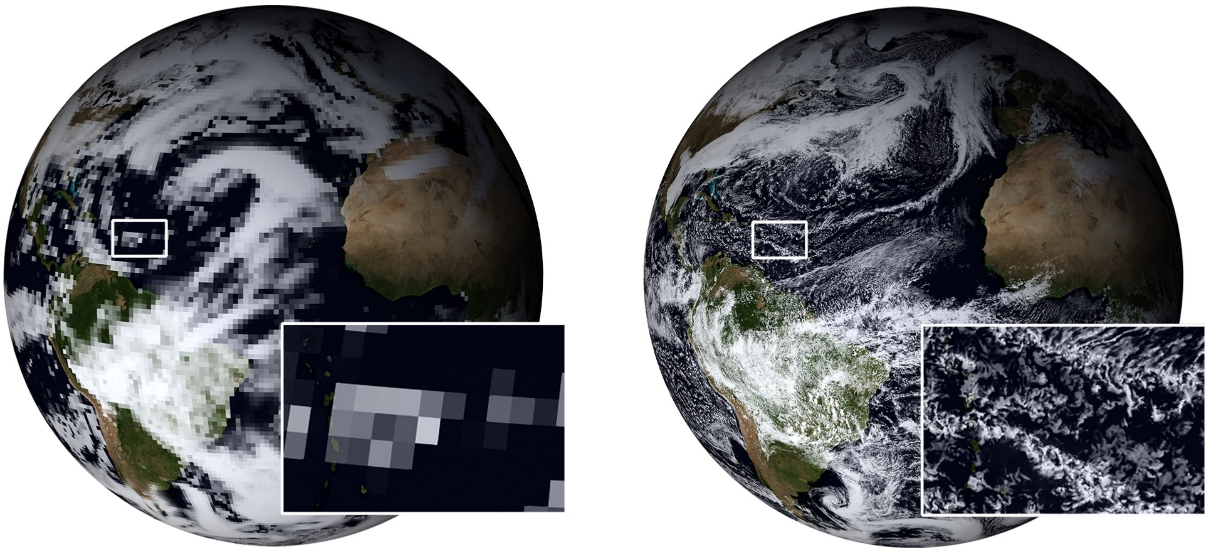
ESiWACE2

Deliverable D5.2

Report on the implementation of the ESDM PAV analytical kernels for post-processing, analysis and visualisation (T5.3)



The project Centre of Excellence in Simulation of Weather and Climate in Europe Phase 2 (ESiWACE2) has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 823988

About this document:
Work package in charge: WP5 Data Post-Processing, Analytics and Visualisation
Actual delivery date for this deliverable: 31 July 2022
Dissemination level: PU (for public use)

Lead authors:
Max Planck Institute for Meteorology (MPI-M) : Oliver Heidmann, Reinhard Budich

Other contributing authors:
Centro Euro-Mediterraneo sui Cambiamenti Climatici (CMCC): Donatello Elia, Cosimo Palazzo
Deutsches Klimarechenzentrum GmbH (DKRZ): Florian Ziemen

Title image: CMIP simulation (left) and DYAMOND Winter simulation (right) — Florian Ziemen, DKRZ

Contents

1 Abstract /publishable summary

The Earth System Data Middleware (ESDM) framework was developed in ESiWACE and can now serve as a platform to execute processing, analysis and visualisation (PAV) workflows in a way that is using meta data of the underlying technology and site specific storage systems on one hand and the workflows themselves on the other hand to automatically optimise data access for performance. This has the potential to remove bottlenecks in the workflows for very large data sets as they are to be expected for the upcoming Exascale systems.

Analytical kernels have been developed further, rewritten and tested for the post-processing and analysis systems CDO and Ophidia to show the interplay with ESDM. For visualisation, there now is an ESDM-enabled netCDF library to accommodate direct input from ESDM into the visualisation tool ParaView. These developments and preliminary tests were successful despite the fact that ESDM currently is not used in a widely spread manor for Earth System Model (ESM) outputs. This seems to be due to the fact that so far its potential has not been shown to be useful enough to replace off-the-shelf, proven, mature systems providing the same functionalities.

This document provides an overview of the analytical kernels developed for CDO and Ophidia making use of accelerated architectures (i.e., GPUs) and of streaming I/O APIs (for in-flight analytics).

2 Conclusion & Results

A set of key analytical kernels have been selected and implemented to exploit novel computing architectures and storage technologies with the aim of providing improved PAV tools on top of the ESDM system.

Below the main conclusions and results from this document are summarised:

- ESDM enabled versions of CDO, Ophidia and ParaView (via netCDF library version) ready to be used;
- GPU enabled versions of specific analytical kernels for complex processing available for CDO;
- GPU versions in many cases with minor performance improvements only for the data sets and machines at hand today, but with potential for future workflows;
- in-flight analytical kernels based on the ESDM streaming API for simple operations (e.g., statistical and arithmetic) developed and integrated in Ophidia;
- use of streaming I/O APIs represents a promising solution to improve the data handling workflow from the storage to the compute nodes by delegating the execution to the storage middleware;
- Further tests on "real" exascale machines with "really" hi-resolution data sets necessary and planned as future activities.

3 Project objectives

This deliverable contributes directly and indirectly to the achievement of all the macro-objectives and specific goals indicated in section 1.1 of the Description of the Action:

Macro-objectives	**Contribution of this deliverable**
(1) Enable leading European weather and climate models to leverage the available performance of pre-exascale systems with regard to both compute and data capacity in 2021.	X
(2) Prepare the weather and climate community to be able to make use of exascale systems when they become available.	X

Specific goals in the workplan	**Contribution of this deliverable**
Boost European climate and weather models to operate in world-leading quality on existing supercomputing and future pre-exascale platforms	
Establish new technologies for weather and climate modelling	
Enhance HPC capacity of the weather and climate community	
Improve the toolchain to manage data from climate and weather simulations at scale	X
Strengthen the interaction with the European HPC ecosystem	
Foster co-design between model developers, HPC manufacturers and HPC centres	

4 Detailed Report on the Deliverable

4.1 Introduction

ESM resolutions are increasing quickly jointly with the simulation data produced. Post-processing, analysis and visualisation of these large datasets call for proper software solutions exploiting novel computing and I/O technologies. In that regard, as GPUs have become ubiquitous in all major supercomputing infrastructures, including those being set up in Europe in the context of the EuroHPC JU (e.g., *LUMI*, *Leonardo* and *Marenostrum 5*), the exploitation of these accelerators could allow speeding up complex data analytics. On the other end, efficient I/O middleware and storage solutions could allow improving data access and reducing data movement, potentially off-loading part of the computation on the storage back-ends, which would in turn result in faster data analysis and visualisation.

In the context of ESiWACE2, we are developing tools to mitigate the effects of the data deluge from high-resolution simulation, i.e., ensemble tools to minimise data output, storage middleware to handle IO performance, and improved tools for post-processing data analytics and visualisation.

This deliverable describes the analytical kernels (i.e., core routines for PAV applications) selected and developed in the context of ESiWACE2 WP5 with the aim of supporting improved post-processing tools, analytics frameworks, and visualisation applications. These kernels have been designed and developed to exploit novel computing architectures and I/O technologies for understanding their effectiveness in PAV applications. The implemented kernels make use of GPU-based architectures to target complex computation (Subsection 4.4) and the Earth System Data Middleware (ESDM) API for in-flight analysis scenarios (Subsection 4.5), and they have been integrated into open source solutions such as CDO or Ophidia. After providing a quick overview of the ESDM (Subsection 4.2) and ESDM-PAV (Subsection 4.3) to show how the PAV applications and new kernels fit into the general picture, the next subsections dive into the details of the different approaches implemented for analytical kernels.

4.2 ESDM

As to ESDM, the DoW of ESiWACE2 says (p42):

"The proposed work will build on a number of developments in ESiWACE1, in particular: 1. Semantic Storage Layers (SemSL). “Intelligent knowledge” about the data held in NetCDF and other formats depends on semantics which exploit the format syntax (e.g. variable attributes) but link them in complex ways. The SemSL exploits the NetCDF CF aggregation conventions to fragment NetCDF files into sub-files which can then be managed (using SemSL or other traditional NetCDF tools) across different storage providers whether local or remote. 2. Earth System Data Middleware (ESDM). Existing scientific software formats provide libraries that provide syntactical support for four-dimensional datasets on disk. ESDM targets the widely used HDF library (upon which NetCDF4 depends) to intercept read/writes via HDF to provide more sophisticated, and higher-performance, use of storage. These two approaches – “above the data format” and “below the data format” – share a common conceptual design philosophy (since they were developed together), but have been developed to address different parts of the problem space. ESDM has been developed to provide performance in simulation and is expected to evolve to provide an “active storage” system which is “weather and climate-aware”. SemSL has been developed to assist in data analysis and data management, and is expected to provide “manual control of storage tiering”. One of the goals of ESiWACE2 will be to further integrate these parts and further differentiate around the distinctive roles (“smart/performant/active”, “simple/performant/manual”). Together, they represent the recognition that the efficient execution of workflows and individual applications within a workflow must be able to harness the different characteristics of a site’s storage systems, and that the balance of services between active and manual can and will differ. Both need to address efficiency, performance portability, manageability, data reduction, and robustness (fault tolerance) of workflows.”

