Performance Portable Optimizations of an Ice-sheet Modeling Code on GPU-supercomputers

Oscar Antepara, Samuel Williams
Lawrence Berkeley National Laboratory
Berkeley, USA
{oantepara, swwilliams}@lbl.gov

Max Carlson, Jerry Watkins
Sandia National Laboratories
Livermore, USA
{maxcarl, jwatkin}@sandia.gov

***Abstract*—In this paper, we present GPU-optimizations for an ice-sheet modeling code known as MPAS-Albany Land Ice (MALI). MALI is a C++ template code that leverages the Kokkos programming model for portability and the Trilinos library for data structures, nonlinear and linear solvers and optimization packages for ice-sheet simulations. Performance of the most expensive kernel is assessed via the Roofline model to highlight the potential for code improvement according to the underlying GPU architecture. We perform a collection of optimizations consisting of loop fusions, loop optimizations and local accumulation to productively and portably attain an overall speedup of 3× in either NVIDIA and AMD GPU. We analyze the performance gains using a time-oriented performance portability model based on time per invocation and GPU data movement. Results show an increment between 20% and 50% on the performance portability metric by improving data locality on the GPU kernels of a Stokes solver and highlights the importance of optimizing GPU-ported scientific applications to maximize memory bandwidth and minimize data movement on modern supercomputers.**

***Index Terms*—performance portability, GPUs, loop optimization, finite-element, Kokkos, ice-sheet modeling**

I. Introduction

Research on the Antarctic System has become a priority due to climate change and the impact of rising sea levels, which could lead to devastating global problems [1]. The development of models to compute the dynamic processes of ice sheets and making significant progress on projections for future sea-level rise pose considerable computational challenges. On this topic, high-resolution simulations are required to obtain accurate projections, and High-Performance Computing is needed to deliver faster simulations.

In today's HPC landscape, with the fastest supercomputers based on GPU architectures and with the availability of several programming models, the adoption of modern techniques to improve performance and provide portability for scientific codes that have traditionally targeted CPU-based supercomputers remain fundamental to the advancement of scientific discovery for at least the next decade. This paper presents work on portable GPU optimizations for MALI [2] (MPAS-Albany Land Ice), an ice-sheet model code that consists of MPAS (Model for Prediction Across Scales) library [3] and Albany [4], a finite element code to solve partial differential equations.

Since MALI's performance mainly relies on solving a first-order approximation of the Stokes equation, we will focus on the most expensive GPU operation in the solver and describe our effort on performance portability for NVIDIA A100 GPUs on NERSC's Perlmutter supercomputer and the AMD MI250X GPUs on OLCF's Frontier supercomputer. We will show performance gains across different GPUs by restructuring loops to improve data locality. Furthermore, we present a time-oriented performance portability model that can quantitatively evaluate performance portability for memory-bound kernels, and from which, we can assess the original implementation and our optimizations based on the application and architectural bounds.

The paper makes the following contributions: (1) it demonstrates the performance of GPU kernels in the velocity solver of an ice-sheet model on recent NVIDIA and AMD GPUs; (2) it introduces a time-oriented performance portability model based on time per invocation and GPU data movement to compare kernel implementations against the architectural and application bounds; and, (3) it demonstrates that the optimizations designed to reduce data movement can produce performance gains and are applicable across different GPUs.

II. Related Work

Research efforts on ice-sheet modeling have been focused on improving the models to represent the physical dynamics more closely to reality or to couple them with other Earth system models. As such, algorithm and code development to enable high-resolution simulations and to efficiently use high-performance systems have been the focus in the last few years.

As GPUs increasingly dominate the fastest high-performance computing systems [5], researchers have focused on restructuring algorithms to exploit GPUs to accelerate their simulations. GPU-accelerated codes for different aspects of the ice-sheet model solver using CUDA and targeting NVIDIA GPUs, where [6] evaluated performance based on a CPU to GPU comparison speedup. Additionally, [7] presented a finite difference GPU-accelerated code where they evaluated performance on several GPUs with their most recent GPU being the NVIDIA V100, although their focus is showing validation test cases and weak scalability studies. Both studies focus their efforts on porting code from CPU to GPU and examining their efforts on speedup or weak

scalability. However, as modern supercomputers are based on GPUs of different vendors, approaches based on a specific programming model or optimizations targeting a specific GPU will require extra effort to ensure portability and productivity.

Previously, MALI was analyzed on older architectures such as Intel Knights Landing (KNL) and NVIDIA V100 GPUs as in [8]. However, their work focused more on application-level improvements, where performance portability was characterized by execution time and scalability efficiencies for multicore/manycore processors and GPUs. In this paper, our work expands on these initial efforts by providing a performance evaluation of GPU kernels in the velocity solver in MALI on more recent GPUs, namely the NVIDIA A100 and AMD MI250X, and describe portable GPU optimizations that enable GPU kernel performance near the application and architectural limits.

We cite prior performance portability studies for different scientific applications on the most recent GPUs. A comparison of several mini-applications and a study about performance portability metrics to evaluate performance consistency was described in [9]. Evaluation of OpenMP on earlier GPUs, such as NVIDIA, AMD and INTEL, for a proxy of LAMMPS using the Roofline model was researched in [10] and performance portability studies also using performance efficiencies based on the Roofline model in [11]. Antepara et al. [12] evaluated stencil computations on similar GPUs as in this paper and analyzed performance portability based on fraction of roofline and fraction of theoretical arithmetic intensity. In [13], OpenMP and OpenACC were evaluated on similar GPUs and data movement at GPU device memory was evaluated across programming models and GPU architectures to highlight its implication on performance for GPU-accelerated loops in a plasma physics application. Sun and Lu [14] did a study about the memory-bounded speedup model, performance tools (Roofline model) and provided some insights about the present and future of memory systems. In this context, our work presents a methodology to evaluate performance portability for memory-bound kernels by using a time-oriented performance portability model, which can identify performance bounds to provide some insight to maximize performance.

III. MALI and Land Ice Simulation Test

MALI is an ice-sheet model built on MPAS and Albany. In practice, Albany dominates the MALI's runtime. Albany is a C++ templated finite element library that solves partial differential equations. It uses Kokkos [15], [16] as a programming model to ensure performance portability across several CPU and GPU architectures. Moreover, it integrates Trilinos [17] solvers and optimization packages and uses MPI for distributed memory parallelism. In what follows, we describe the mathematical formulation and numerical method for ice-sheet dynamics used in MALI as well as the numerical test, based on the Antarctica simulations, to evaluate performance on GPUs.

