Veris: Fast & Efficient Sea-Ice Modeling in Python with GPU Acceleration

Jan P. Gärtner[1], Martin Losch[1], Suvarchal K. Cheedela [1], Markus Jochum[2], and Roman Nuterman[2]

[1]Alfred-Wegener-Institut, Helmholtz Zentrum für Polar- und Meeresforschung, Bremerhaven, Germany
[2]Niels Bohr Institute, University of Copenhagen, Copenhagen, Denmark

Correspondence: Jan P. Gärtner (jan.gaertner@awi.de)

Abstract. Climate models ~~are typically~~ have traditionally been developed in Fortran, due to its long-standing use in scientific computing and its excellent computational performance. While Python offers ~~significant~~ substantial advantages in terms of code readability, maintainability, and the availability of libraries and tools, the performance gap between Python and Fortran has historically limited Python's use in ~~large scale climate simulations. The JAX library for array based computing addresses this gap by enabling Just-In-Time compilation~~large-scale climate modeling. This performance gap can be mitigated by using the JAX library, which significantly ~~enhances execution speed . As climate models iterate over numerous time steps, using compiled functions substantially increases the performance of the model. Furthermore, JAX supports both CPU and GPU execution, allowing models to leverage the high parallelism of GPUs. Given that climate models are highly parallelizable, GPUs offer a more efficient alternative to CPUs, both in terms of performance and energy consumption, thereby reducing the computational carbon footprint.~~ improves execution speed of Python code.

~~This work presents~~ We use JAX as a backend for Veris, a new sea ice model ~~that is a Fortran to Pythontranslation of the~~ implemented in Python. Veris builds upon the Fortran-based sea ice component of the general circulation model MITgcm. Benchmark ~~tests on a grid with 10^6 grid cells~~ experiments show that Veris ~~with JAX as the backend is only 1.7 times slower than the Fortran reference . When running~~ exhibits scaling behavior with increasing process counts comparable to the Fortran reference implementation. For small CPU process counts, Veris outperforms the MITgcm, showing the great potential that JAX has for climate modeling, particularly as further improvements in inter-process communication are anticipated. When executed on a high-end GPU, ~~Veris~~ a single-process Veris simulation matches the performance of the parallelized Fortran reference running on ~~45 CPUs on 45 nodes or 224 CPUs on 2 nodes. Additionally~~hundreds of CPU cores, but at a fraction of the energy cost. These results demonstrate the potential of Veris for large-scale HPC-based simulations. Furthermore, Veris can be coupled with the Python-based ocean model Veros to form a fully Python-based coupled sea ice-ocean model~~, that can be used with large grids in HPC-based simulations~~.

Plain Language Summary

Climate simulations help us understand the Earth system and ~~the evolution of its various components. These insights are crucial for informing policies and combating climate change. The climate~~ its evolution. The models used to perform these simulations

are highly complexand , require significant programming expertise to build and modify. Because climate models simulate long timescales on large grids, they take a long time to run and consume a lot of energy. A key component of climate models is their sea ice components. In this work, we present a sea ice model that offers an easier development process and workflow compared to traditional models while still maintaining competitive while maintaining strong performance. By utilizing using GPUs, our model can replace hundreds of CPUs, significantly reducing the energy usageof running climate simulationsits energy usage.

1 Introduction

Climate models are typically written in contain hundreds of thousands of lines of code. The programming language used in these models is typically Fortran, largely due to its legacy in scientific computing. Fortran's strong backwards compatibility allows extensive reuse of existing code in newer models written in updated versions of the language, saving significant development time compared to rewriting legacy code in a different language. Another critical reason for Fortran's prevalence is its high computational efficiency, as it is optimized for the speed of numerical calculations (Méndez et al., 2014). Climate models require the execution of an extensive number of floating-point operations over many time steps and across large spatial grids, making computational performance particularly essential for this field.

Despite these advantages, Fortran is relatively unpopular compared to modern languages like Python, especially among scientists outside climate modeling. This can create a barrier for new researchers, who have to invest substantial time in learning an unfamiliar programming language. Additionally, using a more popular language such as Python would enable access to more libraries and tools. Fortran, though well-suited for high-performance computing (HPC) due to its optimization for numerical and array-based calculations, can be more challenging to work with than higher-level languages. Using a high-level language such as Python would not only facilitate structural modifications, such as implementing new parameterizations and simplifying debugging, but would also make HPC workflows more accessible to researchers who may lack extensive experience in low-level programming.

Although Python's popularity and readability offer advantages, these factors alone do not justify switching from Fortran to Python due to the significant performance gap between the two languages without optimizations or extensions. To overcome this gap, the Python library JAX can be used to significantly increase Python's performance for numerical modeling (Bradbury et al., 2018). JAX is a library for array-based computing with a syntax very similar to NumPy's (Harris et al., 2020), allowing NumPy code to be adapted to JAX without structural changes. JAX includes a set of function transformations like vectorization, parallelization, automatic differentiation and Just-In-Time (JIT) compilation. The JIT compilation significantly reduces the runtime of iterative calculations, making JAX well-suited for climate models as they iterate over many time steps.

In addition to these transformations, a key advantage of JAX is that it is agnostic to the type of accelerator, enabling the same code to run on both CPUs and GPUs. Due to their higher number of corescore counts, GPUs are much more better suited for parallel computing than CPUsand are therefore , provided that the available computational and memory resources are fully utilized. As a result, they are more energy-efficient for parallel tasks such as climate simulations, as they complete these simulations faster than a CPUCPUs. By leveraging GPUs, JAX not only enhances computational speed but also reduces the

power consumption and therefore the carbon footprint of running climate models (e.g. Marquetti et al., 2018; Portegies Zwart, 2020; Li et al., 2023).

As a recent example, the Python based Versatile Ocean Simulator (Veros, Häfner et al., 2018, https://veros.readthedocs.io/en/latest/) is a translation of the Fortran backend of the ocean model pyOM2 (v2.1.0, Eden and Olbers, 2014; Eden, 2016). By using the Just-In-Time compilation of JAX, the performance gap to the original Fortran model was closed (Häfner et al., 2021). When using JAX as the backend, Veros can be run on both CPUs and GPUs. With a single GPU, dozens to hundreds of CPU cores can be replaced which substantially reduces the energy consumption of running Veros.

To date, Veros does not contain a sea ice model, limiting its applicability in high-latitude regions where sea ice significantly influences heat, momentum, and moisture exchanges between the ocean and atmosphere, thereby impacting the local climate. This work introduces Veris (Versatile Sea Ice Simulator), a Python-based sea ice model ~~that can be coupled with Veros to broaden~~ originally developed to be coupled to Veros to extend its range of applications. Like Veros, Veris ~~utilizes~~ supports both NumPy and JAX ~~as~~ backends. When ~~using~~ used with JAX, Veris can also run on ~~a GPU , enabling~~ GPU architectures, providing substantial computational advantages. In order to parallelize Veris, the model infrastructure was subsequently redesigned, resulting in a separate, updated version of Veris that is able to leverage JAX's parallelization capabilities on both CPU- and GPU-based systems. Owing to these architectural changes, this parallelized version is no longer directly compatible with Veros, but instead represents an independent, high-performance sea ice modeling framework optimized for large-scale simulations.

To solve the momentum equation, Veris employs the modified and adaptive Elastic-Viscous-Plastic (EVP) solver (Hunke and Dukowicz, 1997; Bouillon et al., 2013; Kimmritz et al., 2016). Benchmark experiments by Rasmussen et al. (2024) assessed a refactored ~~version~~ implementation of the standard EVP solver ~~as~~ used in the ~~Fortran based~~ Fortran-based sea ice model CICE ~~?~~(Hunke et al., 2024), optimized for ~~the execution on multiple CPUs as well as on GPUs. The EVP solver~~ parallel execution on both CPU- and GPU-based architectures. In these experiments, execution on a single GPU was found to be 1.2 times faster than on 112 ~~CPUs~~CPU cores. Accordingly, a significant performance increase is anticipated for Veris when comparing GPU-basedexecution to CPU-based ~~execution~~configurations.

