows the proportions of total wall-clock for each architecture, resolution and timer. On CPU platforms, **Total Fill** is the dominant contributor to **Total Solve** performance across all resolutions. At lower resolutions, **Preconditioner Construction** becomes a larger contributor. On GPU platforms, it's clear that **Linear Solve** is the largest contributor across all resolutions with **Total Fill** falling to less than 10% at the lowest resolution.

5.3 ALI Greenland ice-sheet 1-to-7 kilometer variable resolution performance test

The last case focuses on solving the first-order velocity equations for a Greenland ice-sheet, 1-to-7 kilometer variable resolution mesh in a nightly performance testing framework for ALI. The numerical test is used to identify performance regressions and improvements within ALI. In this test, a two-dimensional, unstructured, Greenland ice-sheet mesh with 479,930 triangle elements is used. The test first extrudes the mesh by 10 layers using 3 tetrahedra per layer to create a mesh with 14,397,900 elements and 5,520,460 degrees of freedom. Then, equation (1) is solved using the methods described in Section 3, and the mean value of the final solution is compared to previously tested values using a relative tolerance of 1.0×10^{-5}. The basal sliding coefficient is estimated using deterministic inversion from observed surface velocities [53] and a realistic temperature field is provided. The test is currently running on two small clusters with different HPC architectures as shown in Table 9. The simulations executed on Blake and Weaver utilize 8 and 2 nodes, respectively. The historical **Total Time** for the two cases is shown in Figure 10. The plots show variability associated with the code base and the system. Statistically significant changepoints are detected
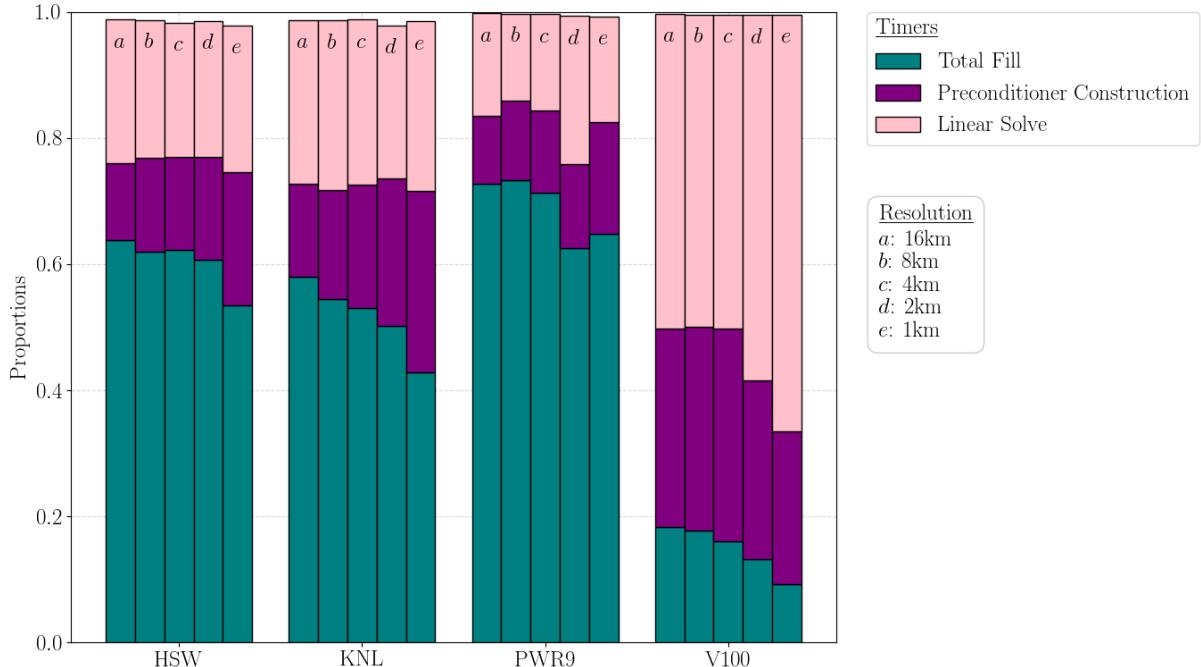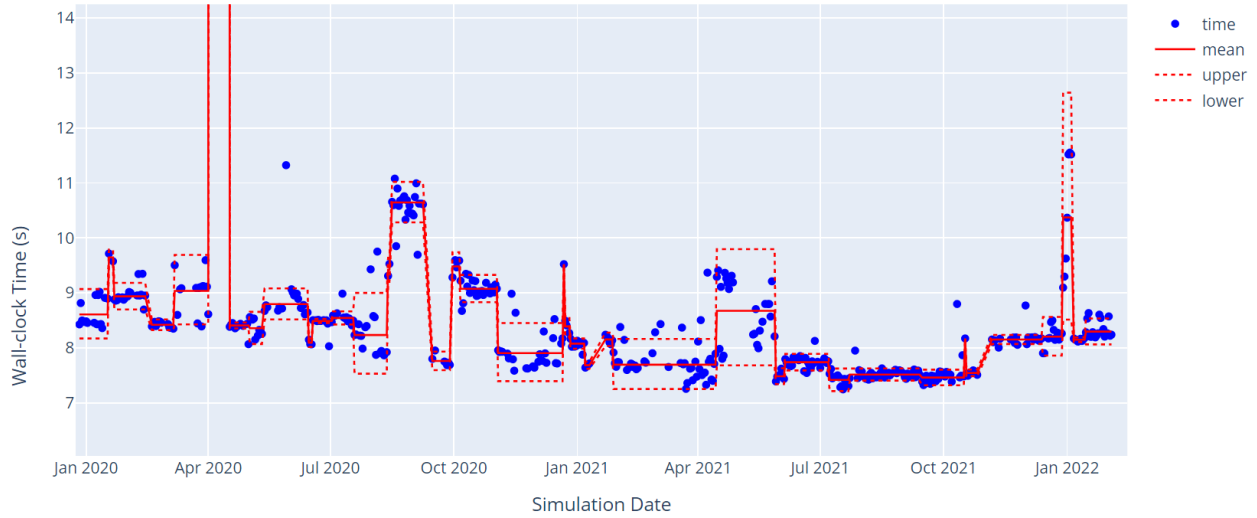
Table 8: A series of increasingly higher resolution Antarctic ice-sheet simulations are executed in a weak scalability study on four architectures (Cori: HSW, KNL; Summit: PWR9, V100). The table below shows the total number of nonlinear iterations (1 nonlinear solve), the average number of linear iterations per nonlinear iteration for all cases and the total linear solve time. A 99% confidence interval is reported (when statistically significant) as (LL, UL) where LL is the lower limit and UL is the upper limit.