A. Mathematical Model and Numerical Method

Albany uses the first-order (FO) approximation to the nonlinear Stokes flow equations for glaciers and ice sheets [18], [19], also referred to as the Blatter–Pattyn model [20], [21]. It follows the partial differential equations:

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

where ρ is the ice density, g is the gravity acceleration, $\dot{\boldsymbol{\epsilon}}$ is the strain-rate tensor and $s \equiv s(x, y)$ denotes the upper surface boundary. The effective viscosity μ is derived from Glen's flow law [22], [23] and is also dependent on the strain rate. In addition to the FO approximation equations, a dynamic equation for mass conservation is also coupled as follows,

$$\frac{\partial H}{\partial t} + \nabla \cdot (H\bar{\boldsymbol{u}}) = \dot{a} + \dot{b}, \tag{2}$$

where H is the ice thickness, t is time, $\bar{\boldsymbol{u}}$ is the depth-averaged velocity vector, $\dot{a}$ is the surface mass balance, and $\dot{b}$ is the basal mass balance. The full details of the model and the boundary conditions for the 3D ice-sheet equations used in Albany are described in [24].

The first-order approximation model defined in Equation 1 is discretized using low-order nodal prismatic finite elements on a 3D mesh extruded from a triangulation dual to the MPAS Voronoi mesh [2]. The discretization of the velocity equations can be written in the compact form,

$$\mathcal{F}\left(U; \{\phi_i\}, \{\nabla\phi_i\}, H, ...\right) = 0, \tag{3}$$

where U is a vector containing the ice velocity values at the mesh nodes. $\{\phi_i\}$ and $\{\nabla\phi_i\}$ are the sets of basis functions and its gradients. $\mathcal{F}$ is a vector function of the solution U, which also depends on the basis functions and ice thickness H.

A damped Newton's method solves the nonlinear system where a Jacobian matrix is computed at each nonlinear iteration using automatic differentiation (AD). Moreover, the resulting linear system is solved with the GMRES method using the matrix-dependent semicoarsening algebraic multigrid (MDSC-AMG) preconditioner [25]. Details about MALI workflow for the velocity solver are described in [8].

B. Land Ice Numerical Test

MALI performance is evaluated using a series of tests consisting of high-resolution Antarctic ice sheet meshes. Numerical studies on Antarctica meshes provide a standalone simulation to analyze the velocity solver performance on GPUs and it has been previously used as a numerical test on other research papers [25], [26]. Figure 1 shows an example of a MALI production run on Perlmutter with the ice velocity solved on the GPUs.

This paper will focus on a single-node test consisting of a 16-kilometer Antarctica resolution mesh with quadrilateral

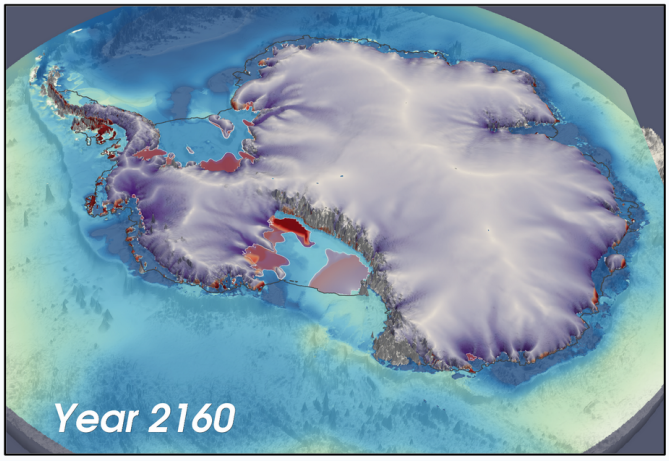

Fig. 1. Simulation snapshot of the Antarctic Ice Sheet under a high greenhouse gas emission scenario using MALI on Perlmutter with the ice velocity solved on GPUs.

elements. The mesh is extruded by 20 layers; the velocity equations are solved by the model and methods described in Section III-A, and the mean value of the final solution is compared to a previously tested value using a relative tolerance of 10^{-5} on all machines.

For this test, the number of elements on each NVIDIA A100 and one GCD AMD MI250X are fixed and are approximately 256K hexahedron elements. Additionally, the test performs a nonlinear solve consisting of eight steps, where at each nonlinear step, the linear solver operates in an iterative manner until reaching a tolerance of 10^{-6}.

IV. Experimental Setup

In this section, we provide details of the GPU-based systems used for the performance and performance portability study. We also provide details about the programming model, modules and profiling tools used in the evaluation.

A. GPU Architectures

Perlmutter [27] is an HPE Cray EX supercomputer at the National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory. Each Perlmutter GPU node consists of one AMD EPYC 7763 CPU and four of NVIDIA Ampere A100 GPUs [28]. Each GPU contains 108 streaming multiprocessors (SM), each with four warp schedulers of 16 integer units and 8 double-precision floating-point units. The GPU provides a peak performance of about 9.77 TFLOP/s of double-precision vector performance. The SMs each include a 192KB shared memory/data cache and share a 40MB L2 cache and 40GB of HBM accessible at 1.55TB/s. The GPUs are individually connected to the CPU with a PCIe 4.0 x16 link providing 32GB/s. Nodes are connected with a Slingshot 11 interconnect using four NICs per node, each providing 25GB/s/direction of bandwidth.

Frontier [29] is the HPE Cray EX supercomputer at the Oak Ridge National Laboratory. Each Frontier node contains one 64-core AMD EPYC 7A53 CPU and four AMD MI250X GPUs [30]. Each MI250X instantiates two Graphical Compute Dies (GCDs), each with 110 compute units (CU). Each CU includes four 16-wide 64b SIMD units to execute either integer or floating-point instructions and a small L1 cache. Each GCD also includes an 8MB L2 cache, provides a peak FP64 performance of about 24 TFLOP/s (vector-based double precision), and is connected to four HBM stacks of 64GB providing 1.6TB/s. Network connection between nodes uses Slingshot 11 system but the NIC is attached directly to the GCDs. From the programmer's perspective, each GCD should be targeted as if it were an independent GPU. Thus, compared to Perlmutter's A100 GPUs, each MI250X GCD provides more than twice peak FLOP rate for FP64, comparable bandwidth and 50% more memory capacity.

B. Programming Model, Compilers and Profiling Tools

Albany is a C++ finite element code that uses the Kokkos programming model and Trilinos libraries and optimization packages. Table I shows the modules and compiler version to compile, build, and execute Albany and its dependencies on Perlmutter-NERSC and Frontier-OLCF.

On Perlmutter, the GNU compiler is required to build Albany. We use NVIDIA Nsight Systems [31] to profile the Antarctica test and NVIDIA Nsight Compute [32] to profile and collect GPU performance metrics related to Roofline analysis.

Frontier offers GNU compilers and a programming environment similar to Perlmutter. We use the AMD ROCm Profiler [33] to profile and collect GPU performance metrics on AMD MI250X.

TABLE I
Programming models, modules and compiler versions used to build and test Albany on Perlmutter-NERSC, and Frontier-OLCF machines.