The architecture of ESDM is detailed quite extensively in ESiWACE D4.2 "New Storage Layout for Earth System Data" Luettgau et al. (2017). Here it may suffice to say: ESDM attempts to combine existing

knowledge about site and installation specific properties of the storage system with that of the data formats used, and that of the use cases provided by the users, in order to performance optimize I/O process to/from the storage at hand. As ESiWACE D4.3 "Final Implementation of the Earth System Data Middleware" Pedro (2019) puts it: "From the application perspective, ESDM provides a data model that manages datasets in virtual containers with their metadata."
It has the following key features:

- provides relaxed access semantics, tailored to scientific data generation for independent writes;
- yields better performance via optimised data layout schemes;
- understands application data structures and scientific metadata;
- maps data structures flexibly to multiple storage backends with different performance characteristics based on site-specific configuration;
- separates file metadata operations from data access, which enables the exploitation of acceleration techniques. With traditional NetCDF and GRIB data, the relevant application metadata and much of the information about actual data layout is invisible to the file system (whether parallel or serial); and
- integrates with the NetCDF API as a subcomponent which allows applications to re- place the standard NetCDF library with the ESDM supported library. By utilising standard tools belonging to the ESDM library, import/export files application between ESDM and NetCDF are now possible.

Moreover, as part of the activities from ESiWACE2, ESDM has been extended to support streaming scenarios to accelerate data reads and to potentiallty off-load the computation to the storage back-end servers (if supported by the technology) Kunkel et al. (2020).

4.3 ESDM PAV

The ESDM-PAV runtime supports the execution of post-processing, analytics and visualisation (PAV) experiments composed of multiple tasks (i.e., a workflow) through the definition of a PAV experiment document (in JSON format). The module builds on top of the ESDM for the I/O operations and supports the scheduling and execution of PAV applications.

The experiment definitions executed by ESDM-PAV can include sub tasks that are from Ophidia, CDO and Paraview and support different types of abstractions, such as flow and data dependencies, flow control and error management.

In order to integrate the applications with the ESDM-PAV and the ESDM API modifications to the Ophdia framework were required. Some minor modifications where also required for the integration of ParaView with ESDM, while CDO only needs to be compiled with the ESDM thus not requiring any change in the code.

Once the task description is submitted, the ESDM-PAV runtime will handle the execution and scheduling of all the sub tasks according to the experiment definition. The scheduling also handles task distribution to different nodes, including tasks that make use of GPU acceleration, to concurrently run the different tasks.

A Python-based module (i.e., the ESDM-PAV client) has also been developed to provide a high-level interface for building PAV experiments composed of different tools, as well as to handle and monitor their execution on the runtime. Figure 1 shows a simplified view of the ESDM-PAV architecture, highlighting the integration with PAV tools and the ESDM API. A more detailed description of the ESDM-PAV runtime and related Python module can be found in Deliverable D5.1 Elia et al. (2021b).

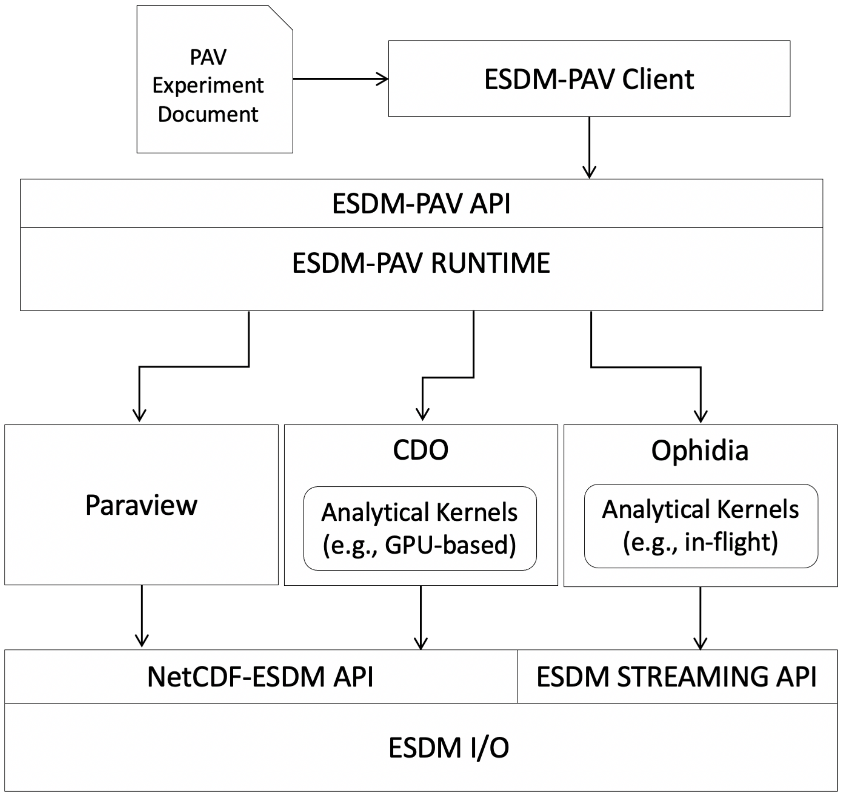


Figure 1: High-level overview of the ESDM-PAV architecture and its integration with PAV tools and the ESDM I/O API

4.4 CDO

CDO is a collection of command line Operators to manipulate and analyse Climate and NWP model Data. Supported data formats are GRIB 1/2, netCDF 3/4, SERVICE, EXTRA and IEG. There are more than 600 operators available. It is developed at MPI-M for many years, and supported by the German Climate Computing Centre DKRZ. The CDO operators are "... a collection of many operators for standard processing of climate and forecast model data. The operators include simple statistical and arithmetic functions, data selection and subsampling tools, and spatial interpolation. CDO was developed to have the same set of processing functions for GRIB and NetCDF datasets in one package." Multiple operators can be executed in parallel by chaining them using polish notation.

```
cdo -subc,-13 -select,jan -add input_1 input_2 result_file
```

Shell 1: CDO example call with chained operators

4.4.1 CDO: Optimized Analytical Kernels, see (T5.3)

Modern data sets are ever increasing in size. Today's CPUs, designed to handle a wide range of tasks, are not the most efficient tool for some tasks required in our field. Even with multiple CPUs on the same node (shared memory) or distributed CPUs (each CPU with their own memory) there are limits to how much acceleration can be gained by parallelism. For example the core count per CPU: there are limits with respect to heat and space which govern how fast a core can be, and how many cores can be on a single die. For distributed systems the communication between the nodes can heavily limit the total efficiency of the system as synchronisation of the calculations between the nodes can result in wait times which cause the system to

use only a fraction of it's actual potential.
GPUs, while bringing their own set of problems, can alleviate some of the problems in this context. They are used in tandem with CPUs. Some tasks which CPUs can not handle efficiently due to their generic purposes are outsourced to the GPUs: Their high specialisation for float and double precision numeric calculations suits them much better for the said tasks. In addition GPU excel when they are used to run tasks that have a relatively low amount of memory accesses while having a high to extreme amount of floating point operations. Because of these possible benefits the CDOs have been analysed to find out if introducing GPU accelerated routines would make the CDOs ready to tackle future exascale data sets and work loads to be efficiently processed.

Experiments outside of CDO Preliminary to implementation of the Analytical Kernels members of the ESiWACE partner Manchester University did tests for several kernels extracted from the CDOs. This was intended to gauge the possible performance improvements of these kernels independent of the CDO implementation.
All in all we decided on three CDO routines which we adapted for GPU acceleration:

- A simple Arithmetical Kernel
- Vertical Interpolation
- Fast Fourier Transformation

First, for a general idea of how to rewrite the code for GPU acceleration, a very simple arithmetic kernel was chosen. Later we identified two more CDO routines for which we expected relevant performance gains.
The first choice of the latter two was the Vertical Interpolation routine. This routine needs to load a whole grid into memory where other routines work on a record for record basis. With a whole grid we expected minimal performance loss through moving data from GPU to the CPU and vice versa as these could be done in a single step instead of moving each record on its own.
The second choice was the Fast Fourier Transformation (FFT) for which a ready to use GPU implementation in form of the CUFFT library exists. This was chosen to test our options for extending CDO through usage of external GPU accelerated algorithms. CUFFT itself is a Nvidia library included in the NVIDIA-toolkit library. This library includes GPU optimized Fast Fourier Transformation routines which are what we introduced as an alternative to CDO's FFT routines. CUDA is NVIDIA's GPU programming interface library and allows to write routines that can be executed on GPUs. Handling of data transfer's from and to GPU memory as well as synchronisation mechanism are included as well.

Changes to the CDO Build System For the kernels we needed to upgrade the CDO build system to handle ACC and CUDA code. For the CUDA code we added new configure options to enable compilation of CUDA files which are later linked to rest of the program. In addition to the configure flags we added an option to CDO itself which enables or disables the use of GPU acceleration. This was introduced to allow testing with the same executable as well as giving the users, depending on their data, the choice if it is useful to use GPU acceleration or not. Depending on the data it, might be useful to not employ the GPU version since for small data sets the overhead of moving data from CPU RAM to GPU VRAM can be more expensive than the regular CPU execution.

