natESM

Sprint Documentation #17

PISM-AsyncIO

Wilton Jaciel Loch[1], Constantine Khrulev[2], Iris Ehlert[1], and Torsten Albrecht[3,4]

[1] Deutsches Klimarechenzentrum GmbH, Hamburg, Germany
[2] University of Alaska Fairbanks, United States of America
[3] Max Planck Institute for Geoanthropology, Jena, Germany
[4] Earth Resilience Science Unit, Potsdam Institute for Climate Impact Research (PIK), Member of the Leibniz Association, Potsdam, Germany

Contact: info@nat-esm.de

Published on 23.02.2026 on https://www.nat-esm.de/services/accepted-sprints

1 Summary

This sprint addressed performance limitations in the Parallel Ice Sheet Model (PISM) by pursuing two independent objectives: first, the implementation of an asynchronous output server to alleviate I/O bottlenecks at high resolution simulations, and second, the evaluation and optimization of the PICO ocean component, which previously exhibited limited strong scaling.

The main effort focused on developing a Python-based asynchronous output server using YAC and an action-based client–server communication scheme capable of handling PISM's interleaved output operations. While structural differences between PISM's output interface and YAC's phase-based design prevented full support for all output file classes, a fully functional implementation for snapshot files (containing the full set of restart information) was achieved. Performance measurements show a significant reduction in output time compared to the original synchronous approach, with increasing benefits for longer simulations. For the experiment employed in the performance analysis, the average reduction in snapshot writing time is around 98%.

In parallel, performance profiling of the PICO component identified the Eikonal equation solver (used for two-dimensional distance measures) as a major bottleneck. A new algorithm based on a breadth-first search (BFS) approach was implemented and it demonstrated substantially improved execution times. The reduction in the solver execution time with the new implementation ranges from 80% to 98%, resulting in a reduction of 61% to 69% in the total ocean component execution time.

Together, these developments represent substantial progress toward improving PISM's scalability and provide a solid foundation for future extensions and optimizations.

2 General Information

Start and end date:	April 2025 – October 2025
Intended period:	06 months
Responsible RSE:	Wilton Jaciel Loch, DKRZ
Responsible scientist:	Torsten Albrecht, MPI-GEA / PIK

PISM is the Parallel Ice Sheet Model, an open source project (https://github.com/pism/pism), which has been designed for HPC applications on regular grids, capable of simulations at high resolution, using MPI and PETSc tools for computational expensive components. It is a fast C++ code, highly modularized and comes with an extensive documentation and user manual. It provides verification and validation tools and is capable of CF-compliant NetCDF I/O, which allows for easy offline coupling to atmosphere, ocean as well as solid Earth and sea-level model components. The model physics cover a hybrid of shallow or higher-order stress balance approximations, applicable to marine ice sheets and outlet glaciers. PISM comes with a polythermal energy conservation scheme, subglacial hydrology and till module, and various iceberg calving parameterizations (https://www.pism.io/overview). To estimate sub-shelf melt rates and subshelf temperatures based on given ocean temperatures and salinity, we use the PICO ocean module, which mimics the vertical overturning circulation in ice shelf cavities (ice pump) in terms of consecutive coarse boxes (https://www.pism.io/docs/climate_forcing/ocean.html). To assign individual ice shelf indices and to measure geometric distances, e.g. between grounding lines and ice shelf fronts, PICO makes use of the connected-components (CC) labeling method and solves for the Eikonal equation.

3 Sprint Objectives

The sprint had two main independent objectives, both motivated by future performance requirements of the model. These were:

1) Implement an asynchronous output server to more efficiently handle the model data production, specially when going for higher resolutions with multi-million grid points.
2) Evaluate the performance of the PICO ocean component and if time allowed, tackle some of the performance bottlenecks. This was motivated by previous performance assessments done in a PISM sprint check, which showed that PICO had a flat strong scaling curve.

4 Procedure and Insights

4.1 Technical Approach / Procedure

The sprint started with the implementation of the asynchronous output. The first step was to evaluate which tools could be used for the development of the output server. Given that there was in principle no foreseen requirements for the outputs to be done on a grid different than the one used during the simulation, interpolation was not a required functionality of any investigated tool. As possibilities therefore, YAXT, YAC, and ADIOS2 were evaluated. As of the time of evaluation, ADIOS2 only provided support for classic NetCDF files, and was ruled out because of this limitation. YAXT would be a suitable option but it has limited multi-language support (e.g. no Python interface), does not provide high-level auxiliary functions, like time management and metadata communication, and has limited documentation. Therefore, although YAC was not designed to be an output library, it was chosen due to its broad documentation and in-house development support, multi-language interface, interpolation capabilities and auxiliary functions like time-management, metadata, etc. Another important aspect of YAC is that it is very flexible in the ways it can be used, so that simply including it as a data sink opens up the possibility of many other applications on the receiving side, like online coupling, output interpolation and in-situ visualization. The last point in favor of YAC was that it was already integrated into PISM and its build system, as some of its functionalities were already used for the interpolation of input data at runtime.