HPC System	Programming Model	Modules and Compiler Versions
Perlmutter NERSC	Kokkos	gcc/11.2.0, PrgEnv-gnu/8.3.3, cudatoolkit/11.7, cpe/23.03, craype/2.7.20, cray-mpich/8.1.25, e4s/23.05, NvidiaNsightSystems/23.3.1, NvidiaNsightCompute/23.2.1
Frontier OLCF	Kokkos	gcc/12.2.0, PrgEnv-gnu/8.3.3, cray-mpich/8.1.28, ROCm/5.4.3, craype/2.7.31

V. Antarctica Test GPU Profiling and GPU Optimizations

This paper focuses on the Antarctica numerical test described in Section III-B. The test was initially profiled on the NVIDIA A100 GPU, and the most time-consuming GPU kernel was identified during the evaluation of the local

BASELINE

```
template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void StokesFOResid<EvalT, Traits>::
operator() (const LandIce_3D_Tag& tag,
            const int& cell) const{
 for (unsigned int node=0; node<numNodes; ++node){
   Residual(cell,node,0)=0.;
   Residual(cell,node,1)=0.;
 }

 if (cond){
   ...
 }else{
   for (unsigned int qp=0; qp < numQPs; ++qp) {
     ScalarT mu = muLandIce(cell,qp);
     ScalarT strs00 = 2.0*mu*
                      (2.0*Ugrad(cell,qp,0,0) +
                      Ugrad(cell,qp,1,1));
     ScalarT strs11 = 2.0*mu*
                      (2.0*Ugrad(cell,qp,1,1) +
                      Ugrad(cell,qp,0,0));
     ScalarT strs01 = mu*(Ugrad(cell,qp,1,0)+
                      Ugrad(cell,qp,0,1));
     ScalarT strs02 = mu*Ugrad(cell,qp,0,2);
     ScalarT strs12 = mu*Ugrad(cell,qp,1,2);
     for (unsigned int node=0; node<numNodes;
     ++node) {
       Residual(cell,node,0) +=
         strs00*wGradBF(cell,node,qp,0) +
         strs01*wGradBF(cell,node,qp,1) +
         strs02*wGradBF(cell,node,qp,2);
       Residual(cell,node,1) +=
         strs01*wGradBF(cell,node,qp,0) +
         strs11*wGradBF(cell,node,qp,1) +
         strs12*wGradBF(cell,node,qp,2);
     }
   }
  }
  for (unsigned int qp=0; qp < numQPs; ++qp) {
    ScalarT frc0 = force(cell,qp,0);
    ScalarT frc1 = force(cell,qp,1);
    for (unsigned int node=0; node < numNodes;
    ++node) {
      Residual(cell,node,0)+=frc0*
                             wBF(cell,node,qp);
      Residual(cell,node,1)+=frc1*
                             wBF(cell,node,qp);
    }
  }
}
```

OPTIMIZED

```
template<typename EvalT, typename Traits>
template<int NumNodes>
KOKKOS_INLINE_FUNCTION
void StokesFOResid<EvalT, Traits>::
operator() (const LandIce_3D_Opt_Tag<NumNodes>& tag,
const int& cell) const{
  static constexpr int num_nodes =
  LandIce_3D_Opt_Tag<NumNodes>::num_nodes;
  ScalarT res0[num_nodes] = {};
  ScalarT res1[num_nodes] = {};
  for (size_t qp=0; qp < numQPs; ++qp) {
    ScalarT mu = muLandIce(cell,qp);
    ScalarT strs00 = 2.0*mu*(2.0*Ugrad(cell,qp,0,0)+
                     Ugrad(cell,qp,1,1));
    ScalarT strs11 = 2.0*mu*(2.0*Ugrad(cell,qp,1,1)+
                     Ugrad(cell,qp,0,0));
    ScalarT strs01 = mu*(Ugrad(cell,qp,1,0)+ U
                     grad(cell,qp,0,1));
    ScalarT strs02 = mu*Ugrad(cell,qp,0,2);
    ScalarT strs12 = mu*Ugrad(cell,qp,1,2);
    ScalarT frc0 = force(cell,qp,0);
    ScalarT frc1 = force(cell,qp,1);
    for (size_t node=0; node < num_nodes; ++node){
      res0[node]+= strs00*wGradBF(cell,node,qp,0)+
            strs01*wGradBF(cell,node,qp,1) +
            strs02*wGradBF(cell,node,qp,2) +
            frc0*wBF(cell,node,qp);
      res1[node]+=strs01*wGradBF(cell,node,qp,0)+
            strs11*wGradBF(cell,node,qp,1) +
            strs12*wGradBF(cell,node,qp,2) +
            frc1*wBF(cell,node,qp);
    }
  }
  for (size_t node=0; node < num_nodes; ++node){
    Residual(cell,node,0)=res0[node];
    Residual(cell,node,1)=res1[node];
  }
}
```

Fig. 2. Baseline and Optimized GPU kernel code listing for the `Jacobian` and `Residual` in Albany. Note that the same code is used for both with the difference that the `Jacobian` uses Sacado data structure to compute the derivatives.

`Jacobian`, which is also the same code for the computation of the `Residual` term of the first-order approximation equations. The difference between both kernels is that the `Jacobian` requires storing and computing derivative components for each local degree of freedom of the `Residual`.

Since Albany is a C++ template code that uses operator overloading, the computation and storage of derivative components in the `Jacobian` calculation are performed with Sacado data structures using automatic differentiation [34]. Sacado is an automatic differentiation package that provides multiple data structures to efficiently use expression template techniques in operator overloading-based implementation for automatic differentiation in C++. The most efficient but least flexible approach is using the Sacado data structure called SFad, where the number of derivative components is set at compile time. This approach is rather convenient since SFad could be set to 16 for the model equation and the mesh employed in this paper; for each hexahedron element that consists of 8 nodes and two velocity components, 16 derivatives are needed to compute the local `Jacobian` term.

In this work, we have improved the local `Jacobian` and `Residual` kernel performance by restructuring the GPU kernel loops to improve data locality. The optimizations performed in the Albany source code are described as follows:

Loop Optimizations: To improve loop performance in a GPU kernel, we use `size_t` and compile-time variables, instead of runtime variables, for the loop condition expression that is evaluated before each loop iteration. This optimization is more noticeable in the performance of GPU kernels with timings closer to latency values.

Loop Fusion: We merge operations under a single loop to avoid unnecessary initialization loops and multiple redundant loops that could be grouped together. This optimization minimizes the number of reads/writes on global arrays. Moreover, we take the `if` statements outside the kernel execution to avoid branch divergence and help the compiler to do a better job at optimizing the GPU kernel. Since we look to optimize ice-sheet simulations, the `if` statements are configuration-dependent and it is easy to remove them by creating a specific optimized kernel.

Local Accumulation: Given that we focus on improving the performance of memory-bound GPU kernels, we accumulate results on local arrays instead of the output global array to improve data locality, thus incurring in less data fetching from GPU global memory that has the slowest bandwidth in the GPU memory hierarchy.