The properties of Veris are described in section 2. Section 3 presents benchmark simulations to verify that Veris accurately simulates sea ice dynamics, as well as performance benchmarks comparing Veris to the Fortran reference.

2 Model Description

2.1 Basic Properties

Veris is a Fortran to Python translation of part of the sea ice component of the Massachusetts Institute of Technology general circulation model (MITgcm, Marshall et al., 1997; Losch et al., 2010, https://github.com/MITgcm/MITgcm, https://mitgcm.org/documentation/), originally based on the formulation by Hibler (1979). The core properties of Veris are as follows:

- The variables are stored on a two-dimensional, staggered C-grid (Arakawa and Lamb, 1977). Tracer variables, such as ice thickness and concentration, are stored at the centers of grid cells, while the u- and v-components of velocity are stored at the respective u- and v-grid points.

- Ice grows thermodynamically when the ocean loses heat to the atmosphere through a conductive heat flux across the ice. This heat loss is balanced by freezing at the ice bottom. Additionally, new ice may form in regions of open water when the ocean is loosing heat. For the growth rate calculation, a two level thickness configuration is assumed (Hibler, 1979). Each grid cell is assumed to contain ice of thickness h_{actual} and open water. The proportion of the grid cell that is covered with ice is defined as the ice concentration A. The actual ice thickness h_{actual} is derived from the mean ice thickness per grid cell h according to $h_{actual} = \frac{h}{A}$. The growth of the mean thickness over a single time step is

$$\Delta h = \Delta t \Big(f(h_{actual})A + f(0)(1-A) + f_{ml} \Big) \tag{1}$$

  where $f(h)$ is the growth rate of ice of thickness h, and f_{ml} accounts for ice melting at the base due the mixed layer temperature.

- The ice has no heat capacity, and the adjustment of the ice surface temperature to changes in the surrounding temperature occurs instantaneously ~~(zero-layer thermodynamics, ?)~~(zero-layer thermodynamics, Semtner, 1976).

- Snow can accumulate on the ice cover under conditions of snowfall or freezing rain.

- Additional ice growth can occur if the weight of accumulated snow submerges the ice. When the ice-snow interface lies below the water surface, a simple mass conserving parameterization of snow-ice formation turns snow into ice until the ice surface is back at the water surface (Leppäranta, 1983).

- A viscous-plastic rheology (Hibler, 1979) is implemented using the explicit solution methods of the modified elastic-viscous-plastic solver ~~(mEVP, Bouillon et al., 2013)~~ (EVP*, revisited EVP or mEVP, Lemieux et al., 2012; Bouillon et al., 2013; Ki and the adaptive elastic-viscous-plastic solver (aEVP, Kimmritz et al., 2016). For small internal ice stresses, the ice flow is viscous and the internal stress increases linearly with the strain rate. Once a certain stress threshold is reached, the ice deforms plastically, such that the internal stress does not exceed the yield criterion. Both the mEVP solver and the aEVP solver performing a specified number of subcycling iterations to update the ice stress and velocity fields at each time step 2.3). In the aEVP solver, the relaxation parameters for the subcycling process are computed based on the grid cell size and the viscosities of ice, resulting in a faster convergence in most grid points.

- In addition to the EVP solver, a freedrift solver is implemented, which does not account for ice stress, allowing the ice to drift independently of the presence of ice in adjacent grid cells.

- Landfast ice is sea ice that remains immobile or nearly immobile and attached to a coast for a certain time period. The ice cover may be fixed to the seafloor by down-reaching ice keels or laterally anchored to a coast. The formation of landfast ice due to the ice being anchored to the seafloor in shallow waters is parameterized using a basal drag

coefficient (Lemieux et al., 2015). For landfast ice formed due to the ice being fixed to a nearby coast, a coastal drag parameterization (Liu et al., 2022) is implemented. This coastal drag is applied in conjunction with a free-slip boundary condition. Additionally, a no-slip boundary condition is also implemented.

~~It~~ In the Veros-compatiable version of Veris, it is possible to switch between NumPy and JAX as the backend~~when running Veris~~, which facilitates the development and implementation of new features. Debugging within JAX can be challenging due to its use of just-in-time compilation, which restricts access to intermediate outputs in compiled functions. By initially debugging with NumPy — where intermediate outputs and clearer error messages are available — developers can efficiently implement and debug new code before switching to JAX to take advantage of its optimized performance.

2.2 ~~Validation~~Code Verification

To ~~ensure the accuracy~~ verify the correctness of the translation from Fortran to Python, we conducted a comparative analysis of model outputs from Veris and the MITgcm~~sea ice component~~. The thermodynamic and dynamic ~~processes were evaluated independently~~components were evaluated separately. For the ~~thermodynamics component, we employed~~ thermodynamic component, a one-dimensional benchmark ~~to simulate the evolution of~~ was employed, simulating the temporal evolution of sea ice thickness, snow thickness, and ice concentration within a single grid cell. This ~~approach is feasible~~ setup is sufficient for thermodynamic verification because ~~the ice growth processes in this model are solely influenced by the local conditions of the sea ice and atmospheric forcing within the grid cell itself, without dependence on adjacent~~ thermodynamic growth and melt processes depend exclusively on local state variables and atmospheric and oceanic forcing, without lateral coupling to neighboring grid cells.

The ~~thermodynamics benchmark tests included~~ thermodynamic benchmarks covered a range of initial conditions for ~~sea ice and snow thickness~~ice thickness, snow thickness, and ice concentration, ~~along with~~ as well as variations in atmospheric and oceanic forcing~~conditions, such as air and~~ , including air temperature, ocean surface temperature, precipitation, and ~~both~~ shortwave and longwave radiation. Simulations were ~~conducted over~~ performed for 365 ~~iterations~~ time steps with a daily ~~timestep, resulting in relative differences~~ time step. Differences between Veris and the MITgcm remained on the order of ~~10^{-7} for ice thickness, snow thickness, and ice concentration.These differences were minor, non-accumulative, and~~ $\mathcal{O}(10^{-7})$ for all prognostic variables and exhibited random, non-accumulative behavior (Fig. A1). Small differences are expected in code porting due to a variety of reasons, including differences in compiler-dependent ordering of operations, intrinsic library implementations, and floating-point arithmetic. The bounded and non-growing nature of the differences indicates that the translated thermodynamic formulation represents a stable, non-divergent system. The deviations are therefore attributed to numerical precision limitations (Rosinski and Williamson, 1997).

For the dynamics component, a two-dimensional benchmark ~~was conducted~~following Mehlmann et al. (2021) was conducted on rectangular grids with resolutions of 128×128, 256×256, 512×512, and 1024×1024 grid cells. As in the thermodynamic ~~benchmark~~tests, varying initial conditions for ice ~~and snow thickness~~thickness, snow thickness, and ice concentration were employed, along with variations in ~~the forcing fields, including wind and ocean surface velocity~~atmospheric and oceanic forcing.

In contrast to the thermodynamic component, the ~~dynamical component of a sea ice model requires a short time step due to stability constraints of the dynamics solver and accuracy considerations~~sea ice dynamics solver is subject to stricter stability and accuracy constraints, requiring shorter time steps. Accordingly, a time step of 10 minutes was ~~employed throughout~~ used for 1000 ~~iterations. The resulting relative differences between the models were~~ time steps.