	Resolution	Nodes	Nlin. Its.	Avg. Lin. Its.	Linear Solve Time (s)
HSW	16km	1	8	16.5	9.5s (9.4, 9.5)
	8km	4	8	15.5	9.6s (9.4, 9.8)
	4km	16	9	15.2	10.7s (10.6, 10.8)
	2km	64	9	15.1	11.3s (11.2, 11.4)
	1km	256	9	17.9	14.1s (13.8, 14.4)
KNL	16km	1	8	16.2	20.4s (20.3, 20.5)
	8km	4	8	15.6	23.1s (22.8, 23.5)
	4km	16	9	15.2	26.1s (25.8, 26.4)
	2km	64	9	14.6	26.1s (25.7, 26.4)
	1km	256	9	18.2	33.5s (33.3, 33.7)
PWR9	16km	1	8	16.1	5.5s
	8km	4	8	13.1	4.7s
	4km	16	9	14.8	6.4s (6.3, 6.4)
	2km	64	9	24.0	11.8s
	1km	256	9	17.7	8.7s
V100	16km	1	8	88.7 (88.6, 88.8)	8.8s (8.7, 8.9)
	8km	4	8	87.2 (87.1, 87.3)	9.1s (9.1, 9.2)
	4km	16	9	89.6 (89.6, 89.7)	11.1s (11.0, 11.1)
	2km	64	9	131.4 (131.2, 131.6)	17.1s (17.0, 17.2)
	1km	256	9	194.2 (193.6, 194.8)	27.7s (27.5, 27.8)

Table 9: ALI nightly performance tests are executed nightly on the two small clusters given below.

Name	Blake	Weaver
CPU	Intel Xeon Platinum 8160 Skylake	IBM POWER9
Number of Cores	24	20
GPU	None	NVIDIA Tesla V100
Node Arch.	2 CPUs	2 CPUs + 4 GPUs
Memory per Node	188 GiB	319 GiB + 15.7 GiB/GPU
CPU Compiler	Intel 18.1.163	gcc 7.2.0
GPU Compiler	None	nvcc 10.1.105
MPI	openmpi 2.1.2	openmpi 4.0.1
Node Config.	48 MPI	4 MPI

Figure 9: A series of increasingly higher resolution Antarctic ice-sheet simulations are executed in a weak scalability study on four architectures (Cori: HSW, KNL; Summit: PWR9, V100). Timers are defined in Table 3. This plot shows the ratio of each timer compared to **Total Solve**.
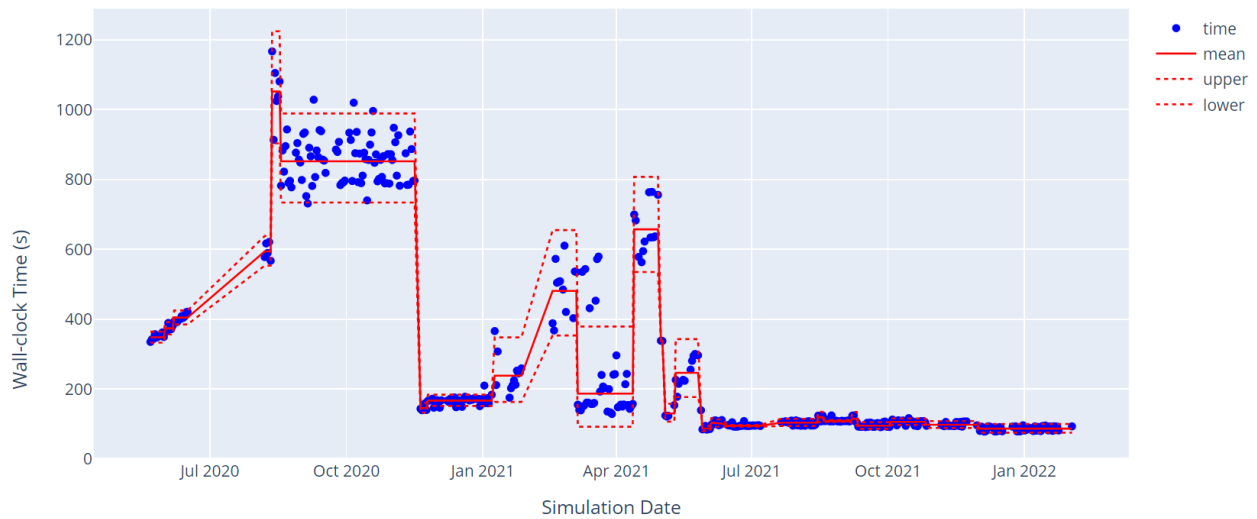
using the methods described in Section 4.5 in order to identify regressions and improvements. The simulations in both time series utilize memoization so the CPU performance in Figure 10a has not changed much. In contrast, many of the other changes described in section 4 have been added over the course of the time series causing dramatic improvements to GPU performance as shown in Figure 10b.

Figure 10 also shows many performance regressions and improvements over the course of the time series. One recent example is the transition to Kokkos 3.5.0 which caused a regression to CPU performance. The regression along with the improvement from the fix is shown in Figure 11. In this particular case, the usage of Kokkos `atomic_add` was changed for the `Serial` execution space. This caused a 15% (99% CI: 14%, 16%) slowdown to **Total Fill**. This was then fixed manually in Albany by avoiding the use of `atomic_add` when running with a `Serial` execution space. After the change, Figure 11b shows that the performance improved by 15% (99% CI: 13%, 16%), reverting the performance loss due to the original regression. Since then, the issue was reported and a fix has been introduced into Kokkos.