For the development of the output server, Python was chosen as the language due to its simplicity and support for quick prototyping in comparison to C/C++ and Fortran.

Even though YAC had many benefits in comparison to other tools, there were a number of roadblocks found along the way due to its design being focused on coupling rather than output and to the different ways in which it and PISM operate.

As the original goal was to run the output server without interpolation and replicate the PISM outputs on the other side, the server grid must be the same as the one defined in PISM. For this, all the grid data has to somehow be made available to the server in the initialization. One approach for doing this would be for the server to read the data independently from PISM. However, the information on the simulation grid used in PISM is not guaranteed to be stored in any file – as it can be interpolated from input during runtime. This information thus is only guaranteed to exist partly as a parameter in the PISM command line and later as PISM runtime data.

Both the grid resolution and actual geometrical data could be transferred using the metadata channel of YAC, however especially for the geometrical data this could become rather cumbersome with higher resolutions or with multiple grids. Therefore, an alternative idea was investigated to implement the output server in C++ instead of Python and reuse the grid-building code of PISM to recreate the output server grid. With this approach only the grid metadata would have to be passed through the metadata channel, improving the implementation. However, it was quickly discovered that extracting the grid initialization without the rest of the PISM initialization would be hard and impractical, so this approach was dropped. The final solution was to implement the creation of an MPI intercommunicator between PISM and the output server to use direct MPI communications between both. With this it was possible to send the grid description needed to initialize YAC as well as metadata for grids, fields and their attributes through the intercommunicator.

In terms of output itself, four classes of files are generated with the normal PISM output: timeseries, results, extra and snapshots. Time series are used to save aggregated (1D) quantities, extra files are used to save 2D and 3D diagnostics, while snapshots and result files cover the full range of variables (2D, but also 3D) necessary for a restart of a simulation. Time series files are small compared to the size of the other three, which at least for the employed test experiment were roughly similar. However, many snapshot files can be generated during a simulation depending on the parameters, which may then represent the largest fraction of written data. Because of this, the snapshot files were the initial focus of the output server.

PISM users need to be able to decide at runtime whether asynchronous output will be used by a given simulation. To make this possible, PISM's output code was re-designed to create an output file interface that serves four purposes: 1) to allow users to choose the old (synchronous) output approach when asynchronous output is undesirable, 2) to test new asynchronous code, 3) to limit coupling between PISM and its output code, ensuring that in most use cases information flows only from PISM to the output server, avoiding unnecessary synchronization and 4) to isolate the code in the PISM's source tree that has to be modified during the initial work on the client (sender) part of the asynchronous output implementation.

The development proceeded in a step-by-step fashion, initially focusing on a single snapshot file and adding features incrementally. Once one full snapshot file was able to be written by the output server with correct data, the implementation was extended to support writing multiple snapshots. This was first done in a hardcoded manner, followed by a more generic action-based approach, where the client would tell the server whether more snapshot files should be expected or the execution should be terminated.

At this stage the server had a hardcoded implicit order of communications with the client, and if calls were done in a different order on the other side, the execution would fail. In PISM however, the output interface separates all operations done to files into individual fairly independent subroutines – like file creation, addition of attributes and variables, writing data, etc. Furthermore, all these operations can be called in an interleaved manner, such that you can create a first file and define dimensions for it, followed by the creation of a second file, followed again by writing data to the first file. Figure 1 shows an example of possible communications between PISM and the server, the arrow colors represent two different files.