4.4.2 Analytical Kernels Description

In this subsection we describe the re-implemented routines in more detail. We will also explain in which operators these kernels are contained and give examples for what they are used commonly.

Simple Arithmetical Kernels For testing purposes and to explore the necessary requirements for CDO to use CUDA we implemented a very simple version of an arithmetical kernel. This routine does contains simple arithmetic operators such as addition and multiplication and is not expected to benefit from GPU acceleration. We implemented these as they were an easy target for testing and implementing the necessary changes for extending the building system of CDO to include Nvidias compiler for CUDA code and management of the different required file types for CUDA files.

Vertical Interpolation Within CDO the vertical interpolation is contained in the operator intlevel. For example with cdo `-intlevel,50/9000/100 infile outfile` we interpolate to get new layers from zero to 9900 with the result that new layers at 50 to 9850 are generated. This operator is often required to process data that does not have the required number of vertical steps. Raw data coming from observations often does not confirm to the requirements of simulations. With `intlevel` these data sets can be modified to adhere to the required structure. The GPU acceleration of these routines is handled by ACC, an CUDA based system that allows code to be transformed into GPU routines by using ACC's pragmas.

```
#ifdef _OPENACC
  Varray<int> lev_idx_1(nlev2);
  Varray<int> lev_idx_2(nlev2);
  Varray<float> lev_wgt_1(nlev2);
  Varray<float> lev_wgt_2(nlev2);

  if (Options::cdoVerbose) cdoPrint("Using GPU version of %s", __FUNCTION__);

  for (int ilev = 0; ilev < nlev2; ++ilev)
    {
      const auto idx = lev_idx[ilev];
      const auto wgt = lev_wgt[ilev];
      restore_index_and_weights(nlev1, idx, wgt, lev_idx_1[ilev], lev_idx_2[ilev],
       ↪ lev_wgt_1[ilev], lev_wgt_2[ilev]);
    }
  size_t dataSize = gridsize * nlev2;
#pragma acc data copyin(vardata1 [0:dataSize])
#pragma acc data copyin(lev_idx_1 [0:nlev2])
#pragma acc data copyin(lev_idx_2 [0:nlev2])
#pragma acc data copyout(vardata2 [0:dataSize])
#pragma acc parallel loop collapse(2)
  for (int ilev = 0; ilev < nlev2; ++ilev)
    {
      for (size_t i = 0; i < gridsize; ++i)
        {
          auto var1L1 = &vardata1[gridsize * lev_idx_1[ilev]];
          auto var1L2 = &vardata1[gridsize * lev_idx_2[ilev]];
          auto var2 = &vardata2[gridsize * ilev];
          var2[i] = vert_interp_lev_kernel<float>(lev_wgt_1[ilev], lev_wgt_2[ilev],
           ↪ var1L1[i], var1L2[i], missval);
        }
    }
#else
  cdoAbort("Option --gpu was used but openacc support is not available");
```

```
#endif
```

Shell 2: ACC pragmas used for interpolation routine

FFT Fast Fourier Transformations are used to transform e.g. time series into a sum of trigonometric functions. These functions and the corresponding coefficient of each can be summed up to reconstruct the original time series. This can be used to transform spectral grids to points grids and back.

```
cdo lowpass,1 testfile_filter.nc result_filter.nc
```

Shell 3: CDO lowpass command using FFT

4.4.3 Generation of CDO Test Data

For all our tests we used CDO to generate suitable data sets. In this section we will include a short explanation of how we generated the data using CDO. CDO includes the unofficial cdiwrite operator which allows us to create sample data for our testing scripts. Cdiwrites parameters are as follows:

```
<nruns, <grid, <nlevs, <ntimesteps, <nvars>>>>>
```

- nruns: number of write cycles;
- grid: gridtype
- nlevs: number of vertical layers;
- ntimesteps: number of Timesteps
- nvars: number of variables;

In discussions about realistic data for the vertical interpolation we found that a 8000x4000 grid with 100 to 200 layers and a single variable should be the starting point for test runs. Sadly files of these sizes already result in problems on Mistral in terms of available VRAM. Because of this we wrote tests that try to get as near as possible to the realistic sizes of files.
For the FFT tests we created our data with the following cdo commands:

```
cdo -f nc4 -del29feb -cdiwrite,1,r800x400,1,3653,1 temp_result.nc
cdo -f nc4 expr,'var1=var1*sin(ctimestep())' temp_result.nc testfile_filter.nc
```

The output of the first call (temp_result) contains a single horizontal layer with a size of 800 x 400 and 3653 time steps. In addition we require the use of the del29feb operator to have each year with the same number of days. This was followed by setting the values of each point with the expr operator so that we do not have a grid filled with only the default values.

4.4.4 Workstations

For a first view into the environment necessary for GPU acceleration in CDO we used our improvements of the CDO build system to build CDO on a regular workstation currently in use at MPI-M. This proved to be far less complicated than working on Mistral. But the capabilities of such workstations proved to be not sufficient for data sets of relevant size. In addition we found that a distribution of CDO to workstations would be problematic. These problems stem from the fact that CDO would need to be compiled on the workstation with different configurations depending on the present hardware. Even if the configuration was not necessary CDO, or - more specifically - the accelerated parts of CDO would need to be recompiled to be able to work with the present hardware. Compilation of CDO should not be expected of the general user and as such a distribution to the general user-base would or should not include the acceleration mechanisms. Pre-installed CDO versions on the other hand could make use of future GPU routines if further tests show that using that CDO version with average workstation hardware provides relevant benefits.

4.4.5 Mistral

Currently the only HPC system we were able to run the kernels on was DKRZ's Mistral. Here we used Mistrals GPU nodes of which some contain a NVIDIA A100 graphics card with up to 80GB of VRAM. In first tests on other GPU nodes without a A100 we noticed that for relevantly sized data the VRAM was not enough and grid sizes of 4000x4000 caused crashes due to VRAM limits. The following list shows which dependencies or rather modules had to be available for running the CDO routines on Mistral.

```
  1) cdo/1.9.10-magicsxx-gcc64          9) libaec/1.0.2-pgi-19.9
  2) imagemagick/6.9.1-7-gcc48         10) libtool/2.4.6
  3) lftp/4.8.1-gcc-9.1.0              11) ncl/6.5.0-gccsys
  4) netcdf_c/4.3.2-gcc48              12) pftp/7.5.1.2-krb5
  5) git/2.28.0-gcc-9.1.0              13) pgi/19.9
  6) defaults                          14) the-silver-searcher/2.2.0-gcc-9.1.0
  7) autoconf/2.69                     15) cuda/10.0.130
  8) automake/1.16.1
```

Shell 4: Required Mistral modules for CDO GPU acceleration

The following scripts shows the configure settings necessary for CDO to work with GPU acceleration. In combination with `export PGI_LOCALRC='/mnt/lustre01/work/mh0287/m300488/pgi/gcc64.localrc'` this produces a working configuration on a Mistral GPU node . The configuration and the compilation need to be done on the node itself since the CUDA components rely on the hardware they are supposed to be run on. The configure setup script explains what is required and needs to be setup for a successful build/execution of the accelerated routines. While this configuration is specifically for Mistral it should provide a good base for other systems as well.

```
set -x
echo  General
CUDA_VERSION=cuda-10.0.130
CDO_MAIN_FOLDER=../cdo

echo  Mistral specific. Just as example.
Mistral_LD_CUDA=-L/sw/rhel6-x64/cuda/${CUDA_VERSION}/lib64
Mistral_LD_PGI=-L/sw/rhel6-x64/pgi-19.9/linux86-64-llvm/19.9/lib

Mistral_NETCDF=/sw/rhel6-x64/netcdf/netcdf_c-4.6.1-gcc64
Mistral_proj=/usr/lib
Mistral_netcdf=/sw/rhel6-x64/netcdf/netcdf_c-4.6.1-gcc64
Mistral_hdf5=/sw/rhel6-x64/hdf5/hdf5-1.8.14-threadsafe-gcc48
Mistral_szlib=/sw/rhel6-x64/sys/libaec-0.3.4-gcc48
Mistral_udunits2=/sw/rhel6-x64/util/udunits-2.2.17-gcc48
Mistral_fftw3=
Mistral_cuda=/sw/rhel6-x64/cuda/${CUDA_VERSION}
Mistral_curl=
Mistral_ossp_uuid=
Mistral_proj=/sw/rhel6-x64/graphics/proj5-5.2.0-gcc64

echo  GPU Options
REQUIRED_PGI_LIB_FLAGS="-lpgf90 -lpgf90_rpm1 -lpgf902 -lpgf90rtl -lpgftnrtl -lnspgc
↪  -lrt -lm -lgcc -lc -lgcc"
GPU_TARGET=-ta=tesla:managed
ACC_ENABLED=-acc
MINFO=-Minfo=accel

echo  Compilers
CDO_CC=pgcc
CDO_CFLAGS=-D_REENTRANT=1
CDO_CXX=pgc++
CDO_FC=pgfortran
CDO_CXXFLAGS="-std=c++17 -O3 ${ACC_ENABLED} ${MINFO} ${GPU_TARGET}"

echo  LIBS
CUFFT_LINKER_FLAG=-lcufft
CUDA_LINKER_FLAG=-lcudart
CDO_LIBS="${REQUIRED_PGI_LIB_FLAGS} ${CUDA_LINKER_FLAG} ${CUFFT_LINKER_FLAG}"

echo  PATHS
LD_CUDA=${Mistral_LD_CUDA}
LD_PGI=${Mistral_LD_PGI}

echo  LDFLAGS
CDO_LDFLAGS="${LD_CUDA} ${LD_PGI}"

echo  configure options
CDO_CONFIG_OPTIONS="--with-proj=/usr/lib"
CDO_CONFIG_OPTIONS+=" --with-netcdf=/sw/rhel6-x64/netcdf/netcdf_c-4.6.1-gcc64"
CDO_CONFIG_OPTIONS+=" --with-hdf5=/sw/rhel6-x64/hdf5/hdf5-1.8.14-threadsafe-gcc48"
CDO_CONFIG_OPTIONS+=" --with-szlib=/sw/rhel6-x64/sys/libaec-0.3.4-gcc48"
CDO_CONFIG_OPTIONS+=" --with-udunits2=/sw/rhel6-x64/util/udunits-2.2.17-gcc48"
CDO_CONFIG_OPTIONS+=" --with-fftw3"
CDO_CONFIG_OPTIONS+=" --with-cuda=/sw/rhel6-x64/cuda/${CUDA_VERSION}"
CDO_CONFIG_OPTIONS+=" --with-curl"
CDO_CONFIG_OPTIONS+=" --with-ossp-uuia"
CDO_CONFIG_OPTIONS+=" --with-proj=/sw/rhel6-x64/graphics/proj5-5.2.0-gcc64"

#combining everything
${CDO_MAIN_FOLDER}/configure CC=${CDO_CC} CFLAGS=${CDO_CFLAGS} CXX=${CDO_CXX}
↪  FC=${CDO_FC} CXXFLAGS="${CDO_CXXFLAGS}" LIBS="${CDO_LIBS}"
↪  LDFLAGS="${CDO_LDFLAGS}" ${CDO_CONFIG_OPTIONS}
```