Algorithm performance comparisons can also be historically tracked within the performance testing framework. One example is the performance comparison of the finite element assembly with and without memoization. In this case, two performance tests which store the wall-clock time of 100 residual and Jacobian evaluations are tracked. Figure 12 shows the observations between the two cases paired by simulation date on Blake and Weaver. Both plots show that the test with memoization has performed faster than the test without memoization throughout the entire time series. On the CPU platform shown in Figure 12a, the relative performance has not changed much but on the GPU platform in Figure 12b the relative performance has increased over time. The key difference in this case was the boundary condition improvements which significantly reduced
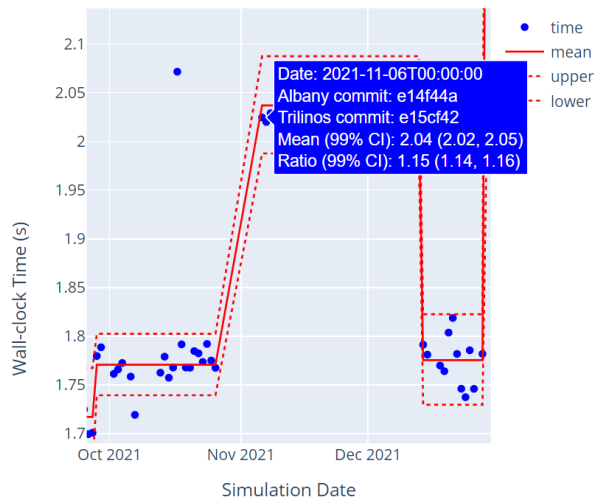
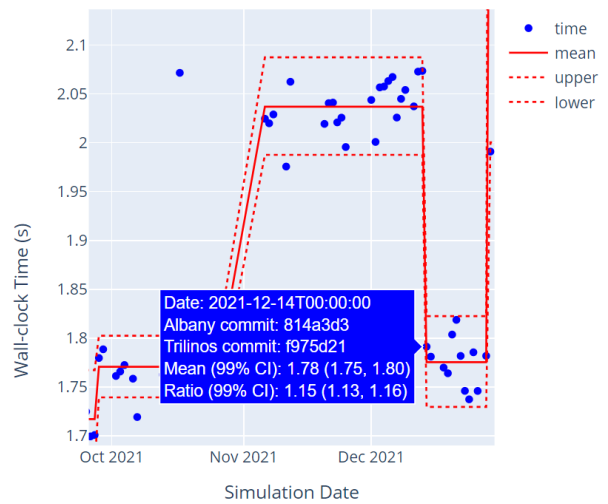(a) **Total Time** on 8 Blake nodes (384 Skylake cores)

(b) **Total Time** on 2 Weaver nodes (8 V100 GPUs)

Figure 10: The ALI Greenland ice-sheet 1-to-7 km variable resolution simulation is executed nightly on two platforms in order to detect regressions and improvements. Blue markers are recorded wall-clock time. Means are computed between changepoints and are indicated by solid red lines. The dotted red lines are ± two standard deviations.
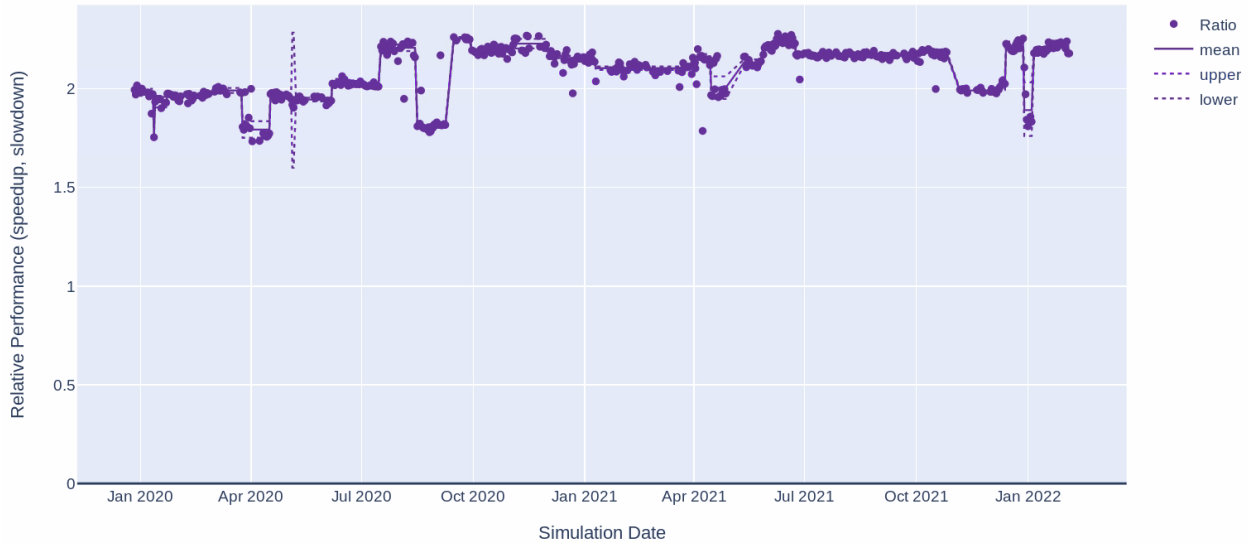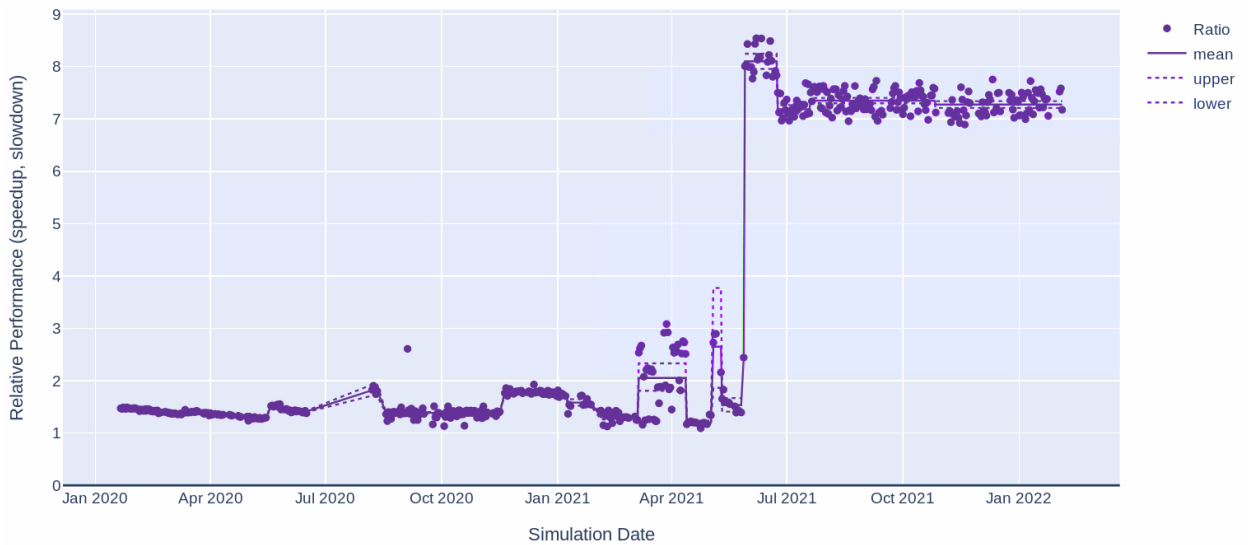
(a) **Total Fill** regression (b) **Total Fill** improvement