Figure 2 shows the baseline and the optimized code to highlight the code differences and the optimizations described in the last paragraph. In the next section, we show the performance baseline for these kernels and the performance gains for the optimized versions on NVIDIA and AMD GPUs.

VI. Results

The performance evaluation presented in this section uses one MPI process per A100 GPU and one process per GCD on MI250X GPUs. We focus on the GPU kernel performance when computing the local `Jacobian` and `Residual` terms for the nonlinear Stokes solver on a 16-kilometer resolution Antarctic mesh on a single node. To analyze the performance results for the GPU kernels, we use the Roofline model [35] to illustrate the kernel's performance against the GPU architectural bounds limited by the GPU high memory bandwidth (HBM) and FP64 Peak Performance in TFLOPS (vector-based double precision).

Figure 3 shows the GPU performance in FLOPS/s and the arithmetic intensity (AI) in FLOPs per byte for the baseline and optimized versions of the `Jacobian` and `Residual` kernels. On the left, in figure 3, we have the performance comparison on an NVIDIA A100 GPU, and on the right, we have the performance results using one GCD on AMD MI250X GPU. Note that the baseline performance for the `Jacobian` kernel, the most time consuming kernel in the solver, is below 40% of the peak GPU memory bandwidth on both GPUs.

Observe that the optimizations applied to improve data locality accelerated both kernels on both GPUs by increasing arithmetic intensity (reducing data movement). Moreover, the optimizations on NVIDIA A100 further enhance performance by increasing memory bandwidth to 90% of peak. This effect was present, albeit smaller, on the AMD MI250X where the kernel attains 60% of peak memory bandwidth. This highlights the fact that optimizing the memory-intensive GPU kernels ubiquitous in scientific computing requires a balanced understanding of maximizing bandwidth and maximizing data locality to minimize data movement.

Given that this is the first analysis of Albany on AMD GPUs, we want to present additional features that could improve performance. Kokkos provides options to users to set execution policies that give hints to the compiler about kernel launch parameters. By using `Kokkos::LaunchBounds<MaxThreads,MinBlocks>`, we can change the workgroup size and analyze how it affects performance. Table II shows the time per invocation for the optimized `Jacobian` and `Residual` kernels, the `LaunchBounds` parameters, and we have included the number of Architectural Vector General-Purpose Registers (Arch. VGPRs) and Accumulation Vector General-Purpose Register (Accum. VGPRs) for each setting. On CDNA2 accelerators like the MI250X, there is a total of 512 VGPRs with 256 available for each type. The default values for `MaxThreads,MinBlocks` are $256, 1$ for `Jacobian` and $1024, 1$ for `Residual`.

TABLE II
TIME PER CALL, ARCHITECTURAL VGPRS, ACCUMULATION VGPRS AND SPEEDUP VS. DEFAULT VALUES FOR OPTIMIZED `JACOBIAN` AND `RESIDUAL` KERNELS ON AMD MI250X GPUS. OBSERVE THAT THE BEST PERFORMANCE IS ACHIEVED WITH `MAXTHREADS,MINBLOCKS` EQUAL TO 128,2 OR 256,2 AND WITH THE MOST USE OF VGPRS AVAILABLE.

	<MaxThreads,MinBlocks>				
Opt. Kernel	Default	128,2	128,4	256,2	1024,2
Jacobian time (s)	8.3e-2	5.4e-2	8.3e-2	5.4e-2	8.5e-2
Arch. VGPRs	128	128	128	128	128
Accum. VGPRs	0	128	0	128	0
speedup		1.54×	1×	1.54×	0.98×
Residual time (s)	2.8e-3	2.4e-3	2.6e-3	2.4e-3	3.0e-3
Arch. VGPRs	84	128	84	128	84
Accum. VGPRs	4	0	4	0	4
speedup		1.17×	1.08×	1.17×	0.94×

With a different set of `LaunchBounds`, results show that using `MaxThreads,MinBlocks` equal to either 128,2 or 256,2 results in a $1.5\times$ and $1.2\times$ speedup for the `Jacobian` and `Residual` kernel respectively. Additionally, it is important to note that the best performance is achieved when the compiler can use most of the registers available as it detects that some operations in the kernels can use architectural and accumulation VGPRs. For the NVIDIA results, there was no change in performance by varying thread block size as the default for both kernels was equal to 128.

Overall results in Table III show speedups from our optimizations ranging from $3.3\times$ for the `Jacobian` and $2.2\times$ for the `Residual` on NVIDIA A100 and $2.7\times$ for the `Jacobian` and $3.5\times$ for the `Residual` on one GCD AMD MI250X, which shows the impact of data locality for memory-bound kernels in the context of two different GPU-accelerated systems.

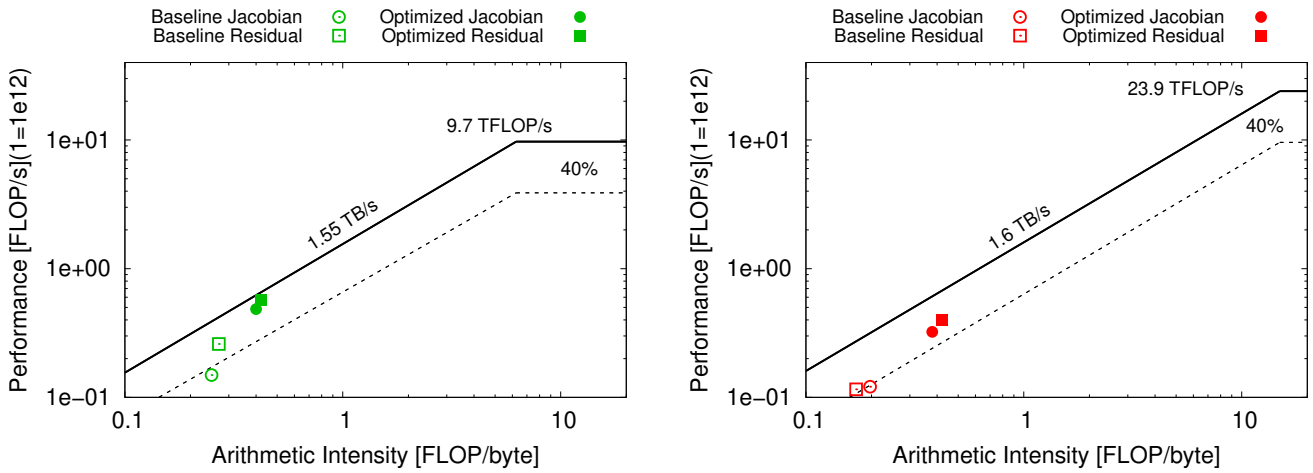

Fig. 3. Roofline for the baseline and optimized versions of the `Jacobian` and `Residual` kernels using Kokkos on NVIDIA A100 GPU (left) and a single GCD on AMD MI250X GPU (right). Notice that data locality focused optimizations for memory-bound kernels can improve the performance of already GPU-ported kernels up to approximately 3× on both GPUs.