The dynamical evolution of the sea ice field is substantially more sensitive to small perturbations than the thermodynamic evolution. To establish a reference for acceptable divergence, the MITgcm benchmark was repeated with perturbations on the order of $\mathcal{O}(10^{-15})$ added to the initial conditions and forcing fields. This procedure follows Rosinski and Williamson (1997), who propose that successful code porting is achieved when the divergence between the original and translated solutions is not exceeding the growth of an initial perturbation introduced into the lowest-order bits of the original code solution. Both Veris and the MITgcm use double precision (float64) arithmetic. Even perturbations of this magnitude led to differences on the order of ~~3 and~~ $\mathcal{O}(10^{-2})$ in ice speed and $\mathcal{O}(10^{-1})$ in ice concentration. These deviations did not accumulate further over time, ~~indicating that the variations were consistent and within acceptable numerical limits~~displayed noise-like patterns, and were constrained primarily to regions with strong ice concentration gradients.

2.3 ~~Parallelization~~

~~The bottleneck in the current version of Veris is the EVP dynamics solver. In its current implementation, the EVP solver cannot run in parallel when using JAX as the backend.~~

~~The momentum equation is solved using the mEVP solver as follows. To advance from time step n to time step $n+1$, a number of subcycling iterations, indexed by p, are performed (Bouillon et al., 2013):~~

$$\sigma^{p+1} = \sigma^p + (\hat{\sigma}^p - \sigma^p)\frac{1}{\alpha}$$

$$\boldsymbol{u}^{p+1} = \boldsymbol{u}^p + \left(\hat{\boldsymbol{u}}^{p+1} - \boldsymbol{u}^p\right)\frac{1}{\beta}$$

~~α and β are numerical relaxation parameters. $\hat{\sigma}^p$ is computed directly from $\boldsymbol{u}^p$, while σ^p is derived using equation (??) , based on the previous subcycling step. The velocities are calculated after the stresses. $\hat{\boldsymbol{u}}^{p+1}$ is computed directly from σ^{p+1}, $\boldsymbol{u}^n$, and the forcing terms. The initial values are $\sigma^{p=0} = \sigma^n$ and $\boldsymbol{u}^{p=0} = \boldsymbol{u}^n$. After a sufficient number of subcycling iterations n_{EVP}, convergence $\sigma^{p+1} \to \sigma^p$ and $\boldsymbol{u}^{p+1} \to \boldsymbol{u}^p$ is assumed. The stress and velocity are then updated as $\sigma^{n+1} = \sigma^{p=n_{EVP}}$ and $\boldsymbol{u}^{n+1} = \boldsymbol{u}^{p=n_{EVP}}$. In the aEVP solver , the relaxation parameters α and β are determined based on the grid cell size and the ice viscosities, rather than being set to fixed values (Kimmritz et al., 2016). This approach allows for a lower number of subcycling iterations.~~

~~Since some calculations within the loop body of the EVP solver require the value of variables on neighboring grid cells, each partitioned region must include an overlap region that contains the values of the corresponding variables from adjacent partitioned regions . To update the values in this overlap,~~ The differences between Veris and the MITgcm are of the same or smaller magnitude than those obtained in the perturbed MITgcm reference simulations (Fig. A2), indicating successful code porting and variations within acceptable numerical limits. In addition to the 1000-time step benchmarks, a longer integration

of 3 months (12960 time steps) was performed to assess Veris's long-term behavior (Rosinski and Williamson, 1997), which remained consistent with the MITgcm.

2.3 Parallelization

In the initial implementation of Veris, developed as a sea ice plug-in for Veros, the EVP solver cannot be executed in parallel when using JAX as the backend. This limitation arises because the EVP solver requires halo exchanges between neighboring partitioning regions within the solver iteration. In the original implementation, these exchanges are performed inside the EVP loop body via a `fill_overlap` ~~function must be called at the end of each subcycling iteration. The basic sequence of this process is as follows~~routine:

```
for i in range(nEVPsteps):
    sigma = calculate_stress(...)
    uIce, vIce = calculate_velocity(...)

    uIce, vIce = fill_overlap(uIce, vIce)
```

When NumPy is used as the backend, the EVP solver is implemented with a ~~basic~~ standard Python for-loop, allowing Veris to ~~run in parallel. In Veris, functions are transformed with the~~ `@veros_kernel` ~~decorator, which is imported from Veros. This transformation incorporates both the Just-In-Time (JIT) compilation from JAX and the MPI parallelization. JIT compilation involves compiling functions at runtime, specifically during their first execution. However, when using JAX,~~ rely on the existing MPI-based communication infrastructure provided by Veros. When JAX is used, however, iterative loops are implemented using the ~~basic for-loop is computationally inefficient because the JIT compilation unrolls the loop~~ `jax.lax.fori_loop`, which compiles the loop body without unrolling every iteration during compilation. ~~Each iteration is treated as an explicit operation that is sent to the compiler. Given that the EVP solver typically requires several hundred subcycling iterations, this approach results in excessive compilation time. To address this issue, the~~ `JAX.lax.fori_loop` ~~is used. Unlike the basic for-loop, this loop is not unrolled during compilation, as the loop body is compiled only once. The~~ `JAX.lax.fori_loop` ~~is implemented as follows: But, unfortunately, the~~ `JAX.lax.fori_loop` ~~does not support parallel execution. While the loop body~~ This compilation model is not compatible with the communication mechanism employed in Veros. The communication mechanism of Veros enables MPI-based halo exchanges through a decorator that contains both the JIT compilation and the MPI communication. This decorator can only wrap functions that are not automatically compiled by JAX, and since the EVP loop is automatically compiled, ~~it is not compiled by the~~ `@veros_kernel` ~~decorator, and MPI parallelization is not included in this compilation. Consequently,~~ this decorator-based approach cannot be applied, and the `fill_overlap` ~~function cannot be used within the~~ routine cannot be invoked inside the compiled loop body. ~~For the EVP solver to run in parallel with JAX, a custom loop function would need to be developed that enables MPI parallelization~~ Enabling MPI-based parallelism in this context would therefore require a custom Veros-compatible mechanism that supports communication inside a compiled loop

while ensuring the loop body is traced and compiled only once. ~~Developing such a function~~ The development of such an interface is beyond the scope of this ~~manuscript~~work.

To enable parallel execution, we instead adopted JAX's native sharded array framework. Sharded arrays encode the distribution of array data across devices and provide point-to-point communication primitives that can be invoked within compiled functions. The model infrastructure of Veris was restructured around sharded arrays, while preserving the existing formulation of sea ice physics. This redesign resulted in a standalone, fully JAX-based version of Veris that supports scalable parallel execution on both CPU- and GPU-based architectures. All performance benchmarks presented in section 3 were conducted with this updated version of Veris, except for the Numpy-based simulations.

3 Results

3.1 Simulated Sea Ice Field

To demonstrate that Veris accurately simulates sea ice dynamics and its rheology, a simulation was conducted using a forcing that is designed to induce fracturing of the sea ice cover (Fig. 1). The forcing fields ~~(Fig. ??)~~ include a circular anti-cyclonic ocean surface current, which grows stronger towards the boundaries, and a circular cyclonic wind field with some horizontal convergence, displaced from the grid center and is strongest in its center. These fields are taken from the sea ice benchmark experiment described by Mehlmann et al. (2021), with the exception that the wind field is stationary in this work. The quadratic model domain is enclosed by land, resulting in boundary conditions of zero velocity. The simulation was run with a simulation time of 10 days with a time step of 10 minutes. The initial conditions included a constant ice thickness of $h = 0.3$ m and an ice concentration of $A = 1$. In the sea ice field, narrow features in the sea ice concentration are observed, resulting from inhomogeneities in the deformation field caused by the internal ice stress responding to the forcing fields. These linear kinematic features are discussed in Mehlmann et al. (2021) and show that Veris's dynamics are on par with state-of-the-art sea-ice dynamics codes.