Figure 11: The ALI Greenland ice-sheet 1-to-7 km variable resolution simulation is executed nightly on Blake in order to detect regressions and improvements. Blue markers are recorded wall-clock time. Means are computed between changepoints and are indicated by solid red lines. The mean values are also given in the blue box along with a 99% confidence interval (CI). CIs are reported as (LL, UL) where LL is the lower limit and UL is the upper limit. The dotted red lines are ± two standard deviations. Lastly, the blue boxes also show ratios (speedup, slowdown) between sets with a 99% CI. This particular case highlights a performance regression and its later improvement from the fix.

(a) **Total Fill** on 8 Blake nodes (384 Skylake cores)

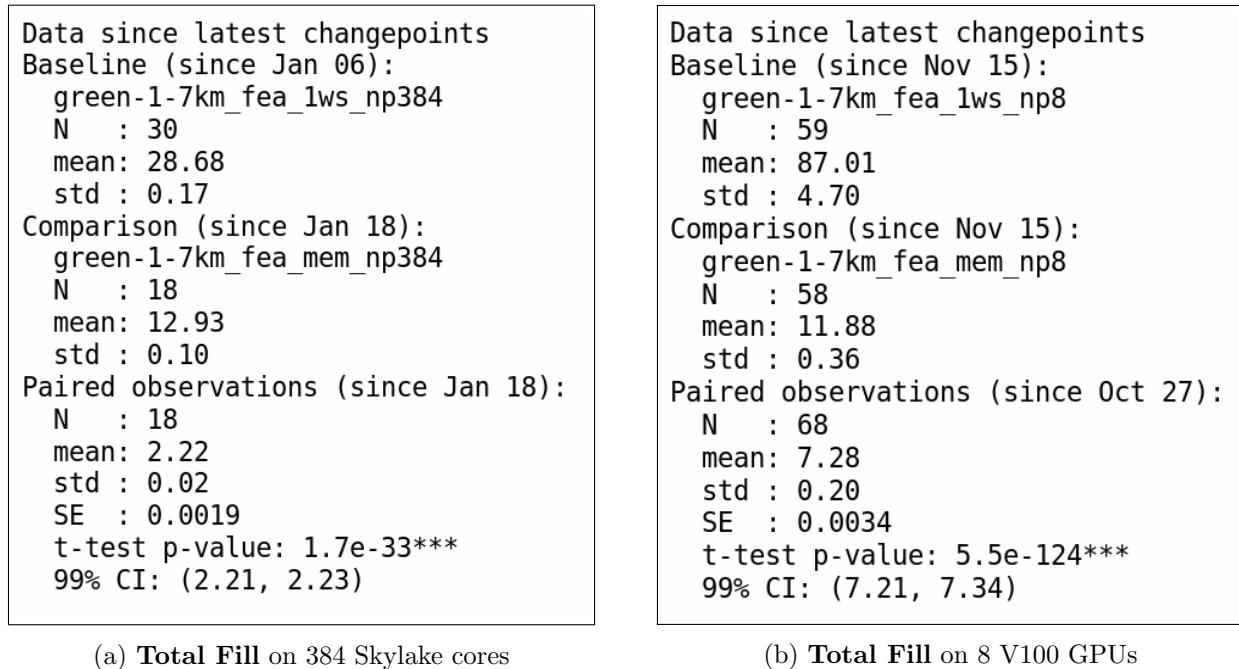(b) **Total Fill** on 2 Weaver nodes (8 V100 GPUs)

Figure 12: The ALI Greenland ice-sheet 1-to-7 km variable resolution finite element assembly tests with and without memoization are executed nightly on two platforms in order to detect regressions, improvements and analyze comparisons. Observations from the two cases are joined by date by taking the difference between the log of the timer data and plotting the relative performance (speedup, slowdown) with markers. Solid lines indicate means between changepoints and dotted lines represent a 99% confidence interval for the mean.

the **Total Fill** time and caused evaluators without memoization to take up a larger portion of the **Total Fill** time.

The time series since the most recently detected changepoint can be used to determine whether the latest relative performance for memoization is statistically significant. A paired t-test is used to test the mean difference and the data is summarized as shown in Figure 13. The results show

```
Data since latest changepoints
Baseline (since Jan 06):
  green-1-7km_fea_1ws_np384
  N   : 30
  mean: 28.68
  std : 0.17
Comparison (since Jan 18):
  green-1-7km_fea_mem_np384
  N   : 18
  mean: 12.93
  std : 0.10
Paired observations (since Jan 18):
  N   : 18
  mean: 2.22
  std : 0.02
  SE  : 0.0019
  t-test p-value: 1.7e-33***
  99% CI: (2.21, 2.23)
```

```
Data since latest changepoints
Baseline (since Nov 15):
  green-1-7km_fea_1ws_np8
  N   : 59
  mean: 87.01
  std : 4.70
Comparison (since Nov 15):
  green-1-7km_fea_mem_np8
  N   : 58
  mean: 11.88
  std : 0.36
Paired observations (since Oct 27):
  N   : 68
  mean: 7.28
  std : 0.20
  SE  : 0.0034
  t-test p-value: 5.5e-124***
  99% CI: (7.21, 7.34)
```

(a) **Total Fill** on 384 Skylake cores   (b) **Total Fill** on 8 V100 GPUs

Figure 13: The ALI Greenland ice-sheet 1-to-7 km variable resolution finite element assembly tests with and without memoization are executed nightly on two platforms in order to detect regressions, improvements and analyze comparisons. Observations from the two cases are joined by date by taking the difference between the log of the timer data and performing a paired t-test. The figures show sample sizes since the last changepoint for each test as well as mean wall-clock times (s) and standard deviation. The paired relative performance (speedup) is also given along with the standard error, p-value and a 99% confidence interval (CI). CIs are reported as (LL, UL) where LL is the lower limit and UL is the upper limit.

that the current estimated speedup from memoization for this case is 2.22 (99% CI: 2.21, 2.23) on CPU platforms and 7.28 (99% CI: 7.21, 7.34) on GPU platforms.

