Sprint documentation #06

ESMValTool Sprint

Jörg Benke[1] and Birgit Hassler[2]

[1]Jülich Supercomputing Center, Jülich, Germany
[2]Deutsches Zentrum für Luft- und Raumfahrt e.V., Oberpfaffenhofen, Germany

Contact: info@nat-esm.de

Published on 13.05.2024 on https://www.nat-esm.de/services/accepted-sprints

1 Summary

The primary objective of this sprint was to enhance the memory efficiency of ESMValTool. This was pursued by systematically integrating Dask functions across the preprocessing chain and replacing NumPy arrays with appropriate Dask arrays. By leveraging Dask's out-of-core computing capabilities, operations are executed on smaller data chunks (which are smaller than memory) instead of loading entire arrays into memory at once. This "lazy evaluation" approach (chunks are only loaded on demand and computation on this data then will be –executed) optimizes memory usage and accelerates processing steps, enabling computations larger than available memory. Furthermore, utilizing Dask's distributed schedulers facilitates easy distribution of computations across multiple nodes on HPC systems. While progress was made in porting ESMValTool for single-node execution, full testing across multiple nodes was impeded by time constraints.

2 General information

Start and end date:	June 2023 – November 2023
Intended period:	6 months
Responsible RSE:	Jörg Benke (JSC)
Responsible scientist:	Birgit Hassler (DLR)

The Earth System Evaluation Tool (ESMValTool) is an open-source software package for the evaluation and analysis of output from Earth System Models (ESMs) that facilitates comparisons of single or multiple models, either against results from predecessor versions or against reanalysis datasets and observations. All output created by ESMValTool is assigned a provenance record that allows for traceability of the results by providing information on input data used, processing steps, diagnostics applied, and software versions used. ESMValTool version 2, released for the first time in 2020, has been optimized for handling the large data volume of the output from CMIP6.

ESMValTool has initially been designed to process and analyze output from CMIP models and observational data sets, but has now been extended to evaluate, analyze, or monitor simulations from individual models such as ICON, providing preprocessing of forcing data for hydrological models, and also include machine learning analysis methods. The large collection of diagnostics and metrics implemented in ESMValTool include, for instance, a set of large-scale diagnostics for quasi-operational and comprehensive evaluation of ESMs, diagnostics for extreme events, regional model and impact evaluation and analysis of ESMs, as well as diagnostics for emergent constraints and analysis of future projections from ESMs. As such, ESMValTool has been used for contributions to several chapters of the Sixth Assessment Report (AR6) of the Intergovernmental Panel on Climate Change (IPCC), in particular Chapter 3 of Working Group 1 on the human influence on the climate system.

3 Sprint objectives

Making ESMValTool more memory efficient would be an important step for the ESM community, since it enables ESMValTool to process output from high-resolution models and thus makes the wide set of diagnostics available for model evaluation and development. This should be achieved through the consequent application of Dask functions throughout the whole preprocessing chain of ESMValTool. An additional aim was to make the ICON output fully UGRID-compliant within ESMValTool during runtime to take advantage of more sophisticated regridding schemes.

A prerequisite to be able to process high-resolution simulations with ESMValTool is to optimize the current memory usage of common preprocessing tasks such as vertical regridding, masking, or calculation of derived variables. At the time of writing the sprint application, only about half of the available preprocessing functions had been optimized by taking advantage of out-of-core computation using Dask's lazy arrays. That leaves many preprocessor functions still reading whole datasets into memory for processing. With an increasing number and size of data files from high-resolution simulations becoming available and model development generally moving towards exascale computing, optimizing the memory usage of the remaining preprocessor functions via Dask is increasingly important to prepare ESMValTool for future applications. This was the main task of the sprint.

As a second objective of the sprint, the Dask schedulers executing the task graphs created with the help of Dask Collections (in case of ESMValTool, mostly the Dask Array class is used) need to be optimized for the specific problems they are implemented for to take full advantage of distributed schedulers. This additional second step will allow for further improved memory efficiency as well as open up the possibility to use more memory by using several nodes of an HPC system and is thus a necessary future step towards getting ESMValTool ready for exascale computing.