~~Stationary ocean surface velocity and wind field.~~

Veris is also configured for integration with Veros to form a coupled sea ice-ocean model. Veros is run using setup files that define the configuration of the model, including specific extensions and the spatial discretization. Currently, Veris is integrated into the setup of a global simulation with a spatial resolution of 4°. This serves as a proof of concept, demonstrating that Veris can be successfully coupled with Veros to form a coupled sea ice-ocean model. This integration enables comprehensive simulations of sea ice dynamics within an oceanic framework. The results of a simulation using this setup are shown in Fig. 2, which displays ~~the~~ a snapshot of simulated Antarctic sea ice concentration on July 1, based on a model run that included zero ice thickness and concentration on January 1 as initial conditions. More realistic simulations require adequate configurations of Veros, for example with grids covering the Arctic Ocean. These configurations are not yet available and creating new setups for Veros is beyond the scope of this manuscript. In the following we concentrate on the computational performance of Veris.

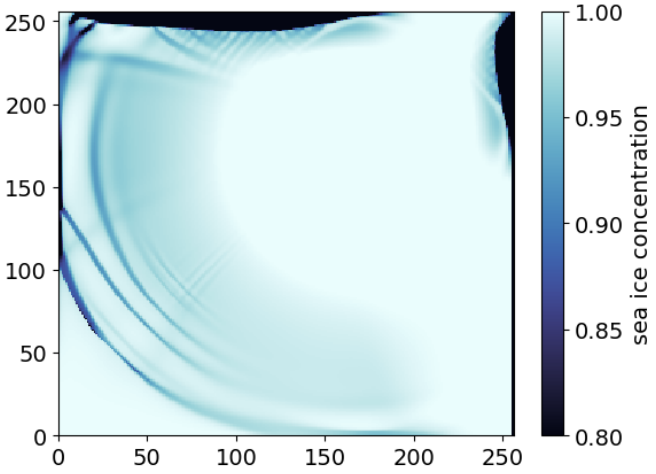

Figure 1. Simulated sea ice concentration generated by Veris after 1440 time steps of 10 minutes (equivalent to 10 days), driven by ~~the forcing~~ circular ocean surface currents and wind fields~~shown in Fig~~. **??.** ~~The~~ initial conditions included a uniform sea ice concentration of 1 and a sea ice thickness of 0.3 m. A reference simulation with the MITgcm, employing the same forcing, produced the same features (Fig. A2).

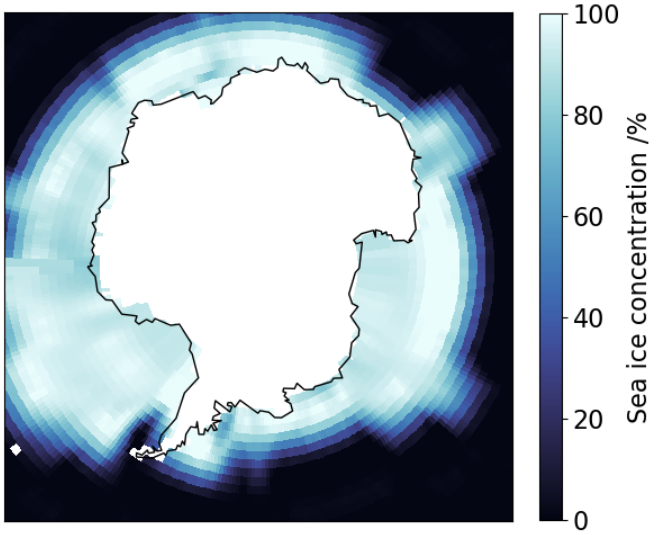

Figure 2. Simulated sea ice concentration field in the Antarctica generated by the Veris-Veros coupled model on July 1 in a one-year simulation using the ECMWF ERA5 reanalysis data from 1986 as forcing fields (Hersbach et al., 2020). The model was initialized with zero sea ice thickness and concentration, and the results were interpolated onto a grid with a spatial resolution of 1°.

3.2 Benchmark Results

To evaluate the performance of Veris, a series of idealized and scalable benchmark experiments were conducted~~using the benchmark suite available in the Veros repository~~. These experiments were performed on a rectangular grid with varying process count and problem size, utilizing the scaled forcing fields described in section 3.1. The benchmarks serve two primary purposes. First, they compare the performance of Veris to the Fortran reference, assessing the impact of transitioning from Fortran to Python. ~~For this, Veris running on a single CPU with Jax and Numpy as the backend is directly compared to the Fortran reference running on a single CPU~~Second, the benchmarks analyze the scaling behavior of Veris with increasing process count and domain size. Additionally, Veris running on a GPU is compared to the Fortran reference across various parallel CPU configurations, focusing on both time to solution and total power consumption. ~~Second, the benchmarks analyze the scaling behavior of Veris with increasing domain size.~~

During the first ~~iteration~~time step, the compilation process with JAX's JIT compiler introduces ~~overhead of approximately 40 seconds, which is independent of the problem size. Consequently~~significant overhead. To properly evaluate the model performance, the first ~~three iterations are discarded in the Veros benchmark suite to account for this initial overhead~~time step is therefore discarded.

The reference runs were performed with the MITgcm ~~,~~configured to include only the dynamical component, as the thermodynamical calculations are computationally less expensive and negligible in comparison to the dynamics. Veris was similarly run with just the dynamical component enabled. The models are compared using the aEVP solver with ~~400~~ 120 subcycling iterations.

The benchmarks are run on two different nodes of the HPC Levante, a supercomputer of the German Climate Computing Center (DKRZ). The specifications of the different node types are listed in Tab. 1.

	CPU node	GPU node
CPU	2x AMD 7763 CPU (128 cores in total)	2x AMD 7763 CPU (128 cores in total)
GPU	-	4x NVIDIA A100 (80 GB memory)
Memory	~~256~~ 512 GB	512 GB
Operating system	Red Hat Enterprise Linux 8.9	Red Hat Enterprise Linux 8.9
Software	OpenMPI 4.1.2, jaxlib 0.4~~.23~~ .30	CUDA ~~12.5~~13.0, jaxlib 0.4~~.23~~ .30

Table 1. Specifications of the benchmark platforms.

3.2.1 Scaling With ~~Domain Size~~Number of Processes

~~To compare the performance of Veris with the Fortran reference and to analyze its scaling behavior with domain size, benchmark~~ This section analyzes the performance scaling with respect to the number of processes (Fig. 3). For the CPU-based simulations, grids with approximately 10^6 and 9.4×10^6 grid cells were evaluated, hereafter referred to as the 1 M and 9.4 M grids. The 1 M grid experiments were conducted ~~using domain sizes ranging from 10^3 to 10^7 gridcells over 20 iterations. The mean~~

times per iteration are presented in Fig. 4. When using NumPy as the backend, Veris is approximately four times slower than on a single node. Both Veris and the MITgcm exhibit a similar initial increase in performance with rising core counts, a plateau in performance between 8–64 and 16–64 processes, respectively, and a renewed increase at 128 processes. For Veris, increasing the process count requires additional memory due to JAX's compilation cache, larger communication buffers, and the overhead associated with Python objects and metadata. For the 9.4 M grid, this necessitates distributing the simulation across multiple nodes, with at most 16 processes per node. The MITgcm is able to run the 9.4 M grid on a single node; however, this configuration results in a substantial performance degradation and poorer scaling behavior. When the Fortran reference for a grid size of 10^6 grid cells. This performance difference is consistent with that observed between Veros and its Fortran reference (Häfner et al., 2021). With JAX as the backend, Veris performs significantly better, being only 1.7 times slower than the Fortran reference for a grid size of 10^6 cells. Notably, Veris with Jax demonstrates the same scaling behavior as simulation is distributed across multiple nodes, the MITgcm outperforms Veris, achieving approximately a fourfold speedup over Veris at 128 processes (Fig. 3). When constrained to a single node, the MITgcm exhibits both worse scaling and lower absolute performance than Veris running across multiple nodes.