6   Conclusions

In this paper, the performance portable features of MALI are introduced and analyzed on the two supercomputing clusters: NERSC Cori and OLCF Summit. First, the first-order velocity model and the mass continuity equation are introduced along with their implementations within Albany Land Ice and MPAS, respectively. This is used to further describe improvements that have been made to the finite element assembly process and linear solve within MALI. The new features focus on improving performance portability in MALI but are extensible to other applications targeting HPC systems.

Two numerical experiments are provided to analyze the expected performance on different HPC architectures. The first case utilized MALI to simulate an initial state calculation and single time

step for a Greenland ice sheet 1-to-10 kilometer resolution mesh and compared baseline simulations without specific features with improved simulations with the features described in the paper. The results show that finite element assembly with memoization, MDSC-Kokkos and tuned smoothers are performant across all architectures with an expected speedup of 1.60 (99% CI: 1.32, 1.93) on Cori-Haswell, 1.82 (99% CI: 1.78, 1.86) on Cori-KNL, 1.26 on Summit-POWER9 and 1.30 (99% CI: 1.29, 1.32) on Summit-V100. The study also highlights specific regions in need of improvement in the model. In particular, the need to improve the performance of the coupling between MPAS and Albany, the finite element assembly process on CPUs, and the preconditioner on GPUs.

The second numerical experiment utilized ALI to perform a steady-state simulation of the Antarctic ice sheet on a series of structured meshes in a weak scalability study. The results show that simulations on Summit-V100 perform the best with a 1.92 (99% CI: 1.91, 1.92) speedup over Summit-POWER9 in the low resolution case and 1.24 (99% CI: 1.21, 1.28) speedup over Summit-POWER9 in the high resolution case. The best results on Summit are shown during finite element assembly where the speedup over Summit-POWER9 is 8.65 (99% CI: 8.22, 9.10). The results also show good weak scaling in finite element assembly for CPU/GPU but poor weak scaling in the preconditioner on CPU/GPU and in the linear solve on GPU architectures. Further analysis shows that the average number of linear iterations per nonlinear iteration increases dramatically as the resolution increases, highlighting the need for a more scalable preconditioner for this particular problem.

This paper also introduces a changepoint detection method for automated performance testing. A detailed description of the method is given along with examples of how the method can be used to detect performance regressions, improvements and differences in algorithm performance over time. In this case, an automated performance testing framework is used with ALI to simulate the Greenland ice sheet using a 1-to-7 kilometer variable resolution mesh. The results show the method being exercised on two years of data and an example of a successful detection of performance regression and improvement. The results also show an example of a nightly performance comparison where two tests are used to compare ALI with and without memoization. This case was used to show how the method can be used to detect regressions and improvements in algorithm performance over time as the utility of memoization has improved to up to 7.28 (99% CI: 7.21, 7.34) speedup over simulations without memoization on GPU platforms over the course of two years.

Data availability

The performance testing framework and data is available in `https://github.com/sandialabs/ali-perf-tests` and `https://github.com/sandialabs/ali-perf-data`. The results are accessible via a browser here: `https://sandialabs.github.io/ali-perf-data`. Performance data, pre-processing scripts and post-processing scripts are available in `https://github.com/sandialabs/ali-perf-data` under the directory paper-data/watkins2022performance.

Acknowledgments

Disclaimer

References

[1] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

[2] Hartwig Anzt, Werner Augustin, Martin Baumann, Hendryk Bockelmann, Thomas Gengenbach, Tobias Hahn, Vincent Heuveline, Eva Ketelaer, Dimitar Lukarski, Andrea Otzen, et al. HiFlow3–a flexible and hardware-aware parallel finite element package. *Preprint Series of the Engineering Mathematics and Computing Lab*, (06), 2010.

[3] Christopher G Baker and Michael A Heroux. Tpetra, and the use of generic programming in scientific computing. *Scientific Programming*, 20(2):115–128, 2012.

[4] Eric Bavier, Mark Hoemmen, Sivasankaran Rajamanickam, and Heidi Thornquist. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255, 2012.

[5] Luc Berger-Vergiat, Christian Alexander Glusa, Jonathan J Hu, Christopher Siefert, Raymond S Tuminaro, Mayr Matthias, Prokopenko Andrey, and Wiesner Tobias. MueLu User's Guide. Technical Report SAND2019-0537, Sandia National Laboratories, 2019.

[6] Heinz Blatter. Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients. *Journal of Glaciology*, 41(138):333–344, 1995.

[7] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936.

[8] Christian Fredborg Brædstrup, Anders Damsgaard, and David Lundbek Egholm. Ice-sheet modelling accelerated by graphics cards. *Computers & Geosciences*, 72:210–220, 2014.

[9] Boris Brodsky. *Change-point analysis in nonstationary stochastic models*. CRC Press, 2016.

[10] Jed Brown, Barry Smith, and Aron Ahmadia. Achieving textbook multigrid efficiency for hydrostatic ice sheet flow. *SIAM Journal on Scientific Computing*, 35(2):B359–B375, 2013.

[11] Max Carlson, Jerry Watkins, and Irina Tezaur. Improvements to the performance portability of boundary conditions in Albany Land Ice. *CSRI Summer Proceedings*, pages 177–187, 2020.

[12] Chao Chen, Leopold Cambier, Erik G Boman, Sivasankaran Rajamanickam, Raymond S Tuminaro, and Eric Darve. A robust hierarchical solver for ill-conditioned systems with applications to ice sheet modeling. *Journal of Computational Physics*, 396:819–836, 2019.