TABLE III
OVERALL RESULTS FOR TIME PER CALL, AND SPEEDUP VS. BASELINE JACOBIAN AND RESIDUAL KERNELS ON NVIDIA A100 AND AMD MI250X GPUS. NOTE THAT WORKING ON GPU KERNEL OPTIMIZATIONS FOCUSED ON IMPROVING DATA LOCALITY CAN REDUCE THE TIME PER CALL BETWEEN 2× AND 4× FOR BOTH KERNELS AND GPUS.

Kernel	Time in seconds			
	Baseline A100	Optimized A100	Baseline GCD MI250X	Optimized GCD MI250X
Jacobian	1.2e-1	3.6e-2	1.4e-1	5.4e-2
speedup		3.3×		2.7×
Residual	3.7e-3	1.7e-3	8.3e-3	2.4e-3
speedup		2.2×		3.5×

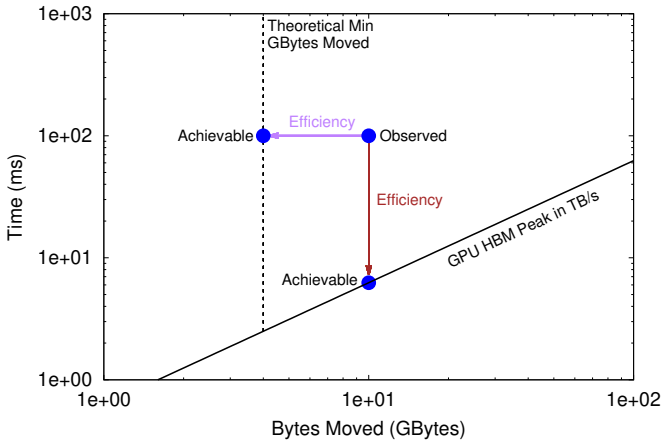

Fig. 4. Illustration of the Time-oriented Performance Portability Model based on time per invocation and bytes moved through GPU HBM. Scientific GPU kernels can be evaluated (Observed) and compared against the architectural and application bounds (Achievable). Observe that the architectural bound is defined with the GPU HBM peak and the application bound with the theoretical minimum number of bytes moved in the GPU kernel based on the number of reads/writes of the multidimensional arrays used in the GPU kernel computation.

A. *Performance Portability Model and Evaluation*

To advance research on climate modeling and improve scientific throughput, one must exploit GPU-accelerated supercomputing facilities capabilities to enable high-resolution simulations. This, in turn, orients research towards software modernization to exploit their GPUs. As such, MALI and other ice-sheet model codes are transitioning their software to support a diverse set of GPU architectures from NVIDIA, AMD, and INTEL.

Although it is imperative to implement algorithmic innovations and performance optimizations in ice-sheet models, productively exploiting the panoply of GPU architectures in a productive manner is an immense challenge. In most ice-sheet models, the time spent on executing velocity solvers dominates the run time. Performance analysis in this regard has been mostly related to improving performance and reporting the performance differences between CPU and GPU implementations. Although raw performance can be quantified through performance-oriented metrics, including FLOP/s or run-time, they are ultimately insufficient when assessing performance portability as they lack insight into the interplay between algorithmic characteristics and architectural capabilities. Thus, we aim to characterize performance portability based on architectural and application characteristics that are readily understandable by a broad spectrum of researchers beyond computer scientists and computer architecture.

Roofline-based performance analysis makes it clear that many GPU kernels in scientific modeling and simulations (and most in MALI) are memory-bound. That being said, figure 3 clearly highlights that data movement can differ substantially for the same kernel when run on different GPUs. To that end, we introduce a performance portability model that focuses on GPU kernel execution time and data movement on the GPU HBM. Figure 4 illustrates the time-oriented performance portability model for memory-bound kernels. The x-axis is data movement to/from GPU HBM in GBytes while the y-

axis is GPU kernel time per invocation in milliseconds. Finite GPU memory bandwidth sets a lower bound (diagonal line) to run time below which would imply faster-than-light execution. For streaming computations, the cache:DRAM bandwidth ratio of 1.0 vastly exceeds the cache:DRAM data movement ratio, ensuring DRAM remains the bottleneck. For non-hypersparse sparse matrix operations, typical of operations on unstructured grids, the cache:DRAM data movement ratio is dependent on the number of nonzeros per row but asymptotically approaches 1.5 — far less than the cache:DRAM bandwidth ratio and ensuring computations bottlenecked by DRAM bandwidth. As the A100 GPU and an MI250X GCD have comparable memory bandwidth, we can plot both architectures on a single figure using a common bandwidth lower bound. Ultimately, our approach is similar to the example in [36], where a time-oriented Roofline was introduced based on execution time and arithmetic intensity but presumes all kernels are ultimately memory-bound. We augment this methodology by defining "walls" based on a theoretical analysis of data movement given the size of arrays a GPU kernel operates on and finite cache capacity. No degree of optimization for a given GPU kernel would ever allow that kernel to move less data than this theoretical minimum on data movement. For each kernel, we may define three coordinates that may be plotted in this data movement-run time plane: 1) measured data movement and run time for the baseline implementation, 2) measured data movement and run time for the optimized implementation, and 3) theoretical minimum data movement and theoretical minimum run time (the ratio of theoretical minimal data movement and memory bandwidth).

Figure 5 presents the time-oriented performance model with the measured values of time per invocation in milliseconds and GBytes moved through GPU HBM for the baseline and optimized `Jacobian` and `Residual` kernels. One of the advantages of the time-oriented model is that it can illustrate the comparison between different kernels and GPU architectures. For the ice-sheet model, computing the theoretical GBytes moved is relatively easy since most operations (reads/writes) are based on arrays defined according to the numerical method. In this case, all arrays are multidimensional based on the number of elements, dimensions, quadrature points, number of derivatives, etc. As described in Section V, the `Jacobian` kernel uses the Sacado data structure since it requires a set of derivatives for its computation. Therefore, the `Jacobian` kernel is expected to move 16 times more data compared to the `Residual` kernel, as shown in Figure 5.

Results show the lack of data locality on the baseline implementation. Conversely, the optimized implementations are very close to the theoretical application bounds based on GBytes moved. This means we have achieved nearly optimal data movement for both kernels and architectures. It also illustrates that by improving data locality, we can gain comparable speed-ups up to $3\times$ on both architectures and get closer to the architectural limit defined by the peak GPU HBM.