Shell 5: CDO configuration script for GPU enabled kernels

The choice to use the older PGI compiler instead was based on software limitations on Mistral. A lot of the complexity of the build and configure process stems from PGI. More recent compilers are more streamlined in context of GPU acceleration and we expect a reduction of complexity when Levante is fully online and the more recent software stack is available.

4.4.6 Preliminary Results

In initial testing we found that no relevant speedups could be achieved. Using grids of size 3000 x 4000 and larger caused the routines to fail. At first we thought this was due to high ram and especially VRAM usage, but later tests have shown these setups to run properly now. In our initial tests we were unable to read relevant grid sizes and only the recent changes on Mistral itself then allowed us to run the benchmarks with meaningful input data. Still we have found that the previously extracted, adapted and now reintroduced CDO routines did under-perform when compared to using only the routines outside into CDO.

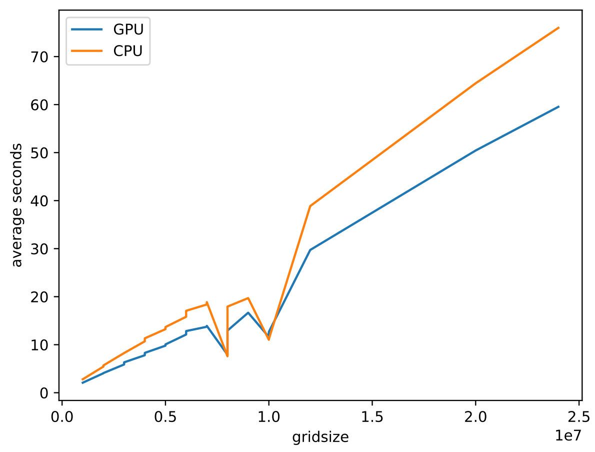


Figure 2: Preliminary Results CPU vs GPU, GPU speeds not as expected

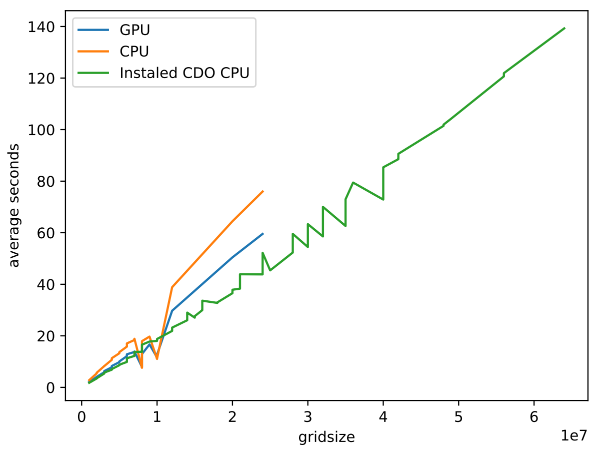


Figure 3: Preliminary Results: Pre-installed version with better optimization still outperforms manually compiled not optimized routines

These results still have to be re-checked since when running the pre-installed version of CDO we observed that both the self-built CPU und GPU versions of the CDOs perform worse (see fig: 3) than the further developed original code. This is attributed to optimization not set to the same level when compiling CDO manually. The missing optimization level could also be a reason for the drastic discrepancies between the extracted routine performance and our own benchmarks.
In addition to these performance results we also found that accelerated routines do most likely not make sense on workstations although the configuration on a personally owned workstation was a lot less complex than working with modular software stacks as seen on Mistral and in the future on Levante. We expect similar complex configurations to be necessary on other HPC machines.
A core problem was and could still be that the software stack on the HPC machines can be out of date by years. In the field of GPU acceleration that means that the lack of modern compilers and libraries leads to very complex and error-prone configurations. We could observe some of these problem to be less pronounced on Levante and have high hopes that the more up-to-date software significantly reduces the necessary steps to get CDO and its GPU kernels to work.

4.4.7 Next Steps

In this section we present the, in the context of CDO and GPU acceleration, next steps which we split into 4 categories. The first category are changes and tests to CDO itself where the other steps describe our plans in regard to other HPC platforms where we will test the changes made to CDO

- CDO

  Due to large differences in performance between the CDO and non-CDO versions of our tests we will profile CDO in terms of bottlenecks as well as checking available optimization options. We hope to find and document weaknesses or at least possible improvements of CDO for future GPU-accelerated routines. Should further benchmarks show significant performance improvements we plan to include, pending discussion with the main maintainers of CDO, the changes into an appropriate release of CDO. This release would include proper documentation on the usage and necessary configuration for the new enhancements as well as developer documentation for further work on GPU acceleration in context of CDO.

- Mistral

  Due to Mistral being replaced by Levante (see below) testing on Mistral is no longer a possibility. Earlier results on Mistral were disappointing and were far off of our expectations. In addition problems with the environment caused crashes with data sets of relevant size. The problems mentioned had been fixed on Mistral, and further reevaluation of previous results had been scheduled, but now had to be omitted.

- Levante

  Levante, the new DKRZ high performance computing cluster succeeding Mistral, is described here: `https://www.dkrz.de/en/systems/hpc/hlre-4-levante`. It is now online so that we will use this system to test our routines. For this the previous configurations from Mistral are now adapted. We were positively surprised about the availability of modern software packages that will relieve some of the problems described in the previous paragraphs.

- Lumi

  One of the largest, most powerful machines worldwide is available to ESiWACE partners: The Lumi machine, see here: `https://www.lumi-supercomputer.eu/lumis-full-system-architecture-revealed/`, for the complete system architecture. It sports about 2500 nodes with a ratio of cpu to gpu cores of 1 to 5 by AMD, delivered by the HPE. The machine is now ready for first tests, and the software environment seems to approach a stable phase. So it is worth while to test if the differences in the architecture between Levante and lumi result in notable and/or relevant differences, which might have influence on our recommendation not to install the GPU-version of the CDOs as a default for a (GPU enabled) site.

  For Lumi we first plan to evaluate the amount of work necessary to get CDO to compile and run there. Should these tests have positive results we will rerun the benchmarks on Lumi.

4.5 Analytical kernel developments for the Ophidia framework

Climate analysis often includes complex operations to be executed on the input data, such as interpolation, regridding, low pass filtering, etc., which require efficient parallelization strategies and adequate compute resources for to ensure an adequate level of performance. During these stages, data must be transferred from the storage nodes to the compute nodes to perform the analysis and extract the results, which might eventually be moved back to the storage system. As the volumes of data produced by ESM simulations is continuously increasing, data movement is becoming more expensive in terms of transfer delay, high bandwidth utilization and energy consumption, especially during the workflow I/O stages. Hence, it is of paramount importance to explore novel solutions to reduce data movement.

From this perspective, innovative *active storage* solutions could provide the means to reduce data movement by off-loading part of the processing activities on the storage back-end servers. These could allow, for certain types of processing, a great reduction in the volume of data transferred from storage to the compute nodes of the HPC system. Of course, storage back-ends could only run simple operations such as statistical aggregations, mathematical functions or subsetting. Despite their simplicity, applying these functions as the first step in the processing pipeline, before moving the data, could allow a significant reduction in the volume of transferred data, as only the results of the functions will be moved, which would speed up the whole process.

