Dear reviewer,

Thank you very much for your interest in our paper and your feedback. In the following, we will directly respond to your comments.

Best regards,

Jan Gärtner

Reviewer Comment 1
Within the model description I still find that there is a discrepancy between the broad model description and the description of the EVP solver that is mostly in focus here. This weight should somehow be reflected in the model description and more focus should be on the EVP . Maybe because the manuscript try to be both a general model introduction and at the same time focus on a specific element.

Author Response
We have added a more comprehensive description of the EVP solver. The basic properties of the model are listed in section 2.1, while the EVP solver is explained in more detail in the newly added section 2.2.

Reviewer Comment 2
The study generalizes a comparisons of CPU's and GPU's based on 1 example of each. In reality this is a bit more complex as one for instance can buy both low end and high-end hardware. Therefore, I think that the conclusions of CPU vs GPU are generalized more than the result allow. It is good that the power consumption is brought into the discussion. It should be mentioned that price will also be a part of the decision when buying hardware. A comment about the price in the discussion is sufficient as these will vary.

Author Response
We have added the following paragraph in the discussion:

The choice of hardware, including the specific GPU and CPU, affects both model performance and energy consumption Rasmussen et al. (2024). In addition to performance, cost is an important factor when evaluating the usefulness of GPU- and CPU-based architectures. Here, we used hardware available at the DKRZ, a representative and widely used HPC system for climate research in Germany. The NVIDIA A100 80GB GPU installed at the DKRZ typically costs $20,000-$30,000, while the AMD 7763 CPU retails for approximately $3,000-$4,000.

Reviewer Comment 3
For the ocean model Veros I find that the discussion of its ability to couple is good but not particular important for this manuscript, thus it should not be mentioned in the abstract as it is not important for the conclusion of this manuscript. I would move this into the discussion with a reference to future development.

Author Response
We have removed Veros from the abstract and added the following paragraph to the discussion:

The initial version of Veris was configured for integration with Veros to form a coupled sea ice-ocean model. This version and its coupling to Veros are described in detail in Gärtner (2023). To enable parallel execution with JAX, however, Veris was restructured around JAX's sharded arrays and is no longer compatible with Veros in this updated version. The coupled Veros-Veris model can be executed using NumPy on both single and multiple processes, and with JAX on a single process. Future work on enabling parallel execution of the coupled model with JAX will require either the development of a

Veros-compatible communication mechanism within compiled loops (see section 2.4) or a redesign of Veros' communication infrastructure around JAX's sharded arrays.

Reviewer Comment 4
For the result/discussion I would like to have a larger focus on how the time to solution/scalability should be improved and if this is possible on the provided hardware. There is short reference to saturation (bandwidth I guess), which is likely the cause, however I miss an analysis of this. Is it feasible to have an effective EVP solver with the high number of MPI calls? Should OMP parallelization be used?

Author Response
Scalability could be improved if communication is further optimized. This is discussed in the second paragraph of the discussion.

Regarding the saturation, we have added this explanation:

In this context, saturation refers to the limited increase in performance with process count, which is attributed to the large communication overhead exceeding computational efficiency.

Whether an effectively parallelized EVP solver is possible is subject of future work. As highlighted in the discussion, effective parallelization relies on effective communication, which is an ongoing area of research.

JAX's sharded arrays are distributed across multiple devices. Communication between devices is managed automatically by the XLA (Accelerated Linear Algebra) compiler, which inserts collective operations tailored to the underlying hardware. OpenMP is a shared-memory threading model, which is not appropriate for JAX because JAX's parallelism operates at the device level, not at the thread level.

Reviewer Comment 5
You have already cited Rasmussen et al., there is a discussion of the bottlenecks in a traditional FORTRAN EVP solver. Is it the same bottlenecks in the Python code? It would be nice if the authors could show what the bottlenecks are.

Author Response
We have added the following sentence to the discussion:

The strong dependence of model performance on HPC configuration indicates that the model is highly bandwidth-bound, a limitation that has also been identified as a major bottleneck in previous studies of Fortran implementations of the EVP solver (Rasmussen et al., 2024).

For established Fortran-based models, MPI-based communication has been extensively optimized over years of development and use. However, it is still identified as another major bottleneck of performance (Rasmussen et al., 2024).

Reviewer Comment 6
Line 25: Especially for the case here focused on the EVP solver is it really the case that the number of floating-point calculations is important for the run time? Hardware has changed with more and more CPU's per node/GPU's and often it is the bandwidth that is the issue.

Author Response
We have revised the manuscript as follows:

Another key reason for Fortran's prevalence is its high computational efficiency in compute-intensive climate models (Méndez et al., 2014). While numerical computation speed was historically the primary bottleneck, modern hardware has shifted this limitation to memory bandwidth. Fortran allows for low-level optimizations, such as SIMD (Single Instruction Multiple Data) vectorization and direct memory addressing, critical for performance in bandwidth-bound climate simulations (Rasmussen et al., 2024).