We now explore the performance portability metric for the ice-sheet model code. Note that we follow the performance portability metric evaluation described in [37], [38]. We define the efficiencies as the comparison of the measured values (observed) in time per invocation and GBytes moved in the GPU HBM against the architectural and application bounds (achievable), respectively, as illustrated in Figure 4. Specifically, we define performance portability $\mathcal{P}$, given a set of platforms H for an application a solving problem p is:

$$\mathcal{P}(a,p,H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a,p)}}, & \text{if } i \text{ is supported } \forall i \in H \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $e_i(a,p)$ is the efficiency. We have defined two efficiencies. The first is based on time per invocation relative to the theoretical lower limit, while the second is based on GPU HBM data movement in bytes relative to the theoretical lower limit as described in Figure 4. Performance efficiency based on the time per invocation represents the ability of the implementation to achieve better GPU architecture usage while maintaining the same GPU HBM data movement. Notice that using time per invocation, as an efficiency, gives a direct insight that most research scientists also use as a performance metric on several scientific domains. Additionally, the performance efficiency based on the GPU HBM data movement represents the implementation ability to incur minimal GPU data movement for the application based on the theoretical minimum calculated for the GPU kernel. This efficiency highlights better data reuse or locality, and it could also be seen as a lower bound that is architecture-independent.

TABLE IV
PERFORMANCE PORTABILITY METRIC $\mathcal{P}$ BASED ON TIME PER INVOCATION (e_{time}) AND GPU HBM DATA MOVEMENT (e_{DM}) EFFICIENCIES. GPU OPTIMIZATIONS SHOW AN IMPROVEMENT BETWEEN 20% AND 50% IN $\mathcal{P}$ DEPENDING ON THE EFFICIENCY.

	Efficiency	Kernel	A100	1 GCD MI250X	$\mathcal{P}$
Baseline	e_{time}	Jacobian	39%	38%	39%
	e_{time}	Residual	62%	42%	50%
	e_{DM}	Jacobian	53%	42%	47%
	e_{DM}	Residual	65%	41%	50%
Optimized	e_{time}	Jacobian	79%	53%	63%
	e_{time}	Residual	88%	60%	71%
	e_{DM}	Jacobian	84%	81%	83%
	e_{DM}	Residual	100%	100%	100%

Table IV shows the efficiencies for the baseline and the optimized `Jacobian` and `Residual` kernels. Moreover, we present the performance portability metric, $\mathcal{P}$, based on Equation 4. We observe that the GPU optimizations to improve data locality have a significant impact on the GPU HBM data movement efficiency (e_{DM}), going from 42-41% to 81-100% on AMD MI250X and from 53-65% to 84-100% on NVIDIA A100. It is clear that achieving e_{DM} closer to 100% is a realistic goal for the `Residual` kernel that moves approximately $16\times$ less data than the `Jacobian` kernel. Additionally, the time per invocation efficiency (e_{time}) goes from closer to 40% on either kernel to 53-60% on one MI250X GCD and from

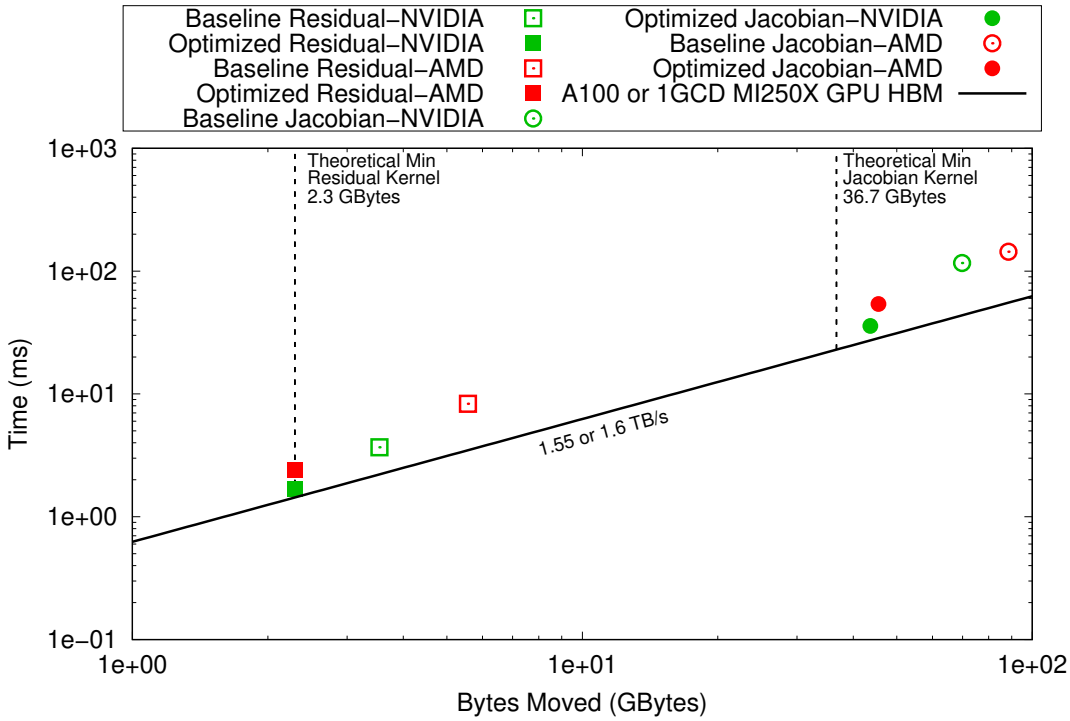

Fig. 5. Time-oriented Performance Portability Model for the baseline and optimized `Jacobian` and `Residual` kernels on NVIDIA A100 and AMD MI250X GPUs. Note that the optimizations focused on data locality reduce the time per invocation and get both kernels closer to the application bound based on GPU HBM data movement on both architectures.

39-62% to 79-88% on A100, with a significant increase on NVIDIA GPU compared to AMD GPU that demonstrates the compiler ability to get the best performance out of the kernel once data locality has been improved. The performance portability metric is also shown for the baseline and optimized kernels computed with the results of both GPU architectures. Overall, the performance portability metric increased by 20% and 30-50% for time per invocation and the GPU HBM data movement, respectively.

VII. Discussion and Future Work

GPU optimizations and portability are among the primary challenges for research scientists looking to improve their software and efficiently use the new GPU-accelerated supercomputing facilities. As we prepare our codes to transition to new GPU architectures, evaluating performance or fully exploiting GPUs from different vendors for relatively large software applications is a path forward.

This paper explores GPU portable optimizations for MALI (MPAS-Albany Land Ice), an ice-sheet model code that consists of MPAS (Model for Prediction Across Scales) library, and Albany, a finite element code to solve partial differential equations. Like most ice-sheet models, most of the computational time relies on the velocity solver, part of the Albany source code. Here, we have shown our optimization efforts on the most expensive GPU operation in the solver and described our efforts on performance portability for NVIDIA A100 GPUs and AMD MI250X GPUs.

To improve data locality and performance on GPUs, we restructured the kernel computation by fusing loops, avoiding branch divergence, using compile-time variables for the loop condition expressions, and using local arrays instead of accumulating the results directly on the global array.

