Evaluation of Atlas 0.21 interpolation capability

A. Piacentini

TR/CGCM/20/105

Table of Contents

Abstract

Atlas is an open source library currently developed at ECMWF providing grids, mesh generation, and parallel data structures targeting Numerical Weather Prediction or Climate Model developments.
It is designed as an Object Oriented modular library, with the capability to take advantage of the most recent computer architectures.
It is meant to provide, among many other features, a set of parallel interpolation methods for the conversion between different distributed representations of discrete physical fields.
We evaluate the possibility of interfacing the Atlas interpolation functions from within the OASIS3-MCT coupler in order to improve the quality and the efficiency of the current SCRIP based interpolations.

1.

1. Rationale

Currently, OASIS3-MCT includes a hybrid MPI+OpenMP parallel version of the SCRIP library (previously fully sequential) leading to great reduction in the off-line calculation time of the remapping weights. The results obtained with this SCRIP parallel version show a reduction in the remapping weight calculation time of 2 or 3 orders of magnitude as compared with the sequential version for high-resolution grids. However, the SCRIP library presents some specific issues near the poles. As noted in the SCRIP User Guide, the SCRIP search and intersection algorithms are based on linear parameterizations which are not valid near the pole; to avoid these problems, a Lambert equivalent azimuthal projection can be used poleward of a given threshold latitude but this option brings

in other issues.
Furthermore, we have to anticipate potential bottlenecks when foreseeing the increasing resolution of new model configurations, the new architectures of the future generations of supercomputers and, finally, the need of on-line run-time updates of the interpolation weights for models using adaptive or moving meshes.
A literature review of other libraries is currently carried on, in order to identify which ones could possibly replace the SCRIP in OASIS3-MCT or be available with the OASIS3-MCT sources in a specific environment so to allow an efficient and high-quality off-line generation of the remapping weights before the run. Candidates include ESMF, XIOS, MOAB/TemestRemap, ATLAS, YAC/CDO and CWIPI. A detailed analysis of the quality and performance of candidate libraries is systematically performed and they are compared to the SCRIP.
This document refers to the ATLAS library, developed at ECMWF, in its official version 0.21.0 available at the time of writing (Aug. 2020).

2. The Atlas library

ECMWF is developing a library called Atlas, with the primary goals to exploit the emerging hardware architectures becoming available in the next few decades, and to support the development of alternative numerical algorithm strategies in operational Numerical Weather Prediction.
Atlas is also expected to facilitate the coupling of an increasing number of Earth system components, such as the atmosphere, ocean, wave, surface, or sea-ice, and could effectively enhance existing couplers.
Nevertheless, when compared to ESMF, it has to be noticed that ESMF and Atlas both provide similar fundamental building blocks for data structures and model development, however, Atlas has the distinct primary goal of accelerating novel numerical algorithm development for emerging hardware architectures, compared to ESMF's effort to enhance collaborative Earth system model development. This has an impact on the way the API's are designed: if ESMF is oriented to the description of externally defined grids, Atlas is oriented toward the use of an internally consistent set of predefined grids and meshes.
We recall three of the stated goals of Atlas in the defining paper Deconinck et al., *Atlas : A library for numerical weather prediction and climate modelling*, Computer Physics Communications (2017), 188-204, 220 that could be relevant for its use in OASIS:

- Facilitate the implementation of different structured and unstructured point distributions on the sphere (global grids) and on limited areas of the sphere (non-global grids)
- Support different programming languages, including Fortran and C++, providing object-oriented (OO) designs and data structure flexibility, so that Atlas can be used to update existing and support new code infrastructures.
- Provide object-oriented programming interfaces enhancing multi-disciplinary collaboration at multiple levels ranging from e.g. high-level mathematical operators, typically developed by domain scientists, to low-level data-storage abstractions, typically maintained by computer scientists.

The development of the Atlas library is part of the wider 'Scalability Programme' ongoing at ECMWF and is still an active task with frequent updates. Public versions of the Atlas library have been delivered as part of the ESCAPE initiative and are available on github.

Atlas relies on the ECMWF set of cmake macros *ecbuild* for its compilation and on the cross-platform C++ toolkit *eckit* and on the fortran toolkit *fckit* for interoperating Fortran with C/C++ and interfacing

eckit. The three packages are freely available on github.