In order to enable this type of functionalities within data analytics applications, specific I/O APIs are required, supporting access to streams of data while also applying custom functions. An approach that considers this type of pre-processing, where part of the computation is delegated to the storage interface and performed while data is being fetched and transferred from the storage to the compute system is here defined as *in-flight data analytics*. As described above, this could benefit from *active storage solution*, i.e., storage back-ends that can run basic functions, as part of the computation could be directly off-loaded to the storage nodes before streaming the data.

In the context of the ESiWACE2 WP5, in-flight analytics is one of the approaches explored to improve data analytics tools and it has been decided to integrate it directly at the level of PAV applications, as briefly explained in Elia et al. (2021b).

In particular, to support in-flight data analytics, the streaming interfaces of the ESDM library (i.e., *esdm_read_stream*) are exploited to apply a *compute kernel* on data during loading from storage and to read the corresponding results only from the PAV application. As reported in Fiore et al. (2019) and Kunkel et al. (2020), this extension to the ESDM API can be used to read an ESDM dataset and to register the compute kernel, potentially offloading computation to the storage back-end, when supported. Through this interface, instead of directly reading the whole data, the *esdm_read_stream* interface applies the kernel operations and returns the results of computation to the caller. As a proof of concept (PoC) the designed in-flight analytics have been integrated through the ESDM API into the Ophidia HPDA framework.

4.5.1 The Ophidia Framework

Ophidia is a CMCC Foundation research effort addressing Big Data challenges for eScience Fiore et al. (2013). The Ophidia framework is an open source (GPLv3) solution for the analysis of scientific multi-dimensional data, mainly targeted at climate science applications, although it has also been successfully exploited in other scientific domains (e.g., seismology, biology, fire prevention). It joins HPC paradigms and Big Data approaches for providing a framework for High Performance Data Analytics (HPDA). The framework supports parallel and in-memory data processing, data-driven task scheduling and server-side analysis. It exploits an array-based storage model, leveraging the datacube abstraction from OLAP systems, and a hierarchical storage organisation to partition and distribute large multi-dimensional scientific datasets over multiple nodes Elia et al. (2021a).

The framework includes more than 50 operators providing features such as data loading and storing, temporal resampling, data aggregation, intercomparison, subsetting, statistical analysis, transformation, etc. The full list of supported operators is reported in the software documentation[1].

More specifically, the extensions for in-flight analytics have been added to the ESDM-enabled Ophidia operators for data loading (e.g., OPH_IMPORTESDM) developed in the context of ESiWACE2 and reported in Bethke et al. (2021). This integration has been performed considering *usability* as a key requirement, so that the kernels could be easily exploited by the user without having to understand the underlying mechanics. The in-flight analytics operations in Ophidia can be enabled, in fact, by simply specifying the required operation through an additional argument to the import operator. Figure 4 shows the revised Ophidia architecture highlighting the integration with the ESDM middleware and the use of the analytical kernels.

It is worth mentioning that the integration approach followed to support in-flight analytics in Ophidia could be easily generalised also for other I/O solutions, besides ESDM, which provide the APIs for applying

[1]Ophidia operators documentation: `https://ophidia.cmcc.it/documentation/users/operators/index.html`

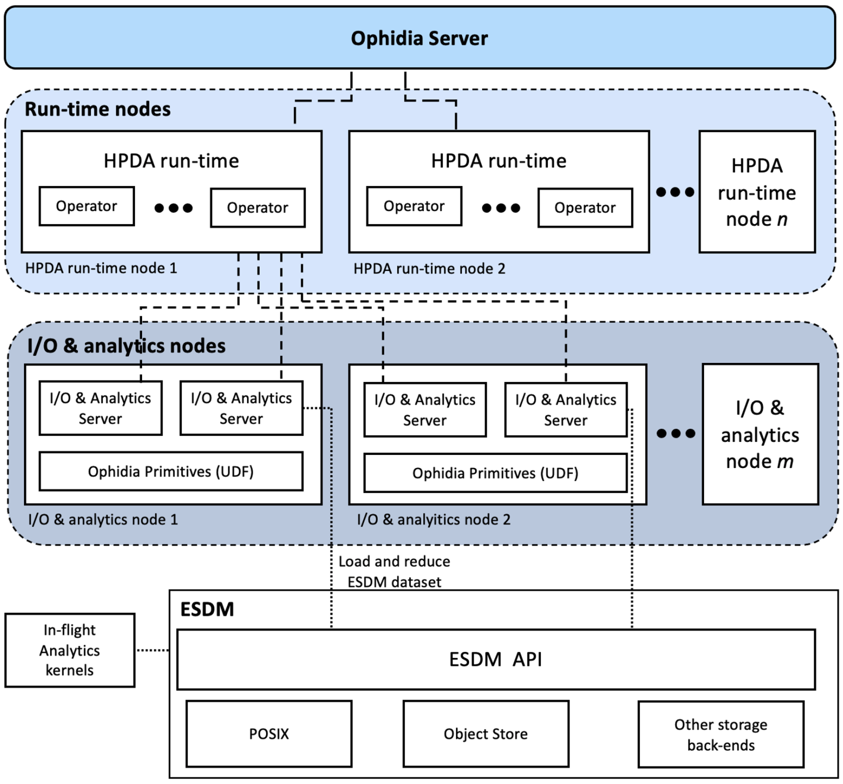


Figure 4: High-level architecture of the ESDM-enabled Ophidia framework. The in-flight analytical kernels are selected by the specific operator and executed through the ESDM API.

the kernels on data streams and potentially offloading computation to the storage layer. Specifically, new Ophidia operators could be developed retaining the same interface as those implemented for this PoC.

The following subsections will dive into the technical implementation details, including the list of available functions and some usage examples.

4.5.2 Integration of ESDM Streaming API into the Ophidia operators

The ESDM streaming API has already been reported in Fiore et al. (2019); here, for the sake of clarity, we reprise the read streaming function used for the integration.

The signature of the ESDM read streaming function, called *esdm_read_stream*, is:

```
esdm_status esdm_read_stream(
    esdm_dataset_t *dataset,
    esdm_dataspace_t *space,
    void *user_ptr,
    esdm_stream_func_t *esdm_stream_func,
    esdm_reduce_func_t *esdm_reduce_func);
```

Here *dataset* and *space* define the dataset to be loaded, *user_ptr* a pointer to a user-defined struct containing metadata to be passed to the kernel (e.g., its input parameters) and, finally, the functions pointers *stream_func* and *reduce_func* define the specific compute kernel to be registered. The ESDM streaming interface has been defined so that a kernel can be executed as a classic map-reduce operation with the first of the two pointers representing the function applied during the *map* stage, while the second one providing the function applied during the *reduce* part.

The interface of the first function *stream_func* is:

```
void *esdm_stream_func(
    esdm_dataspace_t *space,
    void *buff,
    void *user_ptr,
    void *esdm_fill_value),
```

This function accesses an ESDM data fragment (i.e., called *buff*), processes it by exploiting the information carried by the user-defined struct *user_ptr* and returns intermediate results within a user-defined struct (called *stream_func_out* in the next function).

The interface of the second function *reduce_func* is:

```
void esdm_reduce_func(
    esdm_dataspace_t *space,
    void *user_ptr,
    void *stream_func_out)
```

This function processes the user-defined struct *stream_func_out*, representing the outputs of *stream_func*, to produce the final output to be returned to *esdm_read_stream* caller: the function is applied more times, one for each ESDM data fragment that the input ESDM dataset is split into. This phase could be considered as a reduction phase: the intermediate results obtained during the mapping phase (*stream_func_out*) are gathered to produce the final dataset that will be returned, and it represents the (few) data actually returned by the storage middleware.

For the aims of the ESiWACE2 project, as previously mentioned, a number of compute kernels have been developed as an extension to be used through the Ophidia platform. In particular, the ESDM import operators (e.g., OPH_IMPORTESDM), designed for WP4 Bethke et al. (2021), have been enhanced with the read-streaming option of the ESDM library in order to run in-flight analytics operations while loading an ESDM dataset, and potentially exploit active storage capabilities by offloading computation to the storage system.

4.5.3 Compute kernels implementation details

This section provides an overview of how the stream and reduce functions from the ESDM interface have been implemented for the definition of the kernels.

The following definition (Listing 6) considers the specific user-defined struct used to transfer metadata to kernels from OPH_IMPORTESDM, which must be filled and passed to *esdm_read_stream* as reported above.

```
typedef struct _esdm_stream_data_t {
    char *operation;
    char *args;
    void *buff;
    char valid;
    double value;
    double value2;
    uint64_t number;
    void *fill_value;
} esdm_stream_data_t;
```

Shell 6: User-defined struct used to transfer metadata to compute kernels

In particular, the attributes of this struct represent:

- **operation** indicates the name of the kernel to be executed; in the current implementation, it is only possible to consider one operation at a time;
- **args** is used to set possible parameters of the computation (e.g., the value to be added to each item of raw data in case of the kernel ESDM_FUNCTION_SUM_SCALAR, as shown below);
- **buff** points to the memory area allocated for the output;
- **fill_value** points to a possible filling value considered for raw data (otherwise it is set to NULL);
- **valid**, **value**, **value2** and **number** are used for internal purposes; the following pseudo-codes provide further details about the use of these arguments.