Results show that our optimizations can improve kernel performance, measured in time per invocation, compared to the baseline implementation. Effectively implementing those optimizations that work on two different GPU architectures but with similar characteristics provides a step forward in writing efficient GPU portable code. On NVIDIA A100 GPU, our optimizations demonstrated a speedup of $3.3\times$ and $2.2\times$ for the `Jacobian` and `Residual` kernels. On one GCD AMD MI250X GPU, the performance gains were $2.7\times$ and $3.5\times$ for both kernels. These gains are due mainly to improving data locality and avoiding redundant loop operations compared to the baseline implementation. On AMD GPUs, we also improve performance by changing the Kokkos default values for launch bounds, where setting the best workgroup size for these kernels increased the performance between $1.17\times$ and $1.54\times$ on one GCD AMD MI250X GPU.

In our work, we also introduce a time-oriented performance portability model. It uses time per invocation and GPU HBM data movement as efficiencies to analyze performance portability across different GPUs. For memory-bound kernels, time per invocation represents the code's ability to use the GPU resources efficiently, where its peak is the GPU HBM bandwidth. On the other hand, GPU HBM data movement

demonstrates data reuse or data locality in the GPU kernel, where a theoretical lower bound can be derived from the array sizes and the number of reads/writes done during the computation. Additionally, performance portability metrics can be computed from those efficiencies across a diverse set of GPU architectures. Our optimizations demonstrated an increment between 20% and 50% on performance portability, with a significant increase in the GPU HBM data movement efficiency, indicating that our optimizations improved data locality for both NVIDIA and AMD GPUs.

Future work will continue our efforts to optimize the velocity solver for GPUs and explore portability on INTEL GPUs. We will use our performance portability model to evaluate several kernels and to provide an insightful way to evaluate our portability efforts on GPU-accelerating the code. Moreover, we will conduct scalability studies and explore techniques and new performance models to improve productivity and portability for large-scale simulations in the context of ice-sheet models.

ACKNOWLEDGMENT

The authors thank Trevor Hillebrand and Matt Hoffman from Los Alamos National Laboratory for contributing Figure 1.

This research was supported by the U.S. Department of Energy, Office of Science, SciDAC/Advanced Scientific Computing Research under Award Number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

APPENDIX

The paper focuses on the performance-portable GPU optimizations of the most time consuming kernels in the velocity solver of an ice-sheet model code on NVIDIA A100 and AMD MI250X. Here, we describe the hardware and software used to GPU optimize the most time consuming kernels and the methodology used to extract the data needed for GPU data movement included in the paper.

Machines: Results presented in this paper were obtained on a NVIDIA A100 GPU on Perlmutter at NERSC, and AMD MI250X GPU on Frontier at OLCF. In all experiments, we use only a single process running on one A100 GPU and one GDC on an MI250X GPU.

In this section, we describe the command lines given to the profiler to gather GPU metrics for the GPU data movement in the figures described in the paper.

A. Perlmutter-NERSC

On Perlmutter-NERSC, NVIDIA Nsight Compute [32] command line to gather GPU metrics for double precision is depicted below:

```
nv-nsight-cu-cli -k <kernel_name>
--metrics  "dram__bytes.sum" <exe> <param>
```

The metric `dram__bytes.sum` gives the value for GPU data movement presented in the paper. Another NVIDIA profiler to get the roofline plots and GPU data movement is NVIDIA Nsight Compute by using the next command line:

```
srun -n 1 ncu -o output_file
--set full <exe> <params>
```

where results in the $output\_file$ can be visualized using the Nsight Compute API on your local machine.

As scientific applications are rather complex and launch multiple kernels, we can also specify the kernel name for NVIDIA Nsight Compute by including the next command:

```
--kernel-name-base=demangled
-k regex:"KernelName"
```

Finally, GPU data movement can be found in the `Memory Workload Analysis` from the output report.

B. Frontier-OLCF

AMD-rocProf [33] is the profiler available on Frontier-OLCF to collect GPU metrics. The command line used is:

```
rocprof -i  input_file.txt --timestamp on
-o my_output.csv <exe> <params>
```

AMD ROCm Profiler needs an input file with the kernel name and the metrics to be collected. An example of the input file is shown below:

```
kernel: <kernel_name>
pmc : SQ_INSTS_VALU_ADD_F16 SQ_INSTS_VALU_MUL_F16
SQ_INSTS_VALU_FMA_F16 SQ_INSTS_VALU_TRANS_F16
pmc : SQ_INSTS_VALU_ADD_F32 SQ_INSTS_VALU_MUL_F32
SQ_INSTS_VALU_FMA_F32 SQ_INSTS_VALU_TRANS_F32
pmc : SQ_INSTS_VALU_ADD_F64 SQ_INSTS_VALU_MUL_F64
SQ_INSTS_VALU_FMA_F64 SQ_INSTS_VALU_TRANS_F64
pmc : SQ_INSTS_VALU_MFMA_MOPS_F16
SQ_INSTS_VALU_MFMA_MOPS_BF16
SQ_INSTS_VALU_MFMA_MOPS_F32
SQ_INSTS_VALU_MFMA_MOPS_F64
pmc : TCC_EA_RDREQ_32B_sum TCC_EA_RDREQ_sum
TCC_EA_WRREQ_sum TCC_EA_WRREQ_64B_sum
gpu: 0
```

To compute GPU data movement, we use the `rocprof` metrics:

```
TCC_EA_WRREQ_64B, TCC_EA_WRREQ_sum,
```

`TCC_EA_RDREQ_32B`,
`TCC_EA_RDREQ_sum`.

Moreover, we compute GPU data movement with the next formula:

```
GPU Bytes Moved = 64* TCC_EA_WRREQ_64B +
32*(TCC_EA_WRREQ_sum - TCC_EA_WRREQ_64B)
+ 32*TCC_EA_RDREQ_32B +
64*(TCC_EA_RDREQ_sum - TCC_EA_RDREQ_32B).
```

In the CSV report, we can also find the Architectural and Accumulation Vector General-Purpose Registers as `arch_vgpr` and `accum_vgpr`, respectively.

More information can be found here: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#getting-started-with-the-rocm-profiler.

REFERENCES

[1] I. P. on Climate Change (IPCC), *Ocean, Cryosphere and Sea Level Change*. Cambridge University Press, 2023, p. 1211–1362.