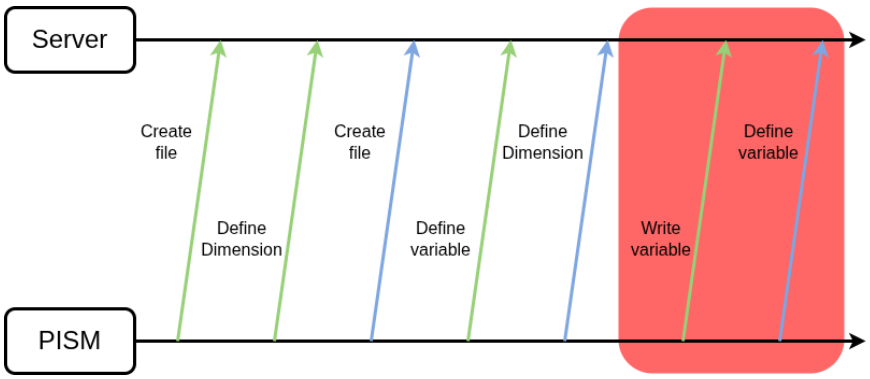

Figure 1: One possible order of file operations performed on PISM. Different arrow colors represent different file classes and the red region represents the conflict for the output server.

With this form of operation, the order of actions in the client can be fairly arbitrary, specially between different files, and to be able to extend the output server to support the other file classes, it required a major overhaul of its structure.

The way forward was to lean on the previously developed incipient action infrastructure and expand it. Actions were created on the server to reflect all possible operations that can be performed on files on the PISM side. These include file, attribute, dimension and variable creation, data writing, file closing and more. This way, the PISM client always sends an action id to the server for every operation. This indicates to the server what to do and what it should expect to receive next, which can be nothing, a JSON-encoded action metadata string and possibly even more communication operations. This rework made the server not only more flexible to support different types of files, but also more flexible to new requirements in general.

The expansion to other files could then continue. However, a major roadblock was found due to the different ways in which YAC and PISM work. The problem is again depicted in Figure 1 and happens because YAC divides its operations into explicitly defined phases while the output in PISM does not rely on the same concept. In YAC, any field can only be defined in the definitions phase, and any field can only be sent after the definitions phase is finished. On the PISM side, due to the interleaved nature of the file operations, one file reaches the data writing point, while another file hasn't yet finished defining its variables. A deadlock situation then forms, where if the YAC definitions phase is ended so that the first file can write, the second file won't be able to define its variables. If the definitions phase is not ended so that the second file would be able to define its variables, then the first file won't be able to write its data.

To solve this problem a change in the PISM output interface is necessary, where PISM will define all the variables and file-variable relations in the beginning of the execution. In this way, all YAC fields can be defined beforehand and just reused later on during the execution. Most of the changes to PISM's output interface needed to avoid this issue were complete by the end of the sprint; all the associated tasks were finished by mid-November 2025.

For the ocean, performance assessments were made to the provided experiment through profiling and tracing. Hotspots were identified, with the main culprits being the Eikonal equation solver and the connected component labeling subroutines. Focus was given mainly to the Eikonal equation solver since the potential for improvement was high in comparison to the foreseen amount of work required to achieve it. It was identified that the approximate discrete algorithm used in PICO to solve the Eikonal equation was suboptimal with many non-necessary checks and loops over the domain. A new version was implemented based on a BFS search and update of the cells in the domain. The new version was compared against the original and showed significantly reduced execution times to achieve the same solution. There was unfortunately no time left to further investigate and improve the PICO code.

4.2 General Insights

The work done during the sprint provided multiple insights and points to consider, especially regarding the implementation of an output server using YAC as the main tool. The first consideration should be how available the grid information is for the output server. In the case of PISM, the runtime grid information was only available within PISM itself and not necessarily in external files. This is not a problem per se as the information can be ultimately sent from the client. But it severely reduces the possibilities in which the server can access this information, which is fundamental for the grid definitions in YAC.

In most scenarios it is useful or even necessary to be able to have arbitrary communications between the client and the server. Since YAC does not provide any native direct communication channel between components it will likely be necessary to create and use an MPI intercommunicator. This is again no problem on its own - and also no large endeavour -, but it should be thought of and familiarized with, as intercommunicators have their own peculiarities within MPI.

The main point to consider when planning to use YAC as a tool for developing an output server is how the output operation is performed in the target client model. It must be kept in mind that YAC works in a phase-based manner and if the client model works with output calls in an interleaved or staggered fashion, deadlocks might occur. To solve this it is necessary to change the client model so that it can declare all the variables that need to be written to any of the output files in the beginning, before any of the output operations.

During the sprint, communication with the maintaining institution of the software in Alaska proved at times challenging. While time-zone differences contributed to coordination difficulties, the main issue arose from extended periods of limited availability on the side of the external development team responsible for maintaining the software.

As in previous sprints following a similar approach, the definition of multiple, partially independent goals helped to reduce the impact of such interruptions. This allowed the RSE to continue work on alternative tasks whenever progress on the primary objective depended on input or feedback that was temporarily unavailable.