In a third step, the capability of ESMValTool to read and process unstructured grids was planned to be extended, which is particularly relevant for processing data on unstructured grids.

4 Procedure and insights

4.1 Technical approach / procedure

The work of the sprint can be divided into three parts (see Section 3 of this document).

- Updating the remaining non-lazy preprocessor functions to use Dask's lazy arrays whenever possible (duration of this task was planned for 4 months). This was the main task.
- Studying Dask distributed schedulers to be able to provide advice on how to further improve memory management of ESMValCore, the core module of ESMValTool performs the actual data processing (duration of this task was planned for 2-4 weeks).
- Updating ESMValCore so that ICON data can be made UGRID-compliant at runtime (duration of this task was planned for 4 weeks).

The sprint started with getting familiar with ESMValTool. This was done via the tutorial at https://tutorial.esmvaltool.org/ which is well organized and a good starting point for learning the core concepts of ESMValTool. Another major source of information was the ESMValTool documentation page with a very detailed description of the tool or for example the workflow for developers. The ESMValTool project repository is located at https://github.com/ESMValGroup/ESMValTool and for ESMValCore this can be found at https://github.com/ESMValGroup/ESMValCore (ESMValCore includes the preprocessor functions the RSE had to work on). ESMValTool and ESMValCore support a full Continuous Integration pipeline with for example linting, style checking, and unit and integration tests (via CircleCI).

The main goal of this sprint was to optimize the memory usage of common preprocessing tasks by taking advantage of out-of-core computation using Dask's lazy arrays instead of NumPy arrays (see https://docs.dask.org/en/stable/array.html and https://docs.xarray.dev/en/v0.9.0/dask.html for Dask arrays). Beside the NumPy library, another important and extensively used library for ESMValCore and ESMValTool is the Iris library, which is a „Python package for analysing and visualising Earth science data" (https://scitools-iris.readthedocs.io/en/stable/). Iris already provides a large collection of high-level functions that are all based on Dask. The total RSEs training period time was approximately 1.5 months (Recap of Python and getting familiar with ESMValTool and ESMValCore, NumPy and Iris library).

The next step of the sprint was to work on making the preprocessor functions lazy which were not available like this yet. All preprocessor functions that require some recoding or restructuring to be made lazy with Dask (or based on Iris) are listed in a GitHub issue (https://github.com/ESMValGroup/ESMValCore/issues/674). The first preprocessor category which the RSE worked on was the masking preprocessors. The first examples to be worked on were the preprocessors "mask_above_threshold", "mask_below_threshold", "mask_inside_range" and "mask_outside_range" which mask out special regions where values are above or below a defined threshold, as well as inside or outside of a defined region. These preprocessors were made lazy by substituting NumPy arrays with Dask arrays (dask.array). This was done successfully, and a test run for each preprocessor with a test scenario (on a single compute node of the Levante supercomputer at DKRZ with 128 cores and 256 GB of RAM) revealed a significant reduction of the runtime by more than a factor of four in some cases after optimizing these preprocessors in ESMValCore (see Figure 1). In addition, the same evaluation can now be done on systems with less memory than the input data size (this has been verified by performing the tests on a shared node on Levante with only a small amount of memory specified). Before these optimizations are implemented here, the evaluation would run out of memory under the same circumstances.