The GPU-based simulations substantially outperform the CPU-based simulations, with a single GPU process matching the performance of the MITgcm running on hundreds of CPU cores. Increasing the number of processes introduces additional communication overhead, which is more pronounced for GPUs than for CPUs because GPUs do not share memory and therefore require explicit data movement between devices for communication. As a result, performance scaling on GPUs is strongly dependent on problem size. For the 1 M grid, increasing the GPU count from 1 to 128 yields only marginal performance gains. The 9.4 M grid exhibits improved scaling but still approaches saturation. Among the tested configurations, the 26 M grid, which contains approximately 26×10^6 grid cells, shows the best scaling behavior, with a consistent performance increase as the GPU count increases. However, even for this grid, only limited performance gains are observed for higher GPU counts, indicating saturation. Overall, these results demonstrate the substantial performance benefits of GPU-based execution compared to CPU-based systems. They further suggest that GPU parallelization becomes increasingly effective for larger problem sizes, highlighting the potential of Veris for very large-scale simulations on GPU-accelerated architectures.

3.2.2 Scaling With Domain Size

In addition to the scaling with respect to process count, we performed benchmarks to analyze the dependence of performance on domain size for single-process simulations (Fig. 4). Veris exhibits scaling behavior consistent with that of the Fortran reference . For optimal GPU performance, it is essential to fully utilize the GPU's memory. Due to implementation. Scaling in this section refers to the significant overhead at smaller grid sizes , the mean iteration time shows no significant change for grid sizes below 10^5 grid cells. However, for larger grid sizes, Veris using JAX on a GPU exhibits the same scaling behaviour as the Fortran reference, demonstrating efficient performance at scaleincrease in time to solution with domain size. When using Numpy as the backend, Veris is approximately five times slower than the MITgcm for domain sizes 10^6 or smaller and around 15 times slower on domain sizes 10^7 or larger. With the JAX backend, Veris consistently outperforms the Fortran reference, with mean iteration times approximately a factor of two smaller for grids larger than 10^6 cells.

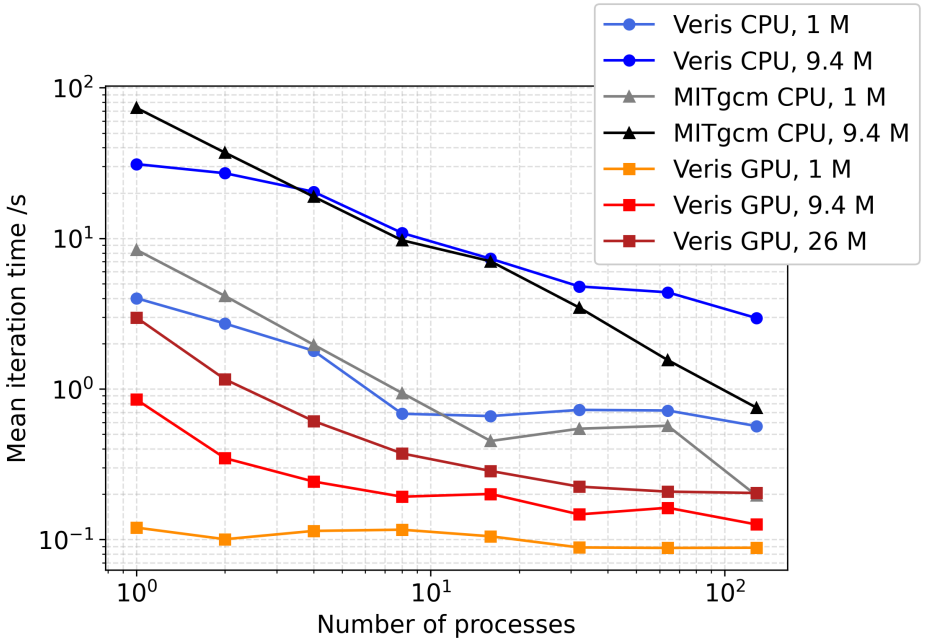

Figure 3. Scaling of mean iteration time with process count for Veris and the MITgcm sea ice component for varying problem sizes (1 M = 10^6 grid cells), calculated over 20 iterations steps.

Simulations executed on a single GPU demonstrate both a substantial absolute performance increase and good scaling with increasing grid size. These results highlight the effectiveness of GPU-based execution for large domains and further emphasize the suitability of the JAX-based implementation for large-scale, high-resolution sea ice simulations.

3.2.3 Power Consumption

~~Next to the~~ In addition to time to solution, ~~another crucial~~ energy consumption is a key metric in high-performance computing~~is the energy consumption of running a simulation. Minimizing energy consumption not only reduces~~ . Reducing energy usage not only lowers operational costs but also ~~lowers~~ mitigates the carbon footprint of ~~computational workloads. A promising approach to achieving significant energy savings is replacing multiple CPUs with a single GPU~~large-scale simulations. One promising approach for improving energy efficiency is to replace large numbers of CPU cores with a smaller number of GPUs, which can ~~drastically reduce the total energy usage of a simulation.~~ provide substantially higher performance per watt.

For the power consumption benchmark, we selected the 9.4 M grid configuration and measured the energy usage over 1000 iterations. The reference simulation was performed using a single GPU. For ~~the Fortran reference, achieving linear scaling of the time to solution with the number of CPUs requires only using one CPU per node. This is likely due to the code being bandwidth limited. Each node has a shared memory subsystem with a finite memory bandwidth. When multiple CPUs on a node are active, they share this memory bandwidth, which can lead to them competing for memory access. For~~

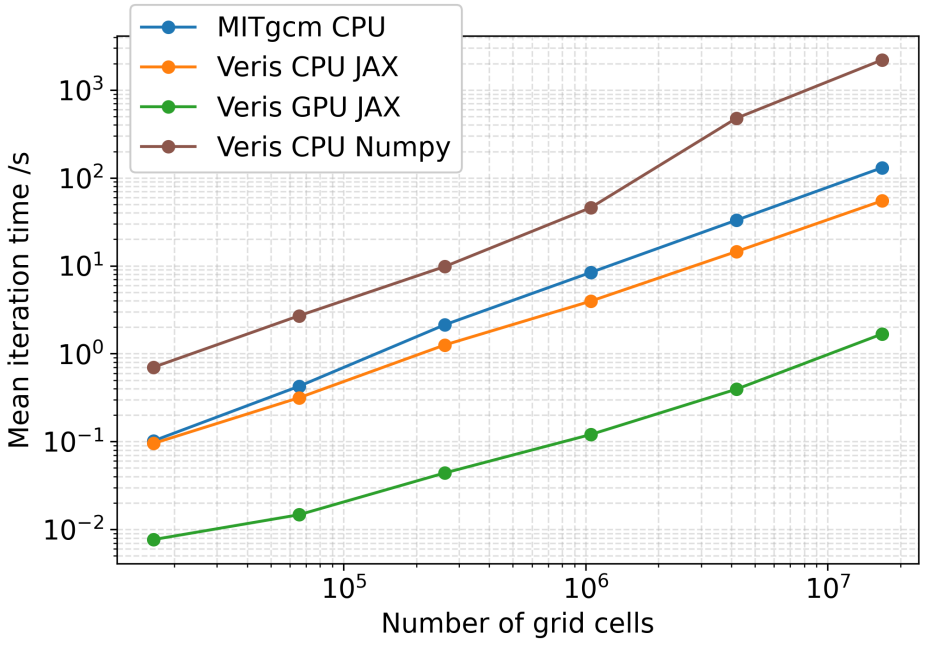

Figure 4. Benchmark results comparing the scaling of mean iteration time with grid size for Veris and the MITgcm, calculated over 20 iteration steps.