5 Results

The sprint results have not completely reached both goals, but represent large strides towards them. An initial version of a working Python-based asynchronous output server was developed and works perfectly for the snapshot files, which represent the file class with the largest volume of written data. Figure 2 shows the measured snapshot writing time with growing simulation lengths for the original synchronous implementation and the asynchronous output server. Each dot represents the total time for writing all the snapshot files in a complete simulation, the presented values are averaged over 5 runs. In both cases PISM runs within a single full node, and for the output server there is one extra task. For each run, snapshots are generated every 5 simulated years.

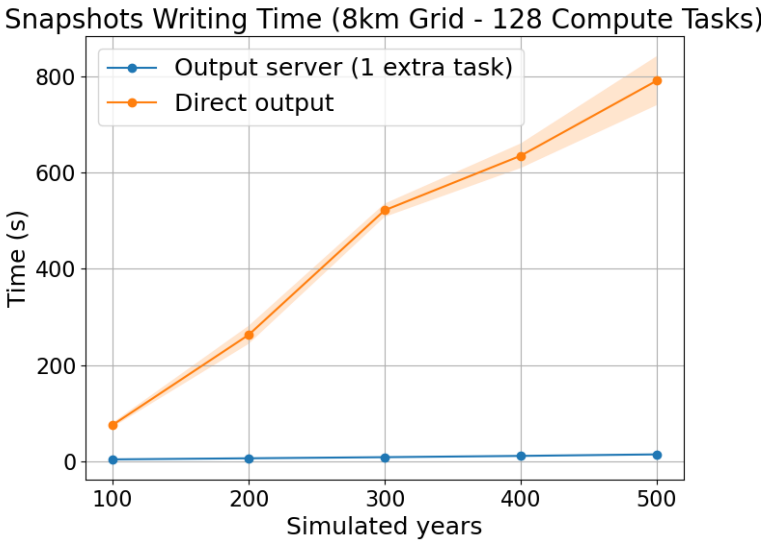

Figure 2: Total snapshot writing time for the original synchronous output and the asynchronous output server with different simulation lengths.

The plot shows that, as expected, for both versions the snapshot writing time grows linearly with the increase in simulation length. However, the slope of the synchronous version is much steeper than the asynchronous version, which results in a dramatic difference in writing time that also grows with longer simulations. For this experiment, the average reduction in writing time for snapshot files is around 98%.

The developed infrastructure code for the output server was already employed as reference for similar developments in other sprints. The action-based communication scheme between client and server is very flexible, and extensions to other types of files or tending to new requirements can be easily done by extending the actions or the action pool.

For the PICO code, a newly implemented algorithm for the Eikonal equation solver was developed. Figure 3 shows the measured execution time of the Eikonal equation solver for the original and new versions of the code. Figure 4 shows the same comparison but for the whole ocean execution time. Although there are clear scaling saturations for this experiment, both plots show that the measured execution time of the new algorithm is significantly smaller than the original one. For numbers of nodes where the scaling is not saturated, the average reduction in time for the approximate Eikonal equation solver ranges from 80% to 98%. For the same range when looking at the whole ocean component execution time, the reduction ranges from 62% to 69%.

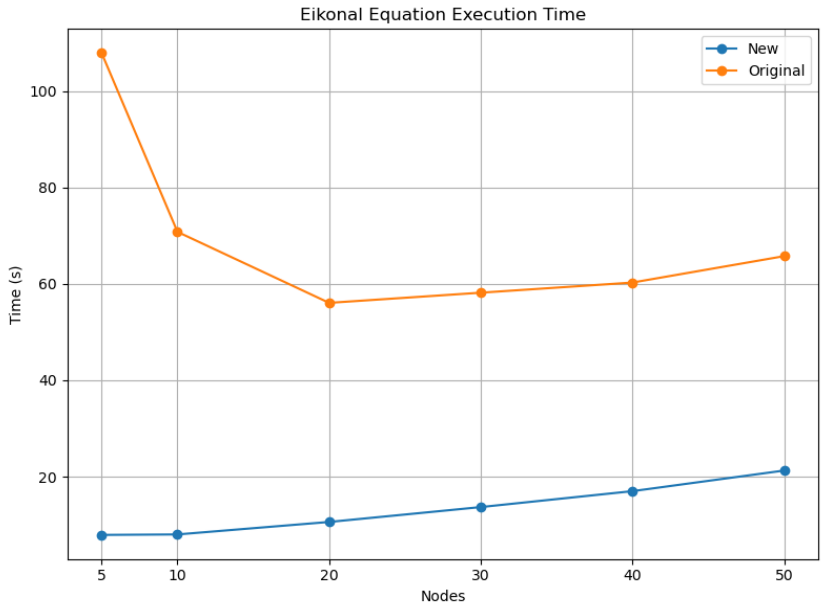

Figure 3: Eikonal equation solution time for original and newly implemented algorithms.