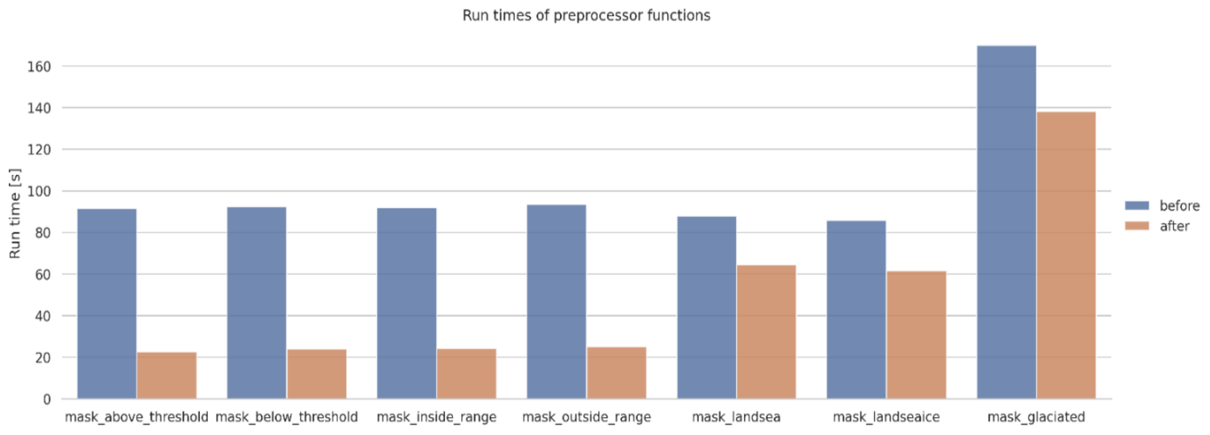

Figure 1: Runtimes of different ESMValTool preprocessor functions before (blue) and after (orange) making the corresponding function lazy. In this example, one single preprocessor function is applied to 65 years of daily 3D data of the CMIP6 model CanESM5. The tests were performed on a single compute node on DKRZ's Levante with 128 cores and 256 GB of RAM.

The process of making the next three preprocessor functions mask_landsea, mask_landseaice and mask_glaciated (to mask out land, sea, ice and glaciated regions) more memory efficient was not as straightforward than for the first four preprocessor functions: a simple function substitution (from NumPy to Dask) was not successful, since Dask implements only a subset of the NumPy array interface. For these three preprocessor functions, the code had to be rewritten to get them more memory and runtime efficient. The tests after the successful porting showed a runtime reduction of about 20-25% (see Figure 1) in these cases.

For the preprocessor mask_fillvalues (which masks multiple input data sets based on the availability of data in the various data sets), much more work was needed to understand the problem and the implemented solution, and how to rewrite it to get it lazy, since there is no direct way to translate this function into Dask. Furthermore, some preprocessor functions were not touched because an adaption of the Iris library was necessary for their porting, which was not possible in the allocated time for the sprint.

Studying the Dask distributed schedulers to be able to provide advice on how to further improve memory management of ESMValCore had been started but did not lead to further advices regarding the improvement until the end of the sprint. Updating ESMValCore so that ICON data can be made UGRID-compliant at runtime was not carried out because of time constraints.

4.2 General insights

First of all, it is to mention that ESMValTool is a very mature and sustainable software with a very good continuous Integration workflow and a very active community. The idea of making the preprocessors lazy and therefore more memory efficient with Dask, in addition to the already extensively used Iris library, is a valid and good approach. Its successful implementation was shown in the ported preprocessors which led to a significant improvement in memory usage and runtime.

However, since the code has grown very complex since the change from ESMValTool version 1 to version 2 and the learning curve especially for ESMValCore is very steep, it was not an easy task for the involved RSE to get familiar with the software in a six-months sprint. Nevertheless, the support from the DLR RSEs and scientists and the ESMValTool community was very good, and it was a pleasure to work in this community.

5 Results

Several Preprocessors were ported successfully with/to Dask with a significant improvement of memory usage and runtime.

6 Conclusions and Outlook

The idea to make all preprocessor functions lazy with Dask, in addition to the usage of the Iris library, is a good and valid approach which was shown to be successful for the modified preprocessor functions. There was a significant improvement in optimizing memory efficiency and runtime for the ported functions. The work of this sprint will be continued in another project (ESiWACE3) which is led by one of the core developers of ESMValTool.

7 References

1. GitHub Repository ESMValTool: https://github.com/ESMValGroup/ESMValTool
2. GitHub Repository ESMValCore: https://github.com/ESMValGroup/ESMValCore
3. ESMValTool Tutorial: https://tutorial.esmvaltool.org/
4. ESMValTool documentation page: https://docs.esmvaltool.org/en/latest/
5. Dask: https://www.dask.org/
6. Dask Tutorial: https://docs.dask.org/en/stable/

7. Iris package: https://github.com/SciTools/iris
8. Iris Tutorials: https://scitools-iris.readthedocs.io/en/stable/