~~memory-intensive workloads, such as simulations that require frequent memory access, this competition becomes a bottleneck. By restricting execution to a single CPU per node, the entire memory bandwidth is dedicated to that CPU, allowing it to perform at maximum efficiency. In this configuration, where only one CPU is used per node, the Fortran reference achieves performance comparable to Veris running on a GPU when 45 CPUs are utilizes across 45 nodes. However, while this configurationmatches the GPU's performance, it significantly increases power usage due to the large number of active nodes . Using 45 CPUs~~ the MITgcm, CPU-based simulations were conducted that matched the time to solution of the single-GPU configuration, using both the minimum number of nodes and the minimum number of CPU cores (Tab. 2). In addition, a simulation utilizing all 4 GPUs available on a single ~~node significantly reduces power consumption, but the~~ GPU node was performed. The time to solution ~~increases substantially due to the CPUs competing for memory bandwidth. The optimal configuration for the Fortran reference, in terms of matching the GPU performance with a minimum number of nodes , is 224 CPUs distributed across 2 nodes. In this setup, the~~ of the single-GPU run was matched by a configuration using 640 CPU cores across 5 nodes and by a configuration using 128 CPU cores across 8 nodes. Both CPU-based simulations exhibited similar total energy consumption~~is 50 % higher than that of Veris on a GPU. A detailed comparison of the time to solution, mean power usage, and total energy consumption for the various configurations is presented in Tab. 2. Veris~~ , requiring approximately 4–4.5 times the energy of the single-GPU simulation. When all four GPUs on a GPU ~~performs a simulation with 10^6 grid cells over 1000 iterations with a total energy usage of 0.08 kWh. In comparison, the Fortran reference running on 224 CPUs distributed across 2 nodes achieves~~

~~670 iterations with the same~~ node were utilized, the total energy consumption ~~.~~ was reduced by a factor of two relative to the single-GPU simulation. These results demonstrate the substantial energy-efficiency advantages of GPU-based execution for large-scale sea ice simulations.

	Time to solution	Mean power usage	Total energy usage
~~45 CPUs on 45 nodes~~ 1 GPU	~~449~~ 854 s	~~45 · 254 W = 11430 W~~ 650 W	~~1.43~~ 0.15 kWh
~~45 CPUs on 1 node~~ 4 GPUs	~~2008~~ 251 s	~~370~~ 1105 W	~~0.21~~ 0.077 kWh
~~224 CPUs on 2~~ 640 CPUs on 5 nodes	~~460~~ 846 s	~~470 W + 480 W = 950 W~~ 2845 W	~~0.12~~ 0.67 kWh
~~1 GPU~~ 128 CPUs on 8 nodes	~~459~~ 742 s	~~630~~ 2960 W	~~0.08~~ 0.61 kWh

Table 2. Power consumption for Veris running on 1 and 4 GPUs (on the same GPU node) and the MITgcm sea ice component running ~~on~~ in various ~~CPU~~ CPU-based configurations~~and Veris on~~ . CPU here refers to a single ~~GPU~~CPU-core process. ~~The~~ This simulation is using ~~10^6~~ 9.4×10^6 grid cells and 1000 iterations. The power usage is reported by Levante's built-in power tracking system.

4 ~~Summary and Conclusion~~Discussion

~~By translating the sea ice component of the MITgcm into Python, a~~ The performance benchmarks of both the MITgcm and Veris showed a strong dependence of model performance on the underlying HPC configuration. Identical numbers of CPU-cores can yield substantially different solution times, depending on how processes are distributed across the nodes and mapped to cores within a node. This shows that optimal performance requires not only efficient model implementations, but also careful tuning of the parallel configuration to the HPC system. Such tuning involves the choice of processor grids, node layouts, and process placement strategies, and presupposes a detailed understanding of the HPC system.

For established Fortran-based model such as the MITgcm, MPI-based communication has been extensively optimized over years of development and use. In contrast, distributed execution in JAX is an active area of development, and there is considerable potential for further improvements in communication backends and process orchestration. The results of this study therefore highlight both the promise of JAX for climate modeling in HPC environments and the importance of systematic communication analysis and optimization. The strong impact of communication overhead is also reflected in the early saturation observed in GPU-based simlations for smaller problem sizes. Given that Veris consistently outperforms the MITgcm at small process counts, it is plausible that further advances in JAX communication could enable Veris to outperform its Fortran reference in strongly parallel regimes as well.

The benchmarks further reveal a significant performance benefit when transitioning from CPU-only systems to GPU-accelerated architectures, underscoring the growing importance of GPUs for climate modeling. This is particularly advantageous in the context of JAX, which is accelerator-agnostic and allows newly developed model components to run on GPUs without additional code modifications.

The GPU-based simulations exhibit the expected scaling behavior, with improved efficiency for larger domains, consistent with the high communication overhead associated with multi-GPU execution. This emphasizes the suitability of multi-GPU

configurations for high-resolution modeling. Moreover, the power consumption measurements demonstrate that GPU-based Veris simulations can achieve comparable or superior performance at a fraction of the energy cost of CPU-based MITgcm runs, showing the potential of GPU-accelerated Python models to reduce both the operational cost and the carbon footprint of climate simulations.

The sensitivity of model performance to the chosen HPC configuration is less pronounced for GPU-based simulations than for CPU-based simulations. This is partly attributable to the fact that JAX is currently better optimized for GPU execution than for distributed CPU-based simulations. These results suggest that JAX holds further promise for CPU-based climate modeling, given the ongoing development of JAX and its distributed runtime, which may lead to further performance improvements in the future.

5 Summary and Conclusion

By translating the sea ice component of the MITgcm into Python, we developed Veris, a fully Python-based ~~sea ice model , Veris, was developed. Veris can be coupled to the~~ sea ice model. Veris can be coupled to the Python-based ~~ocean model Veros to form a coupled~~ ocean model Veros, forming a coupled ocean-sea ~~ice model entirely implemented in Python. With JAX as the backend, Veris achieves significant performance improvements, running 2.4 times faster than with NumPy. For a grid size of 10^6 grid cells, Veris is only 1.7 times slower than the Fortran reference. Furthermore, when executed on a high-end GPU using JAX's GPU support, Veris matches the performance of the Fortran reference running on 45 CPUs distributed across 45 nodes or 224 CPUs distributed across 2 nodes. Veris using JAX demonstrates the same scaling behaviorof mean iteration time with grid size as the Fortran reference, validating its suitability for~~ ice model implemented entirely in Python. This version supports parallel execution with Numpy and single-process execution with JAX on both CPU and GPU architectures. Although this version does not support distributed parallelism for JAX-based simulations, the benchmarks already demonstrated the potential JAX has for large-scale ~~simulations. The results show that Veris is a robust and efficient Python-based sea ice model capable of being used in HPC simulations~~simulations, given the substantial performance gains of GPU execution. ~~Moreover, by replacing multiple CPUs with a single GPU ,~~

To enable scalable parallelism, the ~~energy consumption of running simulations can be drastically reduced. The Fortran reference, in its most energy-efficient configuration (224 CPUs on 2 nodes), consumes 50 % more total energy than Verisrunning on a single GPU.~~ Veris infrastructure was subsequently redesigned around JAX's sharded array framework, allowing distributed execution on both CPU- and GPU-based systems. The benchmarks presented in this study demonstrate the great potential of JAX for climate modeling. At small process counts, Veris outperforms the MITgcm. At larger process counts, communication overhead becomes the dominant performance factor, and the MITgcm runs faster than Veris. This sensitivity on the communication framework was also evident in the dependence of performance on specific HPC configurations. Whereas MPI-based communication in the MITgcm benefits from long-standing optimizations, communication in JAX-based distributed systems is still evolving. The superior performance of Veris at low process counts indicates strong potential for large-scale applications, particularly as JAX's distributed runtime continues to be developed and optimized.