Reviewer Comment 7
Line 45: An important "if" is mentioned here. Many climate models are not built to utilize the hardware of today.

Author Response
Thanks for pointing this out. We have added the following sentence to the manuscript to emphasize this issue:

Unfortunately, many climate models cannot fully utilize state-of-the-art hardware, motivating the development of models with efficient GPU support.

Reviewer Comment 8
Line 59-62: What is needed here. Do you need to change Veros? This means that the coupled system cannot be run in parallel?

Author Response
This has been addressed by the paragraph we have added under reviewer comment 3.

Reviewer Comment 9
Line 140: A2 shows that the bias is localized. Any idea why (besides the lower concentration/higher sea ice gradient)?

Author Response
We have modified the manuscript as follows:

These deviations with patterns nearly on the grid scale did not accumulate further over time and were constrained primarily to regions with pronounced grid-scale variability. This accumulation is attributed to small horizontal displacements in sea ice concentration between the model simulations that have a larger impact in these regions than in regions with smoother sea ice.

Reviewer Comment 10
Sharded arrays are used as the solution but communication is still needed (If I understand this part correct). It would be beneficial to describe this a bit more as it is central to why it works, how efficient is it?

Author Response We have added the following explanation to the manuscript:

To enable parallel execution, we adopted JAX's native sharded-array framework. Sharded arrays explicitly contain information on how their data are distributed across the available devices. Communication and synchronization between devices are managed automatically by the XLA (Accelerated Linear Algebra) compiler, which inserts collective operations and resharding operations tailored to the underlying hardware. While communication remains necessary, it is managed by the compiler and at runtime rather than being expressed manually in the sea ice model code. This separation of scientific implementation from distributed execution reduces programming complexity and improves portability across diverse hardware architectures. Compared with our earlier `mpi4jax`*-based approach, JAX's sharded arrays also reduce maintenance effort by relying on stable, backward-compatible public APIs, whereas* `mpi4jax` *depends on internal JAX APIs that may change between versions and require adaptation. On GPU systems, JAX's sharding backend can leverage optimized communication libraries such as NCCL (NVIDIA Collective Communications Library) for distributed transfers and collective operations. The Veris infrastructure was therefore restructured around sharded arrays while preserving the existing sea ice physics formulation, resulting in a standalone, fully JAX-based version of Veris that supports portable and scalable parallel execution on both CPU- and GPU-based architectures.*

Reviewer Comment 11
Line 190 to 195. How is the ocean initialized? Is it realistic? The run starts on the 1/1 and the plot shows the 1/7. Is that after 6 months? The text states that a one year solution is carried out? I am not entirely sure what figure 2 demonstrates except that the system runs and that it produces sea ice.

Author Response
You are right with pointing out that this figure does not add more to the manuscript than showing that Veris can be coupled to Veros. We have therefore decided to omit this figure as the coupling with Veros is already discussed.

Reviewer Comment 12
Line 202: The just in time compilation is a reason for the performance improvement. I think that it is important to state how significant this overhead is. Is it only timings of the dynamics that are used or is it the full model that runs only dynamics (including I/O)?

Author Response
We have added the following explanations:

The benchmarks are performed over 20 time steps. During the first time step, the compilation process with JAX's JIT compiler introduces significant overhead. To properly evaluate the model performance, the first time step is therefore discarded. The absolute compilation time for CPU-based processes ranges from 2.0 to 32.2 seconds and the relative compilation time ranges from 1.0 to 10.4 model iterations. For GPU-based processes, absolute and relative compilation times range from 2.1 to 32.4 seconds and from 2.1 to 6.2 model iterations, respectively.

For both Veris and the MITgcm, the full model is run with thermodynamics disabled at runtime (i.e., it is compiled but not run), and the reported timings exclude thermodynamics, initialization, and finalization. This approach is justified because thermodynamic calculations are computationally less expensive and negligible compared to the dynamics.

Reviewer Comment 13
Section 3.2.2: It is not entirely clear for me how one GPU is specified. How is this comparable to one CPU?

Author Response
What is compared is not directly a certain number of GPUs vs. CPUs, rather the number of processes for GPU- vs. CPU-based execution. This specification is important because firstly the GPU-based executions rely on CPU-cores for initialization, and secondly a single CPU can include many cores. This is indicated in the manuscript in "This section analyzes the performance scaling with respect to the number of processes." and the explanation of processes we have added under reviewer comment 14.