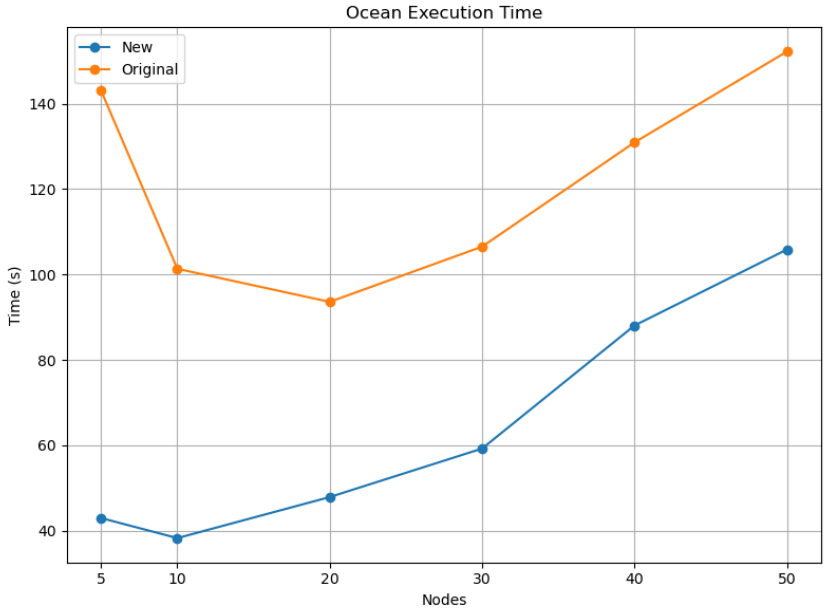

Figure 4: Total PICO execution time for original and new Eikonal equation solver algorithm.

At the end of the sprint both the asynchronous output server [1] and the PICO developments [2] stood as open pull requests on the PISM Github repository [0]. As of early February 2026 both pull requests are merged into PISM's developmental branch and the corresponding changes will be included in the upcoming release.

In summary, as of the time of writing, goal 1 has been fully achieved. During the actual sprint duration, the output server implementation was able to support only snapshot files. The server was then extended by Constantine after the sprint duration to support other types of files, reasonably fulfilling the current requirements in terms of asynchronous output.

Goal 2 was partly achieved, as the performance evaluation was carried out and one of the main performance bottlenecks was substantially improved. However, the behavior of the scaling curve did not change and there was simply a general reduction in the amount of time needed by the PICO component. New performance evaluations should be performed with future versions of the code for possible further improvements.

6 Conclusions and Outlook

The work performed during the sprint laid the foundation for fully reaching both sprint goals. The developed first version of the asynchronous output server is functional and is able to correctly write snapshot files with a significantly smaller time than the original implementation. The action-based client-server communication infrastructure is very flexible, allowing easy changes to the operation of the server or further extensions to support new requirements.

The PICO ocean code was evaluated and one of the main performance bottlenecks was addressed. A new algorithm for solving the approximate Eikonal equation was developed and showed superior performance in comparison to the original implementation, significantly reducing the ocean execution time.

The sprint highlighted the central importance of reliable and continuous communication for successful collaboration, particularly when software is developed and maintained outside the national context, since the practical collaboration depended on the responsiveness of the institution responsible for the ongoing development and maintenance of the software.

For future sprints and for any potential integration into the natESM system, it is therefore essential that maintaining institutions have sufficient capacity to actively support the collaboration and to remain reachable through agreed communication channels. A number of open topics can still be investigated or developed in the future:

For the output server:

- Further investigate the server performance with varying parameters of number of nodes and experiment size
- Investigate the necessity and the possibility of parallelizing the output server

For the PICO performance:

- Redo the performance evaluation of PICO with its newest version and the new version of the Eikonal equation algorithm
- Investigate in detail the connected component labelling algorithm and how it can be further improved
- Investigate if there is a direct influence of the output in the ocean execution time and how it changes by using the output server.

7 References

[0] The PISM GitHub project can be found here: https://github.com/pism/pism

[1] Asynchronous output server pull request: https://github.com/pism/pism/pull/565

[2] PICO eikonal equation solver pull request: https://github.com/pism/pism/pull/564