Listing 7 and 8 show the C-like pseudo-codes associated with the implementation of a simple *stream_func* and *reduce_func* related to the kernel that evaluates the mean value ("avg") of raw data during data loading.

```
void *esdm_stream_func(
  esdm_dataspace_t * dataspace, void *buff, void *user_ptr, void *)
{
  esdm_stream_data_t *stream_data = user_ptr;
  esdm_stream_data_out_t *tmp = NULL;
  if (stream_data->operation == "avg")
  {
    alloc(tmp);
    tmp->value = 0; // Sum of the items to be processed
    tmp->number = 0; // Number of items to be processed
    for each (item) in (buff) based on (dataspace) // Loop on raw data
    {
       if (item is missing) // based on stream_data->fill_value
         continue;
       tmp->value += item; // Item type is handled by using "dataspace"
       tmp->number++;
    }
  }
  return tmp;
}
```

Shell 7: Pseudo-code associated with an example stream_func

```
void esdm_reduce_func(
  esdm_dataspace_t * dataspace, void *user_ptr, void *stream_func_out)
{
  esdm_stream_data_t *stream_data = user_ptr;
  esdm_stream_data_out_t *tmp = stream_func_out;
  if (stream_data->operation == "avg")
  {
    stream_data->value += tmp->value; // Sum of items to be processed
    stream_data->number += tmp->number; // Number of items to be processed
    *stream_data->buff = stream_data->value / stream_data->number;
  }
  dealloc(tmp);
}
```

Shell 8: Pseudo-code associated with an example reduce_func

The function *esdm_stream_func* in Listing 7 is called once for each ESDM fragment belonging to the dataspace to be loaded from storage. The function allocates a struct tmp to save the intermediate results related to a specific ESDM fragment (line 8). Results consist of the sum of the items of the fragment (line 15) and the number of items (line 16). Missing values (detected by means of the field *fill_value* of struct *stream_data*) are neglected from this evaluation (line 13).

The function *esdm_reduce_func* in Listing 8 is executed after processing all the ESDM fragments associated with the target dataspace and it is called once for each fragment again. The function gathers the intermediate results obtained by *esdm_stream_func* (lines 8 and 9) and fills the field pointed by *buff* of user-defined struct *stream_data* with the result to be returned to the Ophidia operator (line 10). This value is updated at each call with the ratio of the sum of the items to their number, so that the final value corresponds to the correct average value.

4.5.4 List of functions available

Table 1 reports a summary of the implemented kernels[2]. The last column points out the number of items actually loaded: "all" means that the size of the dataset to be loaded is equal to the size of the original dataset (at storage-level), whereas "1" means that raw data reduction will be executed and only the result will actually be loaded (i.e., a single value per block of data will be returned to the application, which in the case of Ophidia refers to a single value per each multi-dimensional binary array constituting the datacube).

Subsetting has also been supported as a type of in-flight operation, however no actual kernel for data subsetting has been developed since the ESDM library already embeds this feature directly into its API. Interestingly, this feature can be used jointly with the application of a kernel to process the subset of selected data.

This subsetting feature can also be exploited by the Ophidia operators. From this point-of-view, the user can select a subset of the original dataset and apply the kernel only to the intended part of the dataset. For instance, the user could select the time series associated with a specific space point (subsetting), evaluate the average of that series by using the operation *ESDM_FUNCTION_AVG* (reduction) and retrieve the mean value only (loading). Of course, the use of arithmetic operators that load "all" the values, not used in conjunction with the subsetting kernel, would not result in any benefits since the whole set of data is streamed from the middleware to the PAV application.

4.5.5 Examples of use

To adopt one of these kernels while converting an ESDM dataset into an Ophidia datacube, the user has to simply set the new argument *operation* of the *OPH_IMPORTESDM* import operator to the related *operation code*, as defined in Table 1.

For instance, the following Opidia task implements subsetting jointly with data reduction by importing the ESDM dataset named *pressure* from the container *esdm://pressure.nc*.

The following code shows the Ophidia commands associated with this example. The dimensions of the dataset are latitude (*lat*), longitude (*lon*) and *time*; the coordinates of the selected point are 40° latitude and 20° longitude (i.e., *subset_filter*); the operation carried out by the ESDM is the average of input data, whose operation code is *avg*. Clearly, the arguments used to define the subset can be set to select a generic sub dataspace, not only a single point by providing the bounding box max-min range.

```
oph_importesdm  input=esdm://pressure.nc;measure=pr;
subset_dims=lat|lon;subset_filter=40|20;subset_type=coord;operation=avg;
```

[2]The code of the compute kernels is available at: https://github.com/OphidiaBigData/esdm-pav-analytical-kernels

Table 1: List of implemented compute kernels for in-flight analytics for the Ophidia PoC

Name	Operation Code	Description	Number of items loaded
ESDM_FUNCTION_MAX	max	Load the maximum value of raw data	1
ESDM_FUNCTION_MIN	min	Load the minimum value of raw data	1
ESDM_FUNCTION_AVG	avg	Load the average of raw data	1
ESDM_FUNCTION_SUM	sum	Load the sum of the elements of raw data	1
ESDM_FUNCTION_STD	std	Load the standard deviation of raw data	1
ESDM_FUNCTION_VAR	var	Load the variance of raw data	1
ESDM_FUNCTION_SUM_SCALAR	sum_scalar	Load raw data and add a scalar to each item	all
ESDM_FUNCTION_MUL_SCALAR	mul_scalar	Load raw data and multiply by a scalar each item	all
ESDM_FUNCTION_ABS	abs	Load the absolute values of raw data	all
ESDM_FUNCTION_SQR	sqr	Load the squares of raw data	all
ESDM_FUNCTION_SQRT	sqrt	Load the square roots of raw data	all
ESDM_FUNCTION_CEIL	ceil	Load the ceiling values of raw data	all
ESDM_FUNCTION_FLOOR	floor	Load the floor values of raw data	all
ESDM_FUNCTION_INT	int	Load the floor values of raw data	all
ESDM_FUNCTION_ROUND	round	Load the round values of raw data	all
ESDM_FUNCTION_NINT	nint	Load the round values of raw data	all
ESDM_FUNCTION_POW	pow	Load the power of each item of raw data to a given exponent	all
ESDM_FUNCTION_EXP	exp	Load the exponential of each item of raw data	all
ESDM_FUNCTION_LOG	log	Load the natural logarithm of each item of raw data	all
ESDM_FUNCTION_LOG10	log10	Load the base-10 logarithm of each item of raw data	all
ESDM_FUNCTION_SIN	sin	Load the sine of each item of raw data	all
ESDM_FUNCTION_COS	cos	Load the cosine of each item of raw data	all
ESDM_FUNCTION_TAN	tan	Load the tangent of each item of raw data	all
ESDM_FUNCTION_ASIN	asin	Load the arcsine of each item of raw data	all
ESDM_FUNCTION_ACOS	acos	Load the arccosine of each item of raw data	all
ESDM_FUNCTION_ATAN	atan	Load the arctangent of each item of raw data	all
ESDM_FUNCTION_SINH	sinh	Load the hyperbolic sine of each item of raw data	all
ESDM_FUNCTION_COSH	cosh	Load the hyperbolic cosine of each item of raw data	all
ESDM_FUNCTION_TANH	tanh	Load the hyperbolic tangent of each item of raw data	all
ESDM_FUNCTION_RECI	reci	Load the reciprocals of raw data	all
ESDM_FUNCTION_NOT	not	Load a binary array where each item is set to 1 if the corresponding item in raw data is equal to 0, otherwise is set to 0	all

This simple example shows how the user can (1) *select* a subset of input dataset from the ESDM (through the arguments *subset_dims*, *subset_filter* and *subset_time*), (2) *apply data reduction* on the stream of selected data (setting the argument *operation*) and (3) load the results as an Ophidia datacube in-memory.

It is important to remark that the original implementation of the import operators only allowed the user to select which data to load, while data reduction could be applied as a subsequent operation after loading the data using the appropriate operators. Thus, this new approach based on in-flight data analytics can provide the benefit of running a data reduction operation on top of the data loading phase directly while the stream of data is being read from the storage system, all with a single operator. To this end, a performance comparison between the original Ophidia version and the extended version with the kernels is currently being carried out to evaluate the impact of the developed solution on the execution time. From this point of view, off-loading computation to the storage back-end (when supported) could represent an additional level of improvement, besides the execution of the kernels on the data stream.

4.5.6 Setup documentation

The analytical kernels can be installed directly from source from the GitHub repository. Assuming that the ESDM library is installed in */path/to/esdm/library*, the kernels library can be installed as follows:

```
$ git clone https://github.com/OphidiaBigData/esdm-pav-analytical-kernels.git
$ cd esdm-pav-analytical-kernels
$ ./bootstrap
$ ./configure --prefix=/path/to/kernels -with-esdm-path=/path/to/esdm/library
$ make
$ make install
```

Shell 9: Instructions for building the analytical kernels for Ophidia from source

This installation procedure will create in */path/to/kernels*, a folder *include/* with the header file *esdm_kernels.h* of the kernels library and a folder *lib/* with the shared library called *libesdm_kernels.so* containing all the kernels functions. In order to use the kernels, this object file has to be linked to the PAV application, such as Ophidia, besides the ESDM library.