3. The Atlas conceptual model

The key concepts in the design of the Atlas data structure are:

- *Grid*: ordered list of points (coordinates) without connectivity rules;
- *Mesh*: collection of elements linking the grid points by specific connectivity rules;
- *Field*: array of discrete values representing a given quantity;
- *FunctionSpace*: discretisation space in which a field is defined.

These concepts are depicted in Fig. 1.

A *grid* is merely a predefined list of two-dimensional points, typically structured and using two indices *i* and *j* so that point coordinates and computational stencils (e.g. derivatives) are easily retrieved without connectivity rules. In many cases a grid is enough to define fields with appropriate indexing mechanisms.
For element-based numerical methods (generally unstructured) however, the mesh concept is introduced that describes connectivity lists linking elements, edges and nodes.
A *mesh* may be decomposed in partitions and distributed among MPI tasks. Every MPI task then allows computations on one such partition. Overlap regions (or halos) between partitions can be constructed to enable stencil operations in a parallel context.
In addition to a grid and mesh, it is necessary to introduce the concept of *field*, intended as a container of discrete values of a given variable.
A field can be discretised in various ways. The concept responsible for interpreting or providing the discretisation of a field in terms of spatial projection (e.g. grid-points, mesh-nodes, mesh-cell-centres) or spectral coefficients is the *function space*. The function space also implements parallel communication operations responsible for performing synchronisation of fields across overlap regions.

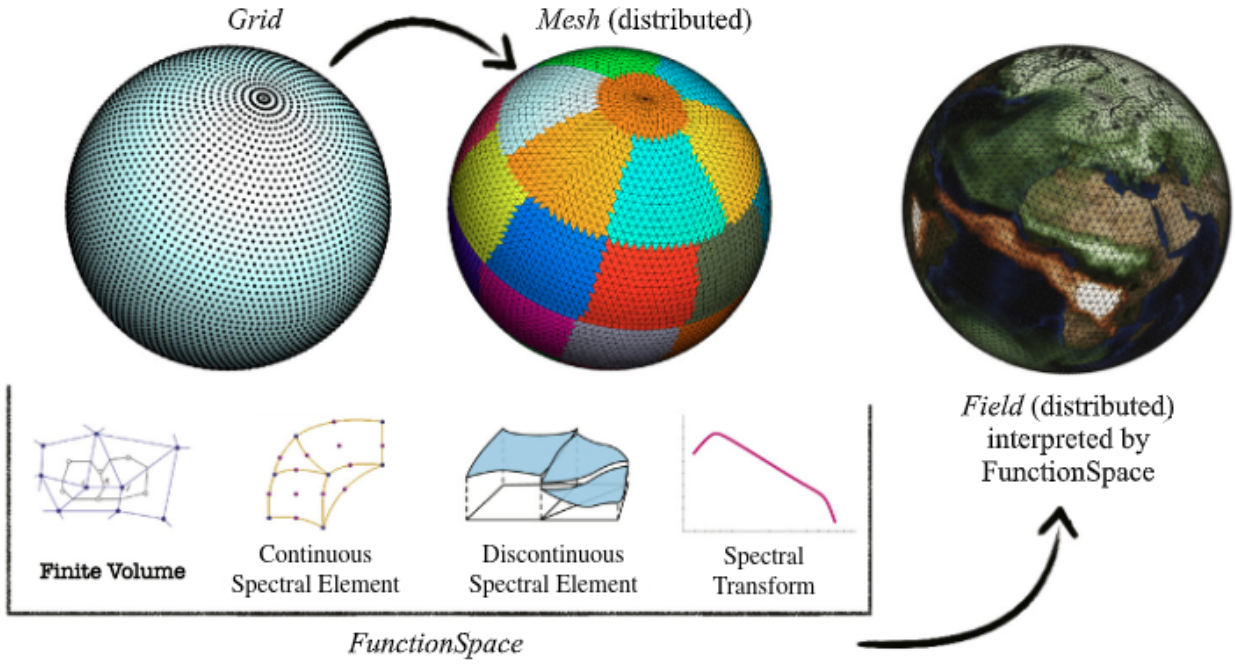

Figure 1: The conceptual design of Atlas

A possible Atlas workflow consisting of the creation and discretisation of a field, is illustrated in Fig. 2