Reviewer Comment 14
How do you specify different number of GPU processes? A GPU processor normally comes with "many" cores/threads that share the workload. Are you sure it is the communication? GPU's likes to have a high throughput of data. The larger grid likely facilitates this.

Author Response
We have added the following explanation to the manuscript:

The number of processes corresponds to the number of partitioning regions and is specified by the scheduler of the HPC system used for benchmarking. A CPU-based process is bound to a single CPU-core, while a GPU-based process is bound to a single GPU, with JAX kernels automatically exploiting all available threads in the GPU.

Reviewer Comment 15
Section 3.2.3: The CPU nodes do not utilize the full number of processors in any of the settings. This requires running more nodes, hence energy. Is this due to lower performance? This also means that a lot of the processers idle. Besides the point that the GPU is more energy efficient in this case it also indicates that the CPU hardware is not suited for the problem at hand or the code is not suited for the CPU hardware. The most efficient utilizes 1/8 of the cpu cores allocated.

Author Response
Using more nodes is required due to memory constraints. This is included in the manuscript in this paragraph: "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."

Reviewer Comment 16

Table 1: The CPU is listed as both a cpu and a gpu. Is that a mistake? Are both used?

Author Response

The CPU is not listed as both a CPU and a GPU; it is rather that the GPU node contains both GPUs and CPUs. We have added this explanation to the table caption:

A GPU node contains both GPUs and CPUs. Even in GPU-based processes, CPU cores are necessary, for example for transferring data from disk to GPU memory, where computations are performed.

Reviewer Comment 17

Figure 2: Yes, it creates ice but is it realistic? I don't think that the discussion of integration with Veros belongs in the result section as there is no real result (except formation of sea ice).

Author Response

This has been addressed in the response to reviewer comment 11.

Reviewer Comment 18

Figure 3: I cannot see the color difference between the two Veris cpu runs. Please modify

Author Response

We have modified the colors in this figure:

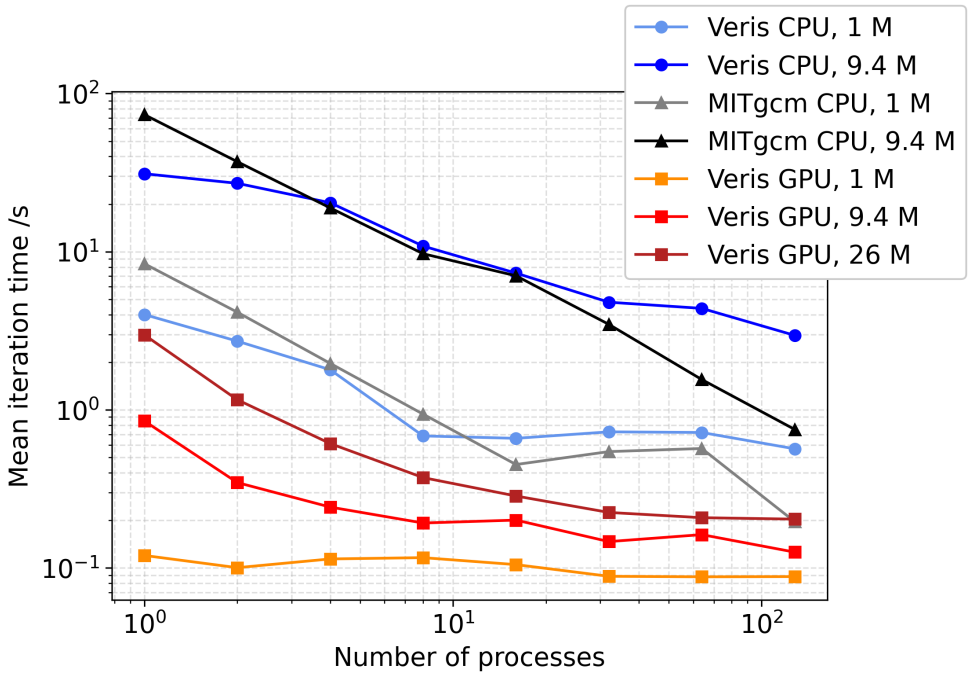

Figure 3: Scaling of mean iteration time with process count for Veris and the MITgcm sea ice component for varying problem sizes (1 M = 10^6 grid cells), calculated over 20 iterations steps.

References

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

Méndez, M., Tinetti, F. G., and Overbey, J. L. (2014). Climate models: challenges for fortran development tools. In *2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, pages 6–12. IEEE.

Rasmussen, T. A. S., Poulsen, J., Ribergaard, M. H., Sasanka, R., Craig, A. P., Hunke, E. C., and Rethmeier, S. (2024). Refactoring the elastic–viscous–plastic solver from the sea ice model cice v6. 5.1 for improved performance. *Geoscientific Model Development*, 17(17):6529–6544.