[13] Stephen L Cornford, Daniel F Martin, Daniel T Graves, Douglas F Ranken, Anne M Le Brocq, Rupert M Gladstone, Antony J Payne, Esmond G Ng, and William H Lipscomb. Adaptive mesh, finite volume modeling of marine ice sheets. *Journal of Computational Physics*, 232(1):529–549, 2013.

[14] Kurt M Cuffey and William Stanley Bryce Paterson. *The physics of glaciers*. Academic Press, 2010.

[15] David Daly, William Brown, Henrik Ingo, Jim O'Leary, and David Bradford. The use of change point detection to identify software performance regressions in a continuous integration system. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 67–75, 2020.

[16] Irina Demeshko, Jerry Watkins, Irina K Tezaur, Oksana Guba, William F Spotz, Andrew G Salinger, Roger P Pawlowski, and Michael A Heroux. Toward performance portability of the Albany finite element analysis code using the Kokkos library. *The International Journal of High Performance Computing Applications*, 2018.

[17] Phillip Dickens. A performance and scalability analysis of the MPI based tools utilized in a large ice sheet model executing in a multicore environment. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 131–147. Springer, 2015.

[18] John K Dukowicz, Stephen F Price, and William H Lipscomb. Consistent approximations and boundary conditions for ice-sheet dynamics from a principle of least action. *Journal of Glaciology*, 56(197):480–496, 2010.

[19] H Carter Edwards, Christian R Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014.

[20] Tamsin L Edwards, Sophie Nowicki, Ben Marzeion, Regine Hock, Heiko Goelzer, Hélène Seroussi, Nicolas C Jourdain, Donald A Slater, Fiona E Turner, Christopher J Smith, et al. Projected land ice contributions to twenty-first-century sea level rise. *Nature*, 593(7857):74–82, 2021.

[21] Yannic Fischler, Martin Rückamp, Christian Bischof, Vadym Aizinger, Mathieu Morlighem, and Angelika Humbert. A scalability study of the ice-sheet and sea-level system model (ISSM, version 4.18). *Geoscientific Model Development Discussions*, pages 1–33, 2021.

[22] Gregory Flato, Jochem Marotzke, Babatunde Abiodun, Pascale Braconnot, Sin Chan Chou, William Collins, Peter Cox, Fatima Driouech, Seita Emori, Veronika Eyring, et al. Evaluation of climate models. In *Climate change 2013: the physical science basis. Contribution of Working*

Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, pages 741–866. Cambridge University Press, 2014.

[23] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. The SPACE of developer productivity: There's more to it than you think. *Queue*, 19(1):20–48, 2021.

[24] O Gagliardini, T Zwinger, F Gillet-Chaulet, G Durand, L Favier, B De Fleurian, R Greve, M Malinen, C Martín, P Råback, et al. Capabilities and performance of Elmer/Ice, a new-generation ice sheet model. *Geoscientific Model Development*, 6(4):1299–1318, 2013.

[25] Heiko Goelzer, Sophie Nowicki, Anthony Payne, Eric Larour, Helene Seroussi, William H. Lipscomb, Jonathan Gregory, Ayako Abe-Ouchi, Andrew Shepherd, Erika Simon, Cécile Agosta, Patrick Alexander, Andy Aschwanden, Alice Barthel, Reinhard Calov, Christopher Chambers, Youngmin Choi, Joshua Cuzzone, Christophe Dumas, Tamsin Edwards, Denis Felikson, Xavier Fettweis, Nicholas R. Golledge, Ralf Greve, Angelika Humbert, Philippe Huybrechts, Sebastien Le Clec'H, Victoria Lee, Gunter Leguy, Chris Little, Daniel Lowry, Mathieu Morlighem, Isabel Nias, Aurelien Quiquet, Martin Rückamp, Nicole Jeanne Schlegel, Donald A. Slater, Robin Smith, Fiamma Straneo, Lev Tarasov, Roderik Van De Wal, and Michiel Van Den Broeke. The future sea-level contribution of the Greenland ice sheet: A multi-model ensemble study of ISMIP6. *Cryosphere*, 14(9):3071–3096, 2020.

[26] Douglas M Hawkins, Peihua Qiu, and Chang Wook Kang. The changepoint model for statistical process control. *Journal of quality technology*, 35(4):355–366, 2003.

[27] Alexander Heinlein, Mauro Perego, and Sivasankaran Rajamanickam. FROSch preconditioners for land ice simulations of Greenland and Antarctica. *SIAM Journal on Scientific Computing*, 44(2):B339–B367, 2022.

[28] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[29] Michael A Heroux, Rajeev Thakur, Lois McInnes, Jeffrey S Vetter, Xiaoye Sherry Li, James Aherns, Todd Munson, and Kathryn Mohror. ECP software technology capability assessment report. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2020.

[30] Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–12, 2015.

[31] Matthew J Hoffman, Mauro Perego, Stephen F Price, William H Lipscomb, Tong Zhang, Douglas Jacobsen, Irina Tezaur, Andrew G Salinger, Raymond Tuminaro, and Luca Bertagna. MPAS-Albany Land Ice (MALI): a variable-resolution ice sheet model for Earth system modeling using Voronoi grids. *Geoscientific Model Development*, 11(9):3747–3780, 2018.

[32] Richard D Hornung and Jeffrey A Keasler. The RAJA portability layer: overview and status. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2014.

[33] Tobin Isaac, Georg Stadler, and Omar Ghattas. Solution of nonlinear stokes equations discretized by high-order finite elements on nonconforming and anisotropic meshes, with application to ice sheet dynamics. *SIAM Journal on Scientific Computing*, 37(6):B804–B833, 2015.

[34] Upulee Kanewala and James M Bieman. Testing scientific software: A systematic literature review. *Information and software technology*, 56(10):1219–1232, 2014.

[35] E Larour, H Seroussi, M Morlighem, and E Rignot. Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM). *Journal of Geophysical Research: Earth Surface*, 117(F1), 2012.

[36] Anders Levermann, Ricarda Winkelmann, Torsten Albrecht, Heiko Goelzer, Nicholas R. Golledge, Ralf Greve, Philippe Huybrechts, Jim Jordan, Gunter Leguy, Daniel Martin, Mathieu Morlighem, Frank Pattyn, David Pollard, Aurelien Quiquet, Christian Rodehacke, Helene Seroussi, Johannes Sutter, Tong Zhang, Jonas Van Breedam, Reinhard Calov, Robert DeConto, Christophe Dumas, Julius Garbe, G. Hilmar Gudmundsson, Matthew J. Hoffman, Angelika Humbert, Thomas Kleiner, William H. Lipscomb, Malte Meinshausen, Esmond Ng, Sophie M. J. Nowicki, Mauro Perego, Stephen F. Price, Fuyuki Saito, Nicole-Jeanne Schlegel, Sainan Sun, and Roderik S. W. van de Wal. Projecting Antarctica's contribution to future sea level rise from basal ice shelf melt using linear response functions of 16 ice sheet models (LARMIP-2). *Earth System Dynamics*, 11(1):35–76, feb 2020.

[37] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of experimental social psychology*, 49(4):764–766, 2013.

[38] GR Markall, A Slemmer, DA Ham, PHJ Kelly, CD Cantwell, and SJ Sherwin. Finite element assembly strategies on multi-core and many-core architectures. *International Journal for Numerical Methods in Fluids*, 71(1):80–97, 2013.

[39] David S Medina, Amik St-Cyr, and Tim Warburton. OCCA: A unified approach to multi-threading languages. *arXiv preprint arXiv:1403.0968*, 2014.

[40] JR Neely. DOE Centers of Excellence performance portability meeting. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), 2016.

[41] JF Nye. The distribution of stress and velocity in glaciers and ice-sheets. *Proc. R. Soc. Lond. A*, 239(1216):113–133, 1957.

[42] Michael Oppenheimer, Bruce Glavovic, Jochen Hinkel, Roderik van de Wal, Alexandre K Magnan, Amro Abd-Elgawad, Rongshuo Cai, Miguel Cifuentes-Jara, Robert M Deconto, Tuhin Ghosh, et al. Sea level rise and implications for low lying islands, coasts and communities. 2019.

[43] Frank Pattyn. A new three-dimensional higher-order thermomechanical ice sheet model: Basic sensitivity, ice stream development, and ice flow across subglacial lakes. *Journal of Geophysical Research: Solid Earth*, 108(B8), 2003.

[44] Frank Pattyn, Lionel Favier, Sainan Sun, and Gaël Durand. Progress in Numerical Modeling of Antarctic Ice-Sheet Dynamics. *Current Climate Change Reports*, pages 1–11, jul 2017.

[45] Roger P Pawlowski, Eric T Phipps, and Andrew G Salinger. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part I: Template-based generic programming. *Scientific Programming*, 20(2):197–219, 2012.

[46] Roger P Pawlowski, Eric T Phipps, Andrew G Salinger, Steven J Owen, Christopher M Siefert, and Matthew L Staten. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part II: Application to partial differential equations. *Scientific Programming*, 20(3):327–345, 2012.

[47] Antony J. Payne, Sophie Nowicki, Ayako Abe-Ouchi, Cécile Agosta, Patrick Alexander, Torsten Albrecht, Xylar Asay-Davis, Andy Aschwanden, Alice Barthel, Thomas J. Bracegirdle, Reinhard Calov, Christopher Chambers, Youngmin Choi, Richard Cullather, Joshua Cuzzone, Christophe Dumas, Tamsin L. Edwards, Denis Felikson, Xavier Fettweis, Benjamin K. Galton-Fenzi, Heiko Goelzer, Rupert Gladstone, Nicholas R. Golledge, Jonathan M. Gregory, Ralf Greve, Tore Hattermann, Matthew J. Hoffman, Angelika Humbert, Philippe Huybrechts, Nicolas C. Jourdain, Thomas Kleiner, Peter Kuipers Munneke, Eric Larour, Sebastien Le clec'h, Victoria Lee, Gunter Leguy, William H. Lipscomb, Christopher M. Little, Daniel P. Lowry, Mathieu Morlighem, Isabel Nias, Frank Pattyn, Tyler Pelle, Stephen F. Price, Aurélien Quiquet, Ronja Reese, Martin Rückamp, Nicole-Jeanne Schlegel, Hélène Seroussi, Andrew Shepherd, Erika Simon, Donald Slater, Robin S. Smith, Fiammetta Straneo, Sainan Sun, Lev Tarasov, Luke D. Trusel, Jonas Van Breedam, Roderik van de Wal, Michiel van den Broeke, Ricarda Winkelmann, Chen Zhao, Tong Zhang, and Thomas Zwinger. Future sea level change under CMIP5 and CMIP6 scenarios from the Greenland and Antarctic ice sheets. *Geophysical Research Letters*, pages 1–8, 2021.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Zedong Peng, Xuanyi Lin, Michelle Simon, and Nan Niu. Unit and regression tests of scientific software: A study on SWMM. *Journal of Computational Science*, 53:101347, 2021.

[50] S John Pennycook, Jason D Sewall, Douglas W Jacobsen, Tom Deakin, and Simon McIntosh-Smith. Navigating performance, portability, and productivity. *Computing in Science & Engineering*, 23(5):28–38, 2021.

[51] Simon J Pennycook, JD Sewall, and VW Lee. A metric for performance portability. *arXiv preprint arXiv:1611.07409*, 2016.

[52] SJ Pennycook, JD Sewall, and VW Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, 2017.

[53] Mauro Perego, Stephen Price, and Georg Stadler. Optimal initial conditions for coupling ice sheet models to Earth system models. *Journal of Geophysical Research: Earth Surface*, 119(9):1894–1917, 2014.

[54] Eric Phipps and Roger Pawlowski. Efficient expression templates for operator overloading-based automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, pages 309–319. Springer, 2012.

[55] TOP500 Project. June 2021 TOP500 list. `https://www.top500.org/lists/top500/2021/06/`. [Online; accessed 25-October-2021].

[56] Andrey Prokopenko, Christopher Siefert, Jonathan J Hu, Mark Frederick Hoemmen, and Alicia Marie Klinvex. Ifpack2 user's guide 1.0. Technical report, Sandia National Laboratories, 2016.

[57] David A Randall, Richard A Wood, Sandrine Bony, Robert Colman, Thierry Fichefet, John Fyfe, Vladimir Kattsov, Andrew Pitman, Jagadish Shukla, Jayaraman Srinivasan, et al. Climate models and their evaluation. In *Climate change 2007: The physical science basis. Contribution of Working Group I to the Fourth Assessment Report of the IPCC (FAR)*, pages 589–662. Cambridge University Press, 2007.

[58] Ludovic Räss, Aleksandar Licul, Frédéric Herman, Yury Y Podladchikov, and Jenny Suckale. Modelling thermomechanical ice deformation using an implicit pseudo-transient method (FastICE v1. 0) based on graphical processing units (GPUs). *Geoscientific Model Development*, 13(3):955–976, 2020.

[59] Ludovic Räss, Ivan Utkin, Thibault Duretz, Samuel Omlin, and Yuri Y Podladchikov. Assessing the robustness and scalability of the accelerated pseudo-transient method towards exascale computing. *Geoscientific Model Development Discussions*, 2022:1–46, 2022.

[60] Florian Rathgeber, David A Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew TT McRae, Gheorghe-Teodor Bercea, Graham R Markall, and Paul HJ Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software (TOMS)*, 43(3):24, 2017.

[61] Florian Rathgeber, Graham R Markall, Lawrence Mitchell, Nicolas Loriant, David A Ham, Carlo Bertolli, and Paul HJ Kelly. PyOP2: A high-level framework for performance-portable simulations on unstructured meshes. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1116–1123. IEEE, 2012.

[62] Todd Ringler, Mark Petersen, Robert L. Higdon, Doug Jacobsen, Philip W. Jones, and Mathew Maltrud. A multi-resolution approach to global ocean modeling. *Ocean Modelling*, 69:211–232, 2013.

[63] Ian C Rutt, M Hagdorn, NRJ Hulton, and AJ Payne. The Glimmer community ice sheet model. *Journal of Geophysical Research: Earth Surface*, 114(F2), 2009.

[64] Andrew G Salinger, Roscoe A Bartlett, Andrew M Bradley, Qiushi Chen, Irina P Demeshko, Xujiao Gao, Glen A Hansen, Alejandro Mota, Richard P Muller, Erik Nielsen, Jakob Ostien, Roger Pawlowski, Mauro Perego, Eric Phipps, WaiChing Sun, and Irina Tezaur. Albany: Using component-based design to develop a flexible, generic multiphysics analysis code. *International Journal for Multiscale Computational Engineering*, 14(4), 2016.

[65] Christian Schoof and Richard CA Hindmarsh. Thin-film flows with wall slip: an asymptotic analysis of higher order glacier flow models. *The Quarterly Journal of Mechanics & Applied Mathematics*, 63(1):73–114, 2010.

[66] Hélène Seroussi, Sophie Nowicki, Antony J. Payne, Heiko Goelzer, William H. Lipscomb, Ayako Abe-Ouchi, Cécile Agosta, Torsten Albrecht, Xylar Asay-Davis, Alice Barthel, Reinhard Calov, Richard Cullather, Christophe Dumas, Benjamin K. Galton-Fenzi, Rupert Gladstone, Nicholas R. Golledge, Jonathan M. Gregory, Ralf Greve, Tore Hattermann, Matthew J.

Hoffman, Angelika Humbert, Philippe Huybrechts, Nicolas C. Jourdain, Thomas Kleiner, Eric Larour, Gunter R. Leguy, Daniel P. Lowry, Chistopher M. Little, Mathieu Morlighem, Frank Pattyn, Tyler Pelle, Stephen F. Price, Aurélien Quiquet, Ronja Reese, Nicole-Jeanne Schlegel, Andrew Shepherd, Erika Simon, Robin S. Smith, Fiammetta Straneo, Sainan Sun, Luke D. Trusel, Jonas Van Breedam, Roderik S. W. van de Wal, Ricarda Winkelmann, Chen Zhao, Tong Zhang, and Thomas Zwinger. ISMIP6 Antarctica: a multi-model ensemble of the Antarctic ice sheet evolution over the 21st century. *The Cryosphere*, 14(9):3033–3070, sep 2020.

[67] Alexander Tartakovsky, Igor Nikiforov, and Michele Basseville. *Sequential analysis: Hypothesis testing and changepoint detection*. CRC Press, 2014.

[68] Irina K Tezaur, Mauro Perego, Andrew G Salinger, Raymond S Tuminaro, and Stephen F Price. Albany/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis. *Geoscientific Model Development*, 8(4):1197, 2015.

[69] Irina K Tezaur, Raymond S Tuminaro, Mauro Perego, Andrew G Salinger, and Stephen F 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*, 51:2026–2035, 2015.

[70] The NOX and LOCA Project Team. The NOX and LOCA Project Website. `https://trilinos.github.io/nox_and_loca.html`. [Online; accessed 4-April-2022].

[71] Christian Trott, Luc Berger-Vergiat, David Poliakoff, Sivasankaran Rajamanickam, Damien Lebrun-Grandie, Jonathan Madsen, Nader Al Awar, Milos Gligoric, Galen Shipman, and Geoff Womeldorff. The Kokkos EcoSystem: Comprehensive performance portability for high performance computing. *Computing in Science Engineering*, 23(5):10–18, 2021.

[72] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, 2022.

[73] R Tuminaro, Mauro Perego, I Tezaur, A Salinger, and Stephen Price. A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling. *SIAM Journal on Scientific Computing*, 38(5):C504–C532, 2016.

[74] Jerry Watkins, Irina Tezaur, and Irina Demeshko. A study on the performance portability of the finite element assembly process within the Albany Land Ice solver. In *Numerical Methods for Flows*, pages 177–188. Springer, Cham, 2020.

[75] R Winkelmann, Maria A Martin, Monika Haseloff, Torsten Albrecht, E Bueler, C Khroulev, and Anders Levermann. The Potsdam parallel ice sheet model (PISM-PIK)-Part 1: Model description. *The Cryosphere*, 5(3):715, 2011.

[76] Charlene Yang, Rahulkumar Gayatri, Thorsten Kurth, Protonu Basu, Zahra Ronaghi, Adedoyin Adetokunbo, Brian Friesen, Brandon Cook, Douglas Doerfler, Leonid Oliker, et al. An empirical roofline methodology for quantitatively assessing performance portability. In *2018*

IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), pages 14–23. IEEE, 2018.