For instance, considering the Ophidia framework, the kernel library has been included as part of the source code and can be compiled by simply enabling ESDM support and setting the option "–enable-esdm-pav-kernels" through the framework configuration script. The following configuration options can be added to the default installation options (see Ophidia documentation[3]) of the Ophidia framework and Ophidia I/O server to enable ESDM and kernel support.

```
$ ./configure --with-esdm-path=/path/to/esdm/library --enable-esdm-pav-kernels [...]
```

4.5.7 Preliminary results and next steps

Some preliminary tests of the proposed extensions have been carried out on the Ophidia-ESDM testbed environment setup on the *Zeus supercomputer* at CMCC. The results show that the analytical kernels for in-flight analytics and the related Ophidia extensions can bring some performance benefits to specific operations (e.g., data aggregations) and data structures (e.g., in terms of proper grid size or data size). In particular, although preliminary, the tests show that the application of compute kernels on streams of data loaded through the storage middleware is a promising approach, even simply on a POSIX-based parallel file system like the one used in the tests.

Figure 5 provides early insights into the impact of the in-flight analytical kernels applied within the scope of the Ophidia framework. In this very simple scenario, an average operation is applied over a 8GB dataset

[3]Ophidia installation guides: `https://ophidia.cmcc.it/documentation/admin/install/index.html`

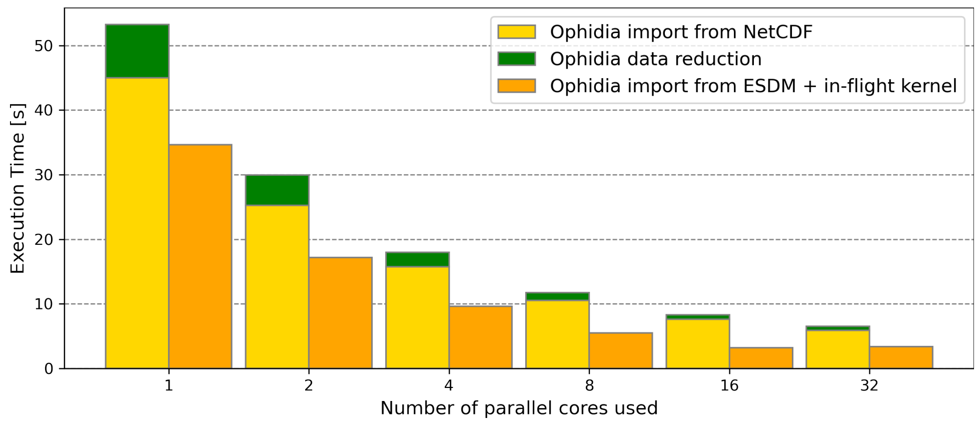


Figure 5: Preliminary results of the comparison between the Ophidia traditional approach for data reduction (from a NetCDF file) and the approach based on in-flight data analysis (via ESDM)

from the ESiWACE2 supported models (i.e., a sample file of the ICON model, see WP1). A number of runs have been executed to measure the execution time of data reduction, by increasing the parallelism exploited by the Ophidia framework from 1 to 32 cores on a single compute node. As it can be observed from the figure, the use of the in-flight extensions during data loading (i.e., via the *OPH_IMPORTESDM* operator shown by the orange bars) was in all cases of the selected scenario faster than the approach currently used by Ophidia, which implies a two-step process consisting of (i) loading data from a NetCDF file first (i.e., *OPH_IMPORTNC2* – yellow bars) and then (ii) applying the operation (i.e., *OPH_REDUCE* – green bars). It is worth mentioning that the application of the analytical kernel while loading data was even faster than the load time of input NetCDF data alone in all configurations.

Of course, further investigation is required, and it will be carried out in the remainder of the project to get a better picture of the potential benefits of the developed in-flight data analysis kernels in Ophidia and, in particular, when these improvements occur and to which extent. To this end, the next steps of this activity will address a more complete evaluation of the performance that shall be presented in deliverable D5.3.

4.6 Visualisation

ParaView is not strictly an analytical kernel but for the sake of completeness of the overview, we briefly report on the current status of our activities regarding ParaView and its integration with ESDM-PAV. For loading the unstructured grids, that are dominant in the high-resolution models, DKRZ has developed the *CDI reader* several years ago. As the storage back-end of the reader uses the netCDF library via CDI, it can be used with ESDM by injecting an ESDM-enabled netCDF library with the `LD_PRELOAD` environment variable. Thus, ParaView can read data directly from ESDM.

DKRZ has extended the CDI-based NetCDF/GRIB reader of Paraview, so it now can handle grids with arbitrary numbers of vertices per cell. Prior to ESiWACE2, the reader was only able to handle triangles. Furthermore, it can now handle files with more-than-one horizontal grid present in the file. This makes it a lot more versatile in directly loading model output, and avoids data-intensive splitting of files that was previously necessary. These new features allow the use of this reader with all ESiWACE and DYAMOND models, and even with the FESOM model (e.g. H2020 project nextGEMS), which has spatially varying vertex counts. The new version of the reader further is more flexible in the handling of missing values, allowing ocean model output to be represented better, as the bathymetry can be handled correctly with minimal effort.

For in-situ visualization, DKRZ has decided to couple ParaView to ICON via its model coupler YAC. This has several major benefits, as the coupler is inherently built for connecting different parts of the model-universe. By connecting to the coupler, and not the model itself, we have a clean layer of separation between the in-situ layer and the model code, so we are only minimally affected (and affecting) future model development.

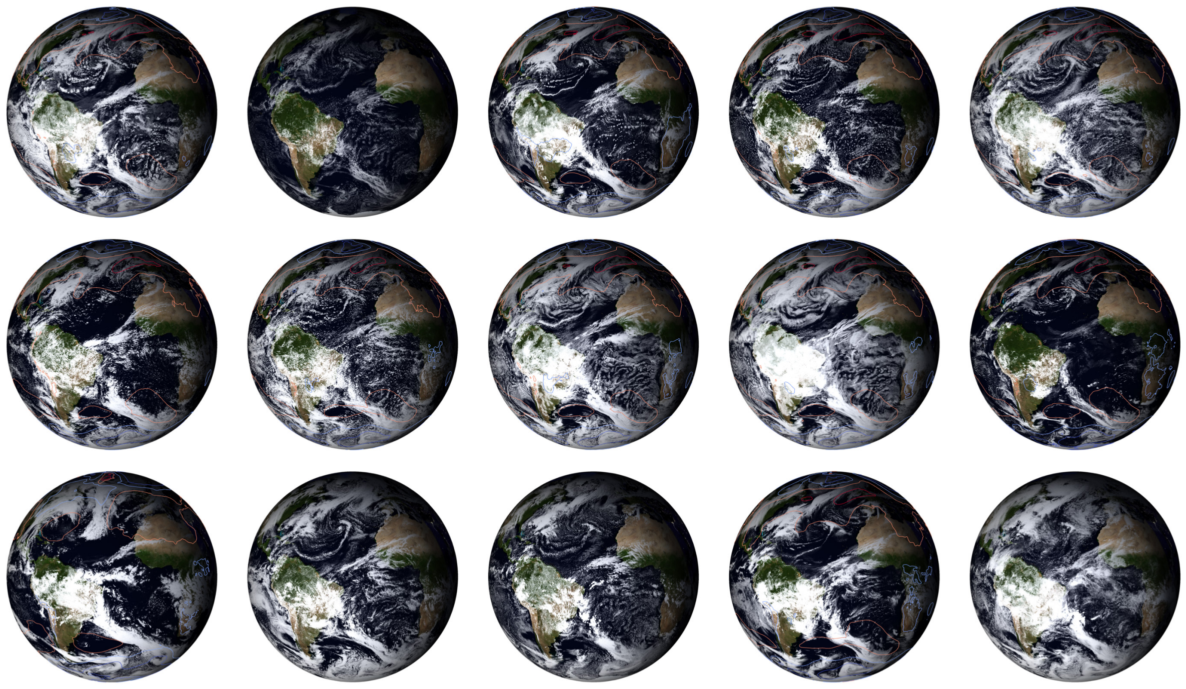


Figure 6: Rendering of the DYAMOND model intercomparison models two days into the simulation. This rendering was performed with the model outputs standardized by ESiWACE WP1 using one ParaView batch script. The rendering shows a composite of cloud ice and water with sea level pressure as overlay if available. Not all models have delivered sea level pressure data. Creating the same rendering without the work performed in ESiWACE would have required remapping to a Paraview-compatible grid and thus additional pre-processing and a loss of information (or massive over-sampling).

Furthermore YAC can handle remaps, sub-setting the model domain and time-aggregation very efficiently, so we only have to pass the data to ParaView that is actually needed for the visualization of interest, and - for example - do not need to handle the *back side* of the world in a spherical projection, thus reducing the computational cost. YAC is currently getting an OASIS compatibility layer, so it would be possible to use the capabilities developed here in models that use the widespread OASIS coupler by simply replacing OASIS with YAC.

4.7 Summary

A set of analytical kernels, making use of GPUs and streaming I/O APIs, have been implemented with the goal of improving PAV applications and integrated with CDO and Ophidia. The implementation and initial testing activities described in this document allowed us to understand the effectiveness of the proposed approaches and to identify potential issues to keep in mind when designing novel core routines for PAV.