~~Despite these promising results, Veris currently cannot run in parallel across multiple CPUs when using JAX as the backend. Achieving this requires a loop function that can compile its loop body without unrolling it, while also supporting distributed execution via MPI. Once implemented, Veris will fully utilize the computational power of HPCs, including support for simulations on multiple GPUs using JAX . Currently, Veris on a single GPU matches the performance of the Fortran reference on 45 CPUs on 45 nodes or 224 CPUs on 2 nodes, but with multiple GPUs, it is expected to match~~ GPU execution yields particularly strong performance benfits. A single-GPU run of Veris matches the performance of the ~~Fortran reference running on significantly larger CPU counts.~~ MITgcm running on hundreds of CPU cores across multiple nodes, while consuming substantially less energy. These results demonstrate not only the competitiveness of Python-based, JAX-accelerated models in terms of time to solution, but also their potential to significantly improve the energy efficiency of climate simulations.

In summary, Veris demonstrates that Python, ~~supported by~~ when combined with JAX, can ~~deliver competitive performance for large-scale sea ice simulations. Its energy efficiency and scalability make it a promising tool for modern HPC applications~~support high-performance, scalable, and energy-efficient sea ice modeling. These findings suggest that JAX-based climate models are a promising pathway towards modern HPC workflows, with the ~~potential for even greater performance gains through future parallelization and multi-GPU support~~prospect of further performance gains as communication backends and distributed execution frameworks continue to improve.

Code availability. ~~Veris is available under https://github.com/team-ocean/veris. All scripts used to perform the presented benchmarks and simulations, raw benchmark results and plotting scripts are available under https://doi.org/10.5281/zenodo.14604718.~~The initial version of Veris is available under https://github.com/team-ocean/veris. It can be used as a plug-in for Veros, available under https://github.com/team-ocean/veros, or run as a standalone model using the scripts available under https://github.com/jpgaertner/veris_minimum_working_example. The updated version of Veris for parallel execution using JAX is available under https://github.com/jpgaertner/veris/tree/jax-only. The scripts for running Veris in this configuration and for creating the benchmarks presented in this study are available under https://github.com/jpgaertner/veris_minimum_workin The MITgcm is available under https://github.com/MITgcm/MITgcm.git. The configurations used to create the thermodynamic and dynamic benchmarks are available under https://github.com/mjlosch/MITgcm/tree/seaice_test_1d and https://github.com/mjlosch/seaice_benchmark.git. The benchmark results presented in this manuscript and the plotting scripts are available under https://doi.org/10.5281/zenodo.18620150..

Appendix A: Code Verification, Comparison of Simulated Sea Ice between Veris and the MITgcm

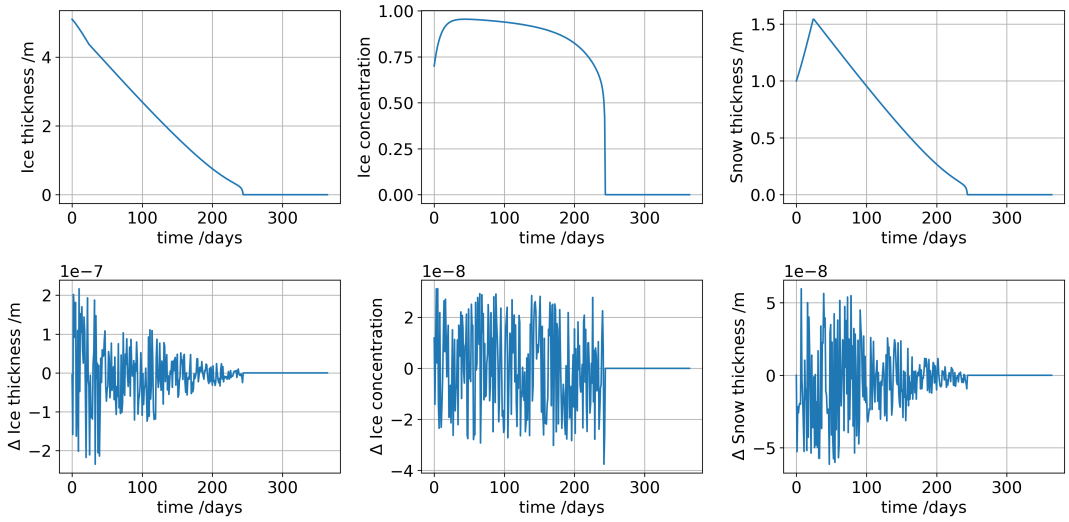

Figure A1. Example benchmark simulation used for verifying the thermodynamic component of Veris. Top row: Thermodynamic evolution of ice thickness, ice concentration, and snow thickness simulated by Veris. Bottom row: Comparison of the simulated sea ice between Veris and the MITgcm employing the same initial and forcing conditions. This figure is representative of the results of different thermodynamic benchmark comparisons using different initial and forcing conditions.

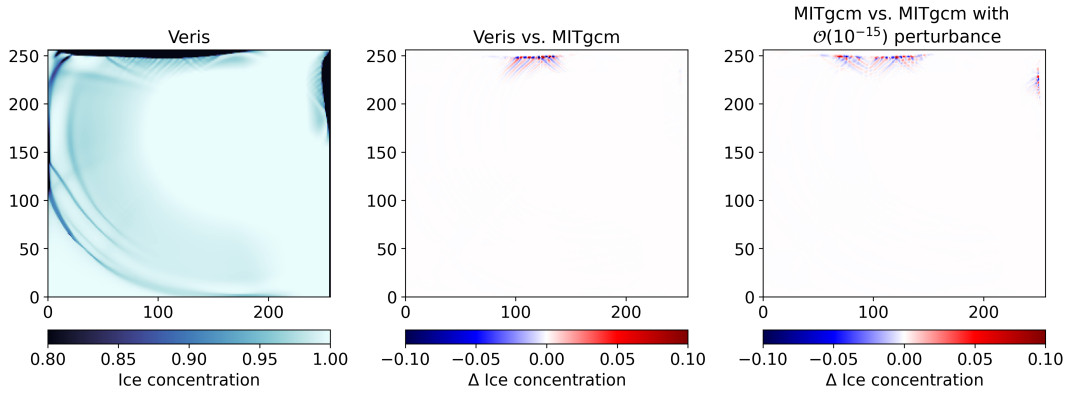

Figure A2. Example benchmark simulation used for verifying the dynamic component of Veris. Left panel: Sea ice concentration simulated by Veris after 1000 10-minute time steps using forcing fields designed to induce ice fracturing. Middle panel: Comparison of the simulated sea ice between Veris and the MITgcm employing the same initial and forcing conditions. Right panel: As a reference for acceptable divergence, the difference between the MITgcm and a rerun of the MITgcm with perturbed initial and forcing conditions is shown. This figure is representative of the results of different dynamic benchmark comparisons using different grid sizes and initial and forcing conditions.

Author contributions. JPG wrote the python code with ML and SKC, carried out the benchmark experiments with ML, SKCand RN, coupled the model with Veros with RN and MJ, and ran the global simulations with ML. JPG wrote the manuscript with help of ML, MJ, RN.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. This paper builds upon the Master's thesis of Gärtner (2023), during which Veris was developed.

We express our gratitude to Dion Häfner for his assistance in coupling Veris with Veros.