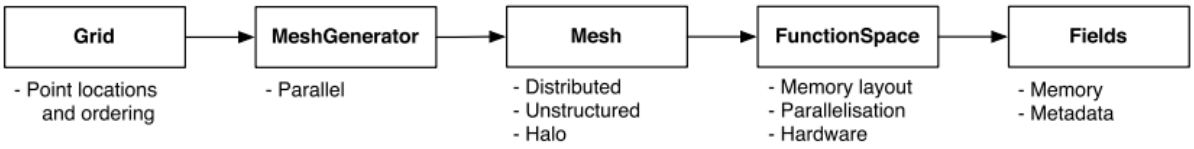

Figure 2: Workflow of Atlas starting from Grid to the creation of a Field, discretised on a Mesh and managed by a FunctionSpace

More in detail, the grids within Atlas are classified hierarchically from a completely unstructured to a fully structured interpretation. One of Atlas' functions is to provide a catalogue of a variety of grids defined by the World Meteorological Organisation. A non-exhaustive classification of grids which drove the design of Atlas is given in Fig. 3. With every step in the classification, more structure is present in the grid that can be exploited. Not all of them have been implemented so far, in particular the grids under the *StructuredZonesGrid* class.

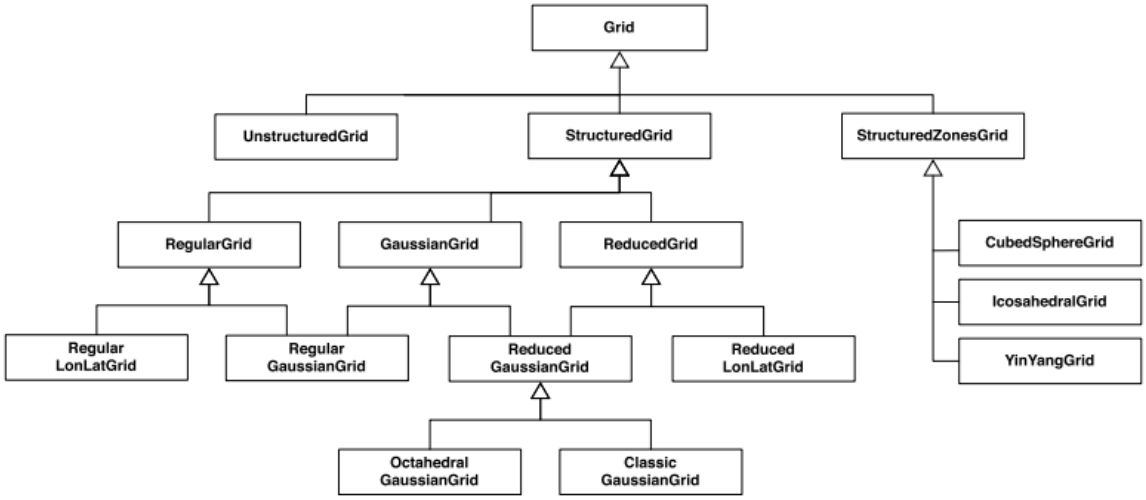

Figure 3: Classification of grid classes.

In this classification the Grid base class presents a generic unstructured view of all the points present in the grid, single indexed. Note that the grid has no knowledge of any domain decomposition or parallelisation strategy.

More details on the Grid classes are provided in the cited reference paper or on the documentation site (under construction and loosely following the current implementation) https://sites.ecmwf.int/docs/atlas/design/grid/

For a wide variety of numerical algorithms, a Grid (i.e. a mere ordering of points and their location) is not sufficient and a *Mesh* might be required. This is usually obtained by connecting grid points using polygonal elements also referred to as cells.
For regular grids and some other structured grids, the mesh element can be inferred, for the other grids, notably the unstructured grids, explicit connectivity rules are required. The *Mesh* class, combining information from the lower level *Nodes*, *Cells*, *Edges* classes, provides a mean to access connectivity and adjacency relations. The *Mesh* can be distributed in memory and includes ghost points for the handling of domain decomposition halos.
A Mesh may simply be read from file by a MeshReader, or generated from a Grid by a MeshGenerator. The latter option is illustrated in Fig. 2, where the grid points will become the nodes of the mesh elements.