The use of GPUs for analytical kernels can provide benefits to specific operations, even though in many cases it resulted only in minor performance improvements with the data sets and machines available today, but still it provides potential for future applications. We could also find that, without the presence of a at least semi modern software stack, especially compilers, working with GPUs requires huge amount of extra steps and is in general unnecessarily complex. Furthermore, we identified possible future problems in terms of distribution of the GPU accelerated routines, since these would need to be recompiled for each system depending on the present hardware and software.

The streaming I/O interface provided by ESDM allows the application of analytical functions while moving data from the storage to the compute nodes. This can improve the overall data analysis process. However, its impact depends on the type of operation implemented and is more significant in case of data aggregations (e.g., average, maximum, minimum, etc.). Moreover, the storage middleware, as in the case of ESDM, could potentially off-load part of the computation to the storage back-end (if this is supported) reducing the data moved between systems and further improving the execution.

4.8 Next Steps

A set of activities have already been identified to further improve the support for the analytical kernels here presented. The next steps related to the different developments presented in this deliverable that will be carried out in the remained of the project are outlined in the following paragraphs.

Ophidia Continuing the preliminary evaluation started at the time of this document writing, a set of different tests applied to the developed in-flight analytical kernels based on the ESDM streaming API will be performed. Tests will evaluate the execution time of different operations by using the original Ophidia implementation versus the one extended with the in-flight kernels on different data distribution scenarios. The aims of this benchmark is to provide a more formal understanding of the actual benefits and limitations of the in-flight analytical kernels.
Moreover, we also plan to use and evaluate these kernels in the context of more structured experiments composed of different tasks and handled via the ESDM-PAV runtime system.
Both the results from the performance evaluation tests and the structured workflows will be presented in deliverable D5.3.

CDO In the following deliverable (D5.3) we intend to present new data from multiple HPC clusters for the GPU accelerated version of the CDOs. further we plan to deliver, independent of the results, more concrete data about the target systems, more detail about our scripts as well as our routines, and the methods of running our benchmarks. This should facilitate reproducibility or at least enough information to see where performance differences on the different systems stem from. The intended targets for the benchmarks are: GPU equipped Workstations at the MPI-M, Levante, and LUMI.
With those benchmarks we plan to make concrete statements about the feasibility and usefulness of introducing GPU acceleration to certain CDO routines. Should the results be unsatisfactory, we will provide pointers to the bottlenecks in the CDOs which make GPU acceleration unattractive for the version then recent, or, if that is the case, describe why the workflows or the CDO architecture are unsuited for this undertaking.

5 References

Eugen Bethke, Bryan Lawrence, Julian Kunkel, Luciana Pedro, Nathanael Hübbe, Neil Massey, Jeff Cole, Grenville Lister, Simon Wilson, Donatello Elia, and Cosimo Palazzo. D4.1 advanced software stack for earth system data, March 2021. URL https://doi.org/10.5281/zenodo.4651493.

Donatello Elia, Sandro Fiore, and Giovanni Aloisio. Towards hpc and big data analytics convergence: Design and experimental evaluation of a hpda framework for escience at scale. *IEEE Access*, 9:73307–73326, 2021a. doi: 10.1109/ACCESS.2021.3079139.

Donatello Elia, Cosimo Palazzo, Manuel Musio, Reinhard Budich, Oliver Heidmann, Dela Spickermann, and Julian Kunkel. Deliverable D5.1 Report on the ESDM runtime extensions for parallel in-flight analytics, December 2021b. URL https://doi.org/10.5281/zenodo.5799437.

S. Fiore, A. D'Anca, C. Palazzo, I. Foster, D.N. Williams, and G. Aloisio. Ophidia: Toward big data analytics for escience. *Procedia Computer Science*, 18:2376–2385, 2013. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2013.05.409. URL https://www.sciencedirect.com/science/article/pii/S1877050913005528. 2013 International Conference on Computational Science.

Sandro Fiore, Donatello Elia, Cosimo Palazzo, Alessandro D'Anca, Jean-Thomas Acquaviva, Niklas Röber, Dela Spickermann, Florian Ziemen, Oliver Heidmann, Hua Huang, Julian Kunkel, and Luciana Pedro. Architecture milestone ms5.1, December 2019. URL https://doi.org/10.5281/zenodo.3592385.

Julian Kunkel, Luciana Pedro, Bryan Lawrence, Nathanael Hübbe, Glenn Greed, David Matthews, Donatello Elia, Sandro Fiore, Neil Massey, Hua Huang, and Jean-Thomas Acquaviva. Esdm architecture milestone ms4.1, March 2020. URL https://doi.org/10.5281/zenodo.3724217.

Jakob Luettgau, Julian Kunkel, Bryan Lawrence, Jens Jensen, GIuseppe Congiu, Huang Hua, and Paola Nassisi. New storage layout for earth system data (d4.2), July 2017. URL https://doi.org/10.5281/zenodo.2573896.

Luciana Pedro. Final Implementation of the Earth System Data Middleware (D4.3), June 2019. URL https://doi.org/10.5281/zenodo.3361225.

6 Changes made and/or difficulties encountered, if any

There have been technical difficulties on DKRZ's Mistral system - coming of age in 2022 - which made it difficult to get the tests running with the gpu - versions of the CDOs. These, in conjunction with Covid19-related absences, made it impossible to get an as complete as desirable picture of possible benchmarks. Also, these absences caused a delay of handing in the deliverable by a month.

7 How this deliverable contributes to the European strategies for HPC

Pre-exascale (and in the near future exascale) supercomputers being setup in Europe consist of very heterogenous computing and storage infrastructures. Being able to exploit these architectures is a key aspect to support post-processing, analytics and visualisation (PAV) on the large volumes of output produced by the currently developed highest-resolution ESM simulations.

T5.3 of ESiWACE2 WP5, of which this deliverable describes the main results, tries to go into this direction by exploring the use accelerators (e.g., GPUs) and alternative I/O APIs (e.g., for streaming processing) for the development of enhanced kernels for PAV applications.

8 Sustainability

The developments described in this document are linked to deliverable *D5.1* Elia et al. (2021b), which reports the ESDM-PAV and the approach followed for the in-flight analytics, as well as to *D4.1* Bethke et al. (2021) from WP4, documenting the ESDM software. Moreover, the activities spawn from the requirements initially described in the project milestones *MS5.1* Fiore et al. (2019).

The extensions to support in-flight processing on data streams have been integrated in the Ophidia framework and made available to users as open source code. Although the developments represent mainly a proof of concept, the approach showed to be promising and the experience gained for the integration of the streaming APIs from ESDM could be reused in the future to target other formats and storage libraries.

The Paraview CDI reader is part of the official Paraview release and a part of DKRZs standard portfolio. The Paraview documentation created for the ESiWACE Visualization courses was merged into DKRZ's official documentation and will continue to be maintained as such. All developments on the CDI reader are independent of the I/O library used in the background, and thus can also be used with commonly used storage options such as netCDF and GRIB, and as CDI is getting FDB/MARS support as part of the H2020 Project ACROSS, and the upcoming BMBF project WarmWorld, we are confident that we will also be able to connect to ECMWF's FDB/MARS Database system, that will most likely form the backbone of the data handling in ESiWACE3, as well.

9 Dissemination, Engagement and Uptake of Results

9.1 Target audience

As indicated in the Description of the Action, the audience for this deliverable is:

X	The general public (PU)
	The project partners, including the Commission services (PP)
	A group specified by the consortium, including the Commission services (RE)
	This reports is confidential, only for members of the consortium, including the Commission services (CO)

This deliverable will be publicly made available on the ESiWACE Zenodo repository. Results will be presented during community-based conferences.

9.2 Record of dissemination/engagement activities linked to this deliverable

See table 2.

9.3 Publications in preparation OR submitted

N/A

9.4 Intellectual property rights resulting from this deliverable

N/A

Table 2: Record of dissemination / engagement activities linked to this deliverable

Type of dissemination and communication activities	Details	Date and location of the event	Type of audience	Zenodo Link	Estimated number of persons reached
Participation in a workshop	Donatello Elia, "A HPDA- enabled environment for scalable climate data analysis", 6th ENES HPC Workshop	29 May 2020, Virtual/ Hamburg (DE)	Scientific Community	`http://doi.org/10.5281/zenodo.3964536`	80
Participation in a workshop	Donatello Elia, "An HPC- enabled Data Science and Learning Environment for Climate Change Experiments at Scale", EXDCI-2 Virtual Workshop on AI and HPC convergence	26 November 2020, Virtual	Scientific Community	`https://doi.org/10.5281/zenodo.4633367`	45
Participation in a workshop	Donatello Elia, "Ophidia: High Performance Data Analytics framework for Climate Change at scale", IS-ENES3 Virtual workshop on requirements for a fast and scalable evaluation workflow	18-19 May 2021, Virtual	Scientific Community	`https://doi.org/10.5281/zenodo.5535704`	30
Participation in a workshop	Donatello Elia, "Towards HPC and Big Data convergence for climate analysis at scale", 7th ENES HPC Workshop	9-11 May 2022, Barcelona (ES)	Scientific Community	`https://portal.enes.org/community/community-workshops/7th-enes-workshop-presentations/06_Elia_HPDA_Ophidia.pdf`	80