References

Arakawa, A. and Lamb, V. R.: Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model, in: General Circulation Models of the Atmosphere, vol. 17 of *Methods in Computational Physics: Advances in Research and Applications*, pp. 173–265, Elsevier, https://doi.org/10.1016/B978-0-12-460817-7.50009-4, 1977.

Bouillon, S., Fichefet, T., Legat, V., and Madec, G.: The Elastic–Viscous–Plastic Method Revisited, Ocean Modelling, 71, 2–12, https://doi.org/10.1016/j.ocemod.2013.05.013, 2013.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q.: JAX: composable transformations of Python+NumPy programs, GitHub, http://github.com/jax-ml/jax, 2018.

Eden, C.: Closing the energy cycle in an ocean model, Ocean Modelling, 101, 30–42, https://doi.org/10.1016/j.ocemod.2016.02.005, 2016.

Eden, C. and Olbers, D.: An energy compartment model for propagation, nonlinear interaction, and dissipation of internal gravity waves, Journal of Physical Oceanography, 44, 2093–2106, https://doi.org/10.1175/JPO-D-13-0224.1, 2014.

Gärtner, J.: Sea Ice Modeling in Python, Zenodo, https://doi.org/10.5281/ZENODO.11474409, https://zenodo.org/doi/10.5281/zenodo.11474409, 2023.

Häfner, D., Jacobsen, R. L., Eden, C., Kristensen, M. R., Jochum, M., Nuterman, R., and Vinter, B.: Veros v0.1–A Fast and Versatile Ocean Simulator in Pure Python, Geoscientific Model Development, 11, 3299–3312, https://doi.org/10.5194/gmd-11-3299-2018, 2018.

Häfner, D., Nuterman, R., and Jochum, M.: Fast, Cheap, and Turbulent—Global Ocean Modeling with GPU Acceleration in Python, Journal of Advances in Modeling Earth Systems, 13, e2021MS002 717, https://doi.org/10.1029/2021MS002717, 2021.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array Programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., et al.: The ERA5 Global Reanalysis, Quarterly Journal of the Royal Meteorological Society, 146, 1999–2049, https://doi.org/10.1002/qj.3803, 2020.

Hibler, W. D.: A Dynamic Thermodynamic Sea Ice Model, Journal of physical oceanography, 9, 815–846, https://doi.org/10.1175/1520-0485(1979)009%3C0815:ADTSIM%3E2.0.CO;2, 1979.

Hunke, E., Allard, R., Bailey, D. A., Blain, P., Craig, A., Dupont, F., DuVivier, A., Grumbine, R., Hebert, D., Holland, M., Jeffery, N., Lemieux, J.-F., Osinski, R., Poulsen, J., Steketee, A., Rasmussen, T., Ribergaard, M., Roach, L., Roberts, A., Turner, M., Winton, M., and Worthen, D.: CICE-Consortium/CICE: CICE Version 6.5.1, Zenodo, https://doi.org/10.5281/zenodo.11223920, 2024.

Hunke, E. C. and Dukowicz, J. K.: An Elastic–Viscous–Plastic Model for Sea Ice Dynamics, Journal of physical oceanography, 27, 1849–1867, https://doi.org/10.1175/1520-0485(1997)027%3C1849:AEVPMF%3E2.0.CO;2, 1997.

Kimmritz, M., Danilov, S., and Losch, M.: On the convergence of the modified elastic–viscous–plastic method for solving the sea ice momentum equation, Journal of Computational Physics, 296, 90–100, 2015.

Kimmritz, M., Danilov, S., and Losch, M.: The Adaptive EVP Method for Solving the Sea Ice Momentum Equation, Ocean Modelling, 101, 59–67, https://doi.org/10.1016/j.ocemod.2016.03.004, 2016.

Lemieux, J.-F., Knoll, D. A., Tremblay, B., Holland, D. M., and Losch, M.: A comparison of the Jacobian-free Newton–Krylov method and the EVP model for solving the sea ice momentum equation with a viscous-plastic formulation: A serial algorithm study, Journal of Computational Physics, 231, 5926–5944, 2012.

Lemieux, J.-F., Tremblay, L. B., Dupont, F., Plante, M., Smith, G. C., and Dumont, D.: A Basal Stress Parameterization for Modeling Landfast Ice, Journal of Geophysical Research: Oceans, 120, 3157–3173, https://doi.org/10.1002/2014JC010678, 2015.

Leppäranta, M.: A growth model for black ice, snow ice and snow thickness in subarctic basins, Hydrology Research, 14, 59–70, https://doi.org/10.2166/nh.1983.0006, 1983.

Li, B., Basu Roy, R., Wang, D., Samsi, S., Gadepally, V., and Tiwari, D.: Toward Sustainable HPC: Carbon Footprint Estimation and Environmental Implications of HPC Systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15, https://doi.org/10.1145/3581784.3607035, 2023.

Liu, Y., Losch, M., Hutter, N., and Mu, L.: A New Parameterization of Coastal Drag to Simulate Landfast Ice in Deep Marginal Seas in the Arctic, Journal of Geophysical Research: Oceans, 127, e2022JC018 413, https://doi.org/10.1029/2022JC018413, 2022.

Losch, M., Menemenlis, D., Campin, J.-M., Heimbach, P., and Hill, C.: On the Formulation of Sea-Ice Models. Part 1: Effects of Different Solver Implementations and Prameterizations, Ocean Modelling, 33, 129–144, https://doi.org/10.1016/j.ocemod.2009.12.008, 2010.

Marquetti, I., Rodrigues, J., and Desai, S. S.: Ecological Impact of Green Computing using Graphical Processing Units in Molecular Dynamics Simulations, International Journal of Green Computing (IJGC), 9, 35–48, https://doi.org/10.4018/IJGC.2018010103, 2018.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., and Heisey, C.: A Finite-Volume, Incompressible Navier Stokes Model for Studies of the Ocean on Parallel Computers, Journal of Geophysical Research: Oceans, 102, 5753–5766, https://doi.org/10.1029/96JC02775, 1997.

Mehlmann, C., Danilov, S., Losch, M., Lemieux, J.-F., Hutter, N., Richter, T., Blain, P., Hunke, E., and Korn, P.: Simulating Linear Kinematic Features in Viscous-Plastic Sea Ice Models on Quadrilateral and Triangular Grids with Different Variable Staggering, Journal of Advances in Modeling Earth Systems, 13, e2021MS002 523, https://doi.org/10.1029/2021MS002523, 2021.

Méndez, M., Tinetti, F. G., and Overbey, J. L.: Climate models: challenges for Fortran development tools, in: 2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering, pp. 6–12, IEEE, https://doi.org/10.1109/SE-HPCCSE.2014.7, 2014.

Portegies Zwart, S.: The Ecological Impact of High-Performance Computing in Astrophysics, Nature Astronomy, 4, 819–822, https://doi.org/10.1038/s41550-020-1208-y, 2020.

Rasmussen, T. A. S., Poulsen, J., Ribergaard, M. H., Sasanka, R., Craig, A. P., Hunke, E. C., and Rethmeier, S.: Refactoring the elastic–viscous–plastic solver from the sea ice model CICE v6. 5.1 for improved performance, Geoscientific Model Development, 17, 6529–6544, https://doi.org/10.5194/gmd-17-6529-2024, 2024.

Rosinski, J. M. and Williamson, D. L.: The accumulation of rounding errors and port validation for global atmospheric models, SIAM Journal on Scientific Computing, 18, 552–564, 1997.

Semtner, A. J.: A Model for the Thermodynamic Growth of Sea Ice in Numerical Investigations of Climate, Journal of Physical Oceanography, 6, 379–389, https://doi.org/10.1175/1520-0485(1976)006%3C0379:AMFTTG%3E2.0.CO;2, 1976.