The *FunctionSpace* class is introduced because a Field can be discretised on the computational domain in various ways: e.g. on a grid, on mesh-nodes, mesh-cell-centres or spectral coefficients. The representation of a given

variable is intimately related to the adopted spatial numerical discretisation strategy (e.g. finite volume, spectral element, etc.).
In addition to interpreting how a Field is discretised, the FunctionSpace also manages how the Field is parallelised and laid out in memory. Concrete FunctionSpace classes may implement parallel operations such as gather and scatter, reduce-all, or point-to-point communications, thus enabling the practical use of fields within parallel numerical algorithms

The currently implemented FunctionSpace classes include *NodeColumns*, *EdgeColumns*, *StructuredColumns* and *Spectral*.
In particular, the *NodeColumns* class describes the discretisation of generic fields with values located at the nodes of a mesh (notice that a Mesh object is needed) and may have multiple layers defined in the vertical direction.
The *StructuredColumns* class describes the subset of fields with values located at the points of a structured grid and does not need the definition of a Mesh object.

Finally, the *Field* class contains the values of a full scalar, vector or tensor field. The Field values are stored contiguously in memory, and moreover they can be mapped to an arbitrary indexing mechanism to target a specific memory layout. The ability to adapt the memory layout to match, for instance, the most efficient data access patterns of a specific hardware is a key feature of Atlas, even if it is hard to see how to take advantage of this capability in the context of the coupling of legacy codes.
A Field also contains Metadata which store simple information like a name, units, or other relevant information.
A Field delegates the access and storage of the actual memory to an Array that accommodates memory storage on heterogeneous hardware (e.g. the Array is responsible to synchronise data across the device - as a GPU - and the host - as a CPU).
Parallelisation related methods are delegated to the associated FunctionSpace: each concrete FunctionSpace may implement methods like haloExchange, gather, scatter, or choose to delegate, in turns, its implementation to one or more parallelisation primitives like HaloExchange and GatherScatter, which are then setup for the required memory layout.
Logically related Fields can be grouped together into one or more *FieldSet*s and accessed from the FieldSet by name or by index.

Many NWP and climate models contain algorithms to perform a variety of *mathematical operations* on fields. These operations relate closely to certain spatial discretisations or FunctionSpaces. Atlas provides implementations for some of these operations given a field that is compatible with the related FunctionSpace. One relevant example is the *fvm::Method* class, which contains everything required to construct mathematical operators using an edge-based finite volume scheme, from which the concrete *fvm::Nabla* class has been used for the implementation of the new non-hydrostatic finite volume dynamical core for IFS.

For our analysis, we are interested in the *interpolation* and *remapping* capabilities which have been introduced in Atlas because of the common need of dealing simultaneously with different representations of the same data (grid points vs. spectral coefficients, different grids, increasing or decreasing resolution, rotating grids, etc.).
Due to its design, Atlas provides a management of fields abstracted from the lower-level details: data locality, global reduce operations (both order-independent and not) and other parallelism concerns. In this way, it provides a solid and flexible foundation to build interpolation methods compatible with a variety of field representations

4. Currently implemented interpolations in Atlas

As in version 0.21.0, Atlas is now capable of some interpolation methods and the choice is limited by the grid class (cf. Fig. 3 for the taxonomy of grids) and the functionspace associated to the source and target representations.

Citing Deconink (personal communication, July 2020), what is currently supported is:

- *Structured source grid to any target grid.*
  - *the source grid can be partitioned with e.g. the equal_regions partitioner.*
  - *the target grid must currently follow the distribution of the source grid (MatchingPartitioner)*
  - *structured interpolation methods in 2D (horizontal) and 3D (horizontal+vertical): linear, cubic, quasicubic*
  - *unstructured interpolation methods, in 2D only: finite-element (requires meshing), k-nearest-neighbour (based on kD-tree)*

- *Unstructured source grid to any target grid.*
  - *source grid can be partitioned with e.g. the equal_regions partitioner, or custom, …*
  - *target grid must currently follow the distribution of the source grid (MatchingPartitioner)*
  - *meshing of source grid required, currently not parallelised.*
  - *interpolation methods: finite-element, k-nearest-neighbour*

What is currently not supported, but under development:

- *missing values*
- *non-matching domain decomposition*
- *native support for the NEMO ORCA grids so that it can be meshed and domain-decomposed quickly, and its structure can be exploited for interpolation.*
- *grid-box-average interpolation*
- *second-order conservative interpolation as in Jones 1998*

Trying to relate this information with the current workflow in OASIS, we need to match the coordinates based description of the grids in OASIS with the grid catalogue from Atlas.
In OASIS, all the grids are entered via the lon, lat coordinates of the cell centres and, optionally, if conservative interpolations are needed, the corners coordinates of each cell. The value of the field can be seen as located at the grid centre (and as such it is used for distance based interpolations as nearest neighbour[s] or bilinear and bicubic) or assigned as constant to the cell described by the corners (and as such it is used for intersection based interpolations as the first or second order conservative methods). For the bilinear and bicubic interpolations some neighbouring information is inferred from the indexing of cartesian or reduced grids, but an explicit mesh connectivity is never needed.
When entering in Atlas a grid by its coordinates, with no other information, it is necessarily entered as an unstructured grid. Notice that for the description of Regular LonLat grids or Structured Reduced Gaussian grids Atlas does not require the point coordinates but only the parameters (number of subdivisions or Gaussian order) for the internal reconstruction of the grid point coordinates.
In the case of unstructured grids, the interpolation methods are therefore restrained to nearest neighbour(s) or finite element approximations on a mesh which has been computed by Delaunay triangulation with no detection of logically rectangular elements. Notice, moreover, that the Delaunay triangulation is currently delegated to the CGAL package, it is not parallelised and its complexity is close to linear in the number of points for points on a sphere surface, but could increase up to quadratic in the number of points for complex non-coplanar geometries: (cf.
https://doc.cgal.org/latest/Triangulation_3/index.html#Triangulation3seccomplexity).
Finally, the current limitation on the Matching Partitions (grossly stated as “distributing the target grid with the

same geographic decomposition of the source grid, so that all the source information for a target cell is already local to its resources") is in contrast with the usage in a coupler that has to accommodate pre-existing model decompositions and resources assignments.

5. Test cases from the OASIS benchmark

In the OASIS3-MCT distribution a predefined set of 2D surface grids is used for the non regression tests and the interpolation quality tests in general.
Grids in the set are uniquely identified by a four letter key. For every grid the cell centres and cell corners coordinates are provided, together with a binary land/sea mask for surface points (and for advanced conservative interpolations a fractional mask for every pair of grids, accounting for non coincident coast representations) and an auxiliary cell areas field for the consistent surface weighting of the cells when computing discrete integral quantities.

The grids are:

- `bggd`: a regular LonLat grid with 144x143 subdivisions
- `ssea`: a reduced Gaussian Grid with 128 latitudes and a varying number of longitudes per latitude, not corresponding to any of the predefined reduction strategies in Atlas
- `torc`: a stretched logically cartesian grid corresponding to the ORCA2 mesh from the NEMO ocean model
- `nogt`: a stretched tripolar logically cartesian grid corresponding to the ORCA1 mesh from the NEMO ocean model
- `icos`: an icosahedral grid from the DYNAMICO core of the LMDz model with 15212 cells
- `icoh`: a high resolution icosahedral grid from the DYNAMICO core of the LMDz model with 2016012 cells

Notice that the `bggd` grid can be described in Atlas also as a structured grid with key `L144x143` and that the `ssea` grid can be described also as a structured *atlas_ReducedGaussianGrid*, even if there is no Atlas key for this reduction strategy, passing an integer array of size 128 (the number of latitude subdivisions) providing the number of longitude subdivisions per latitude.

Since the ORCA and the icosahedral grids are not yet natively supported by Atlas, all the other grids can only be described as unstructured and a Delaunay triangulation is carried on to reconstruct a Mesh object. This practice overrides any intrinsic structure of the logically cartesian ORCA grids and of the analytically described icosahedra. Moreover, for the ORCA grids the overlapping points of the north fold have to be manually removed before entering the coordinates as grid points since Atlas does not handle repeated coordinates.

Since OASIS3-MCT is coded in Fortran, we have implemented a flexible Fortran test program using the Atlas F2003 APIs. We have noticed that only a subset of the C++ APIs have been ported to Fortran, even if the Atlas coding design allows for systematically exporting the C++ APIs to extern C functions that can, in turn, be accessed by Fortran iso c bindings.
We have used an *fckit_configuration* object for parsing a *json* configuration file that we use to select a pair of grids from the OASIS test set (notice that `bggd` and `ssea` have been doubled to `Atlas_bggd` and `Atlas_ssea` for their corresponding structured descriptions) and perform a ping-pong interpolation of an analytical smooth function, optionally modified with a strong gradient structure mimicking the Gulf Stream.
The different Atlas methods can be tested on a single processor or in parallel (Atlas imposes the domain decomposition with an Equal Area partitioner on the source grid and with the corresponding Matching Partitioner – when possible – for the target grid), accordingly to the supported (structured or unstructured) grid representations.

The analytical and the interpolated fields as well as their relative percentage misfits are optionally stored in NetCDF files and – if a Mesh is generated in Atlas – in gmsh files for 3D visualization on the sphere.

An an example, Fig. 4 shows the relative percent error of a k-nearest-neighbours (k=4) interpolation from the `icos` to the `torc` grid. The Delaunay triangular mesh is overlapped in black. The stretched ORCA2 grid points moved from the continents to the Mediterranean, Red, Black and Caspian Seas seas, in order to locally increase the resolution, are clearly visible.

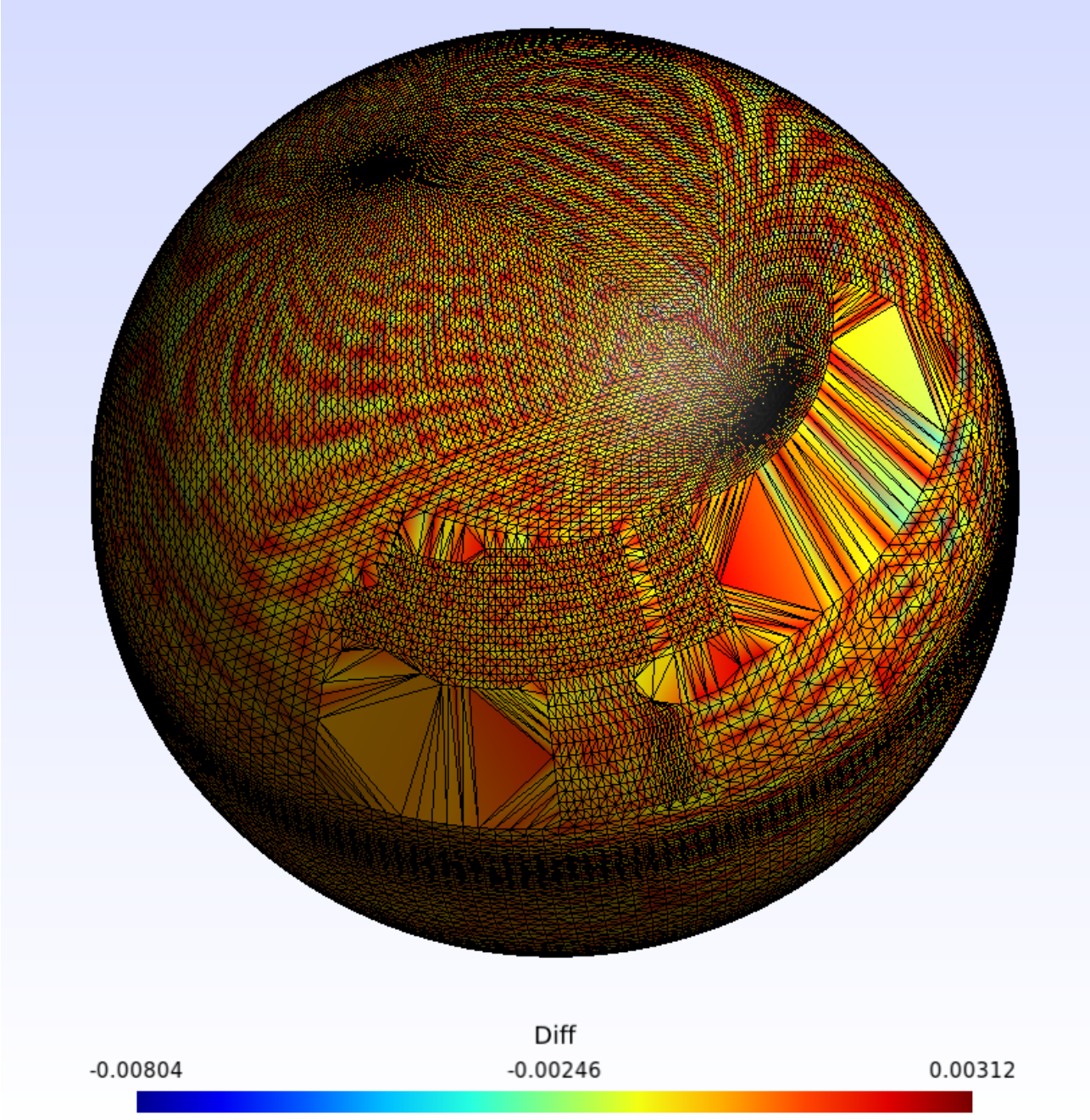

Figure 4: Relative difference for the k-nearest-neighbours (k=4) interpolation from an icosahedral to the ORCA2 grid (both described as unstructured)

Conversely, Fig. 5 shows the difference of the single nearest-neighbour interpolation from the ORCA2 `torc` grid to the low resolution `icos` grid: the global field on `icos` is reconstructed from the shifted continent points, leading to higher errors on the continents and is extrapolated to the Antarctic region from the southernmost values on the ocean grid, not covering the South Pole.

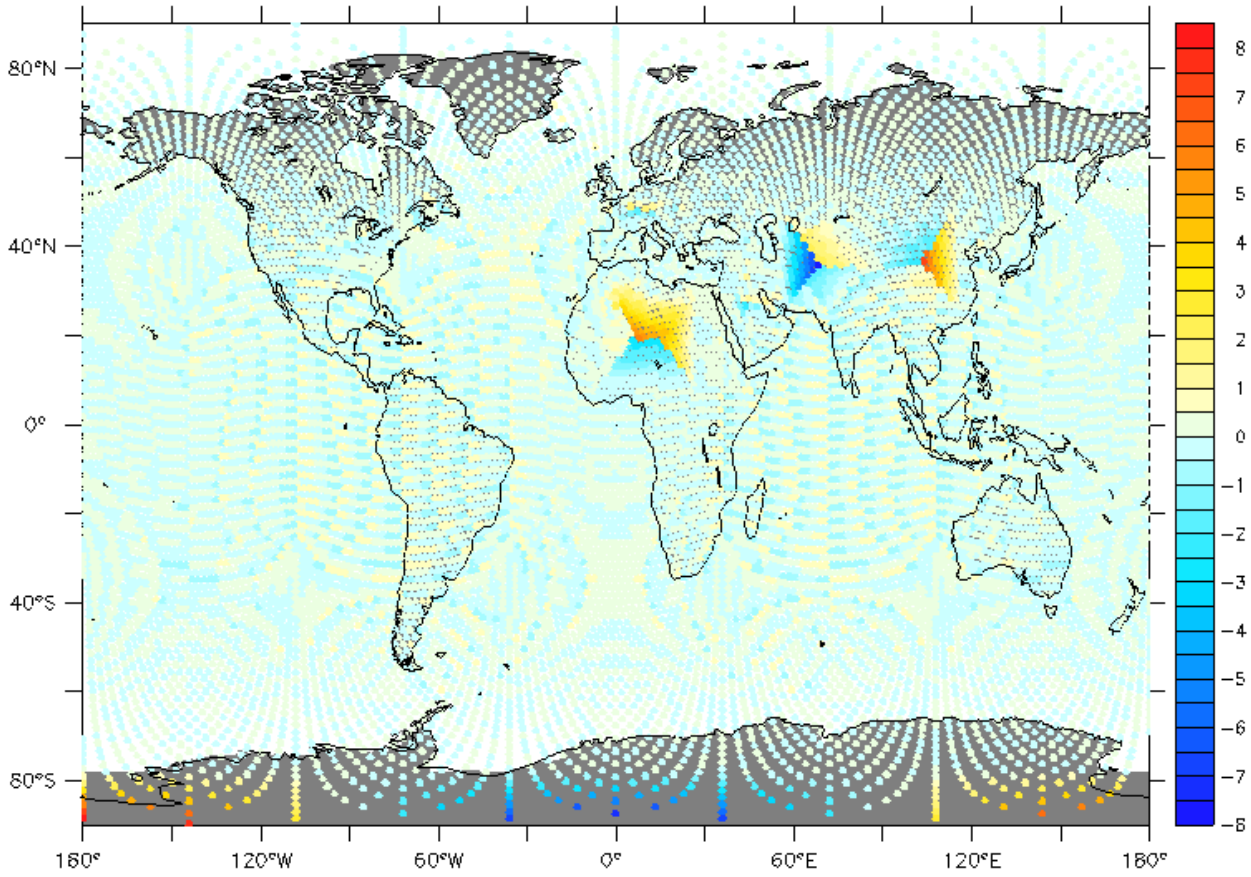

Figure 5: Relative difference for the nearest-neighbour interpolation from the ORCA2 to an icosahedral grid (both described as unstructured)

Fig. 6 shows the error in case of icosahedral grid refinement from `icos` to `icoh` with a nearest neighbour interpolation

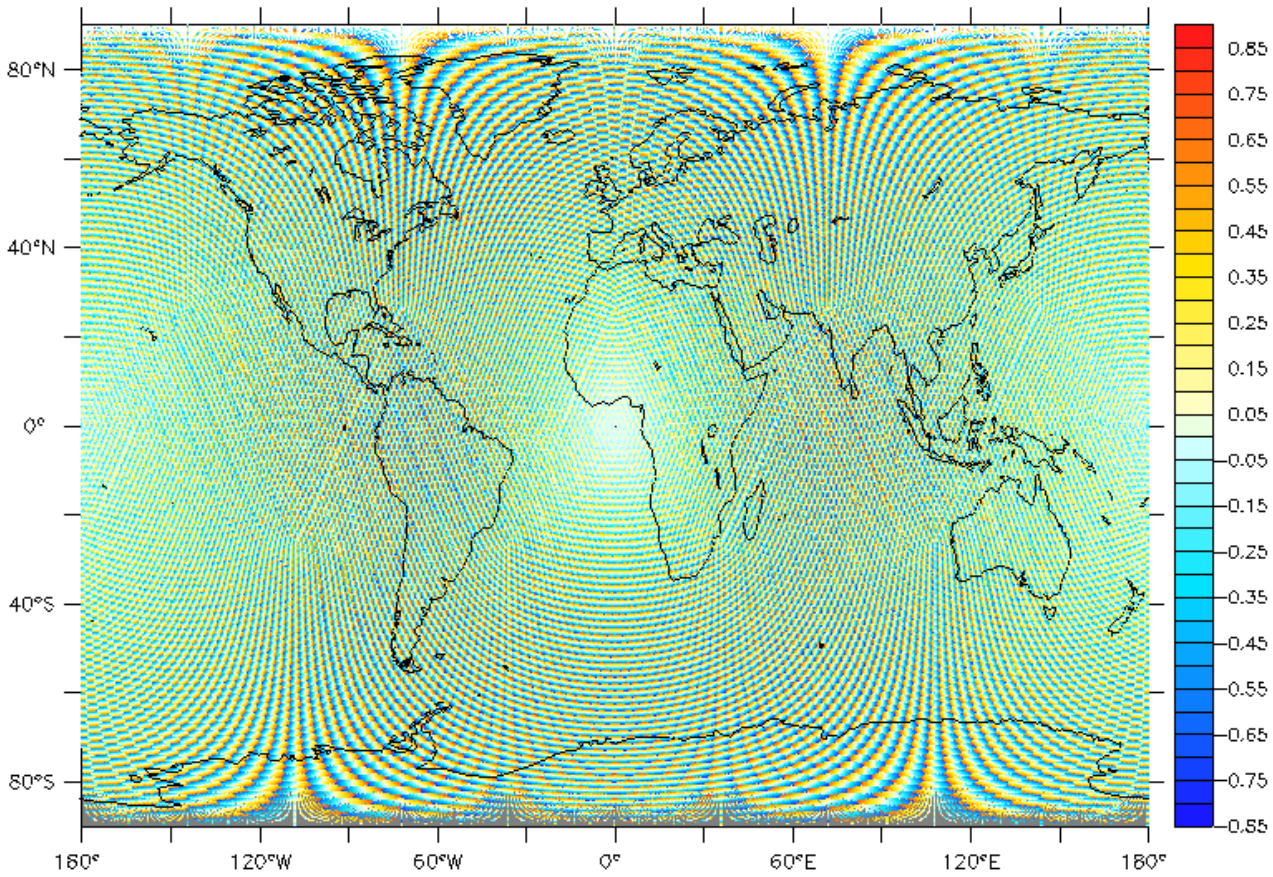

Figure 6: Relative difference for the nearest-neighbour interpolation from the low resolution to the high resolution icosahedral grid (both described as unstructured)

While Fig. 7 refers to the same interpolation but with the finite element method. Notice the concentration of errors near to the coordinates axes intersection, here at -180°,0° (the colorbar has been bounded to avoid hiding the remaining structures), a feature that is seen also with the structured bilinear and bicubic methods.