[2] M. J. Hoffman, M. Perego, S. F. Price, W. H. Lipscomb, T. Zhang, D. Jacobsen, I. Tezaur, A. G. Salinger, R. Tuminaro, and L. Bertagna, "Mpas-albany land ice (mali): a variable-resolution ice sheet model for earth system modeling using voronoi grids," *Geoscientific Model Development*, vol. 11, no. 9, pp. 3747–3780, 2018. [Online]. Available: https://gmd.copernicus.org/articles/11/3747/2018/

[3] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud, "A multi-resolution approach to global ocean modeling," *Ocean Modelling*, vol. 69, pp. 211–232, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1463500313000760

[4] A. G. Salinger, R. A. Bartlett, A. M. Bradley, Q. Chen, I. P. Demeshko, X. Gao, G. A. Hansen, A. Mota, R. P. Muller, E. Nielsen, J. T. Ostien, R. P. Pawlowski, M. Perego, E. T. Phipps, W. Sun, and I. K. Tezaur, "Albany: Using component-based design to develop a flexible, generic multiphysics analysis code," *International Journal for Multiscale Computational Engineering*, vol. 14, no. 4, pp. 415–438, 2016.

[5] TOP 500, "Top 500 website," https://www.top500.org/, 2024.

[6] C. F. Brædstrup, A. Damsgaard, and D. L. Egholm, "Ice-sheet modelling accelerated by graphics cards," *Computers & Geosciences*, vol. 72, pp. 210–220, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009830041400185X

[7] L. Räss, A. Licul, F. Herman, Y. Y. Podladchikov, and J. Suckale, "Modelling thermomechanical ice deformation using an implicit pseudo-transient method (fastice v1.0) based on graphical processing units (gpus)," *Geoscientific Model Development*, vol. 13, no. 3, pp. 955–976, 2020. [Online]. Available: https://gmd.copernicus.org/articles/13/955/2020/

[8] J. Watkins, M. Carlson, K. Shan, I. Tezaur, M. Perego, L. Bertagna, C. Kao, M. J. Hoffman, and S. F. Price, "Performance portable ice-sheet modeling with MALI," *The International Journal of High Performance Computing Applications*, vol. 37, no. 5, pp. 600–625, 2023. [Online]. Available: https://doi.org/10.1177/10943420231183688

[9] J. Kwack, J. Tramm, C. Bertoni, Y. Ghadar, B. Homerding, E. Rangel, C. Knight, and S. Parker, "Evaluation of performance portability of applications and mini-apps across amd, intel and nvidia gpus," in *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2021, pp. 45–56.

[10] N. A. Mehta, R. Gayatri, Y. Ghadar, C. Knight, and J. Deslippe, "Evaluating performance portability of openmp for snap on nvidia, intel, and amd gpus using the roofline methodology," in *Accelerator Programming Using Directives*, S. Bhalachandra, S. Wienke, S. Chandrasekaran, and G. Juckeland, Eds. Cham: Springer International Publishing, 2021, pp. 3–24.

[11] C. Bertoni, J. Kwack, T. Applencourt, Y. Ghadar, B. Homerding, C. Knight, B. Videau, H. Zheng, V. Morozov, and S. Parker, "Performance portability evaluation of opencl benchmarks across intel and nvidia platforms," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 330–339.

[12] O. Antepara, S. Williams, H. Johansen, T. Zhao, S. Hirsch, P. Goyal, and M. Hall, "Performance portability evaluation of blocked stencil computations on gpus," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1007–1018. [Online]. Available: https://doi.org/10.1145/3624062.3624177

[13] O. Antepara, S. Williams, S. Kruger, T. Bechtel, J. McClenaghan, and L. Lao, "Performance-portable gpu acceleration of the efit tokamak plasma equilibrium reconstruction code," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1939–1948. [Online]. Available: https://doi.org/10.1145/3624062.3624607

[14] X. Sun and X. Lu, "The memory-bounded speedup model and its impacts in computing," *Journal of Computer Science and Technology*, vol. 38, pp. 64–79, 2023.

[15] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731514001257

[16] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 805–817, 2022.

[17] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, p. 397–423, sep 2005. [Online]. Available: https://doi.org/10.1145/1089014.1089021

[18] J. K. Dukowicz, S. F. Price, and W. H. Lipscomb, "Consistent approximations and boundary conditions for ice-sheet dynamics from a principle of least action," *Journal of Glaciology*, vol. 56, no. 197, p. 480–496, 2010.

[19] C. Schoof and R. C. A. Hindmarsh, "Thin-Film Flows with Wall Slip: An Asymptotic Analysis of Higher Order Glacier Flow Models," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 63, no. 1, pp. 73–114, 01 2010. [Online]. Available: https://doi.org/10.1093/qjmam/hbp025

[20] H. Blatter, "Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients," *Journal of Glaciology*, vol. 41, no. 138, p. 333–344, 1995.

[21] F. 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*, vol. 108, no. B8, 2003. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2002JB002329

[22] K. Cuffey and W. Paterson, *The Physics of Glaciers*. 4th edition, Butterworth-Heinemann, 2010.

[23] J. F. Nye, "The distribution of stress and velocity in glaciers and ice-sheets," in *Proc. R. Soc. Lond. A*, vol. 239, 1957, p. 113–133.

[24] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. 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*, vol. 8, no. 4, pp. 1197–1220, 2015. [Online]. Available: https://gmd.copernicus.org/articles/8/1197/2015/

[25] R. Tuminaro, M. Perego, I. Tezaur, A. Salinger, and S. Price, "A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. C504–C532, 2016. [Online]. Available: https://doi.org/10.1137/15M1040839

[26] I. K. Tezaur, R. S. Tuminaro, M. Perego, A. G. Salinger, and S. 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*, vol. 51, pp. 2026–2035, 2015, international Conference On Computational Science, ICCS 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915012752

[27] NERSC, "NERSC: Perlmutter gpu nodes," 2024. [Online]. Available: https://docs.nersc.gov/systems/perlmutter/architecture/
[28] NVIDIA, "NVIDIA A100 GPU architecture," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020.
[29] OLCF, "OLCF: Frontier GPU nodes," 2024. [Online]. Available: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html
[30] AMD, "AMD CDNA 2 architecture," https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf, 2022.
[31] NVIDIA, "NVIDIA Nsight Systems documentation," https://docs.nvidia.com/nsight-systems/UserGuide/index.html, 2024.
[32] ——, "NVIDIA Nsight Compute documentation," https://docs.nvidia.com/nsight-compute/NsightCompute/index.html, 2024.
[33] AMD, "AMD rocProf documentation," https://rocm.docs.amd.com/projects/rocprofiler/en/latest/how-to/using-rocprof.html\#using-rocprof, 2024.
[34] E. Phipps and R. Pawlowski, "Efficient expression templates for operator overloading-basedautomatic differentiation," in *Recent Advances in Algorithmic Differentiation*, S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 309–319.
[35] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, apr 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785
[36] S. Williams, "Auto-tuning performance on multicore computers," https://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-164.pdf, 2008.
[37] S. Pennycook, J. Sewall, and V. Lee, "Implications of a metric for performance portability," *Future Generation Computer Systems*, vol. 92, pp. 947–958, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17300559
[38] S. J. Pennycook and J. D. Sewall, "Revisiting a metric for performance portability," in *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2021, pp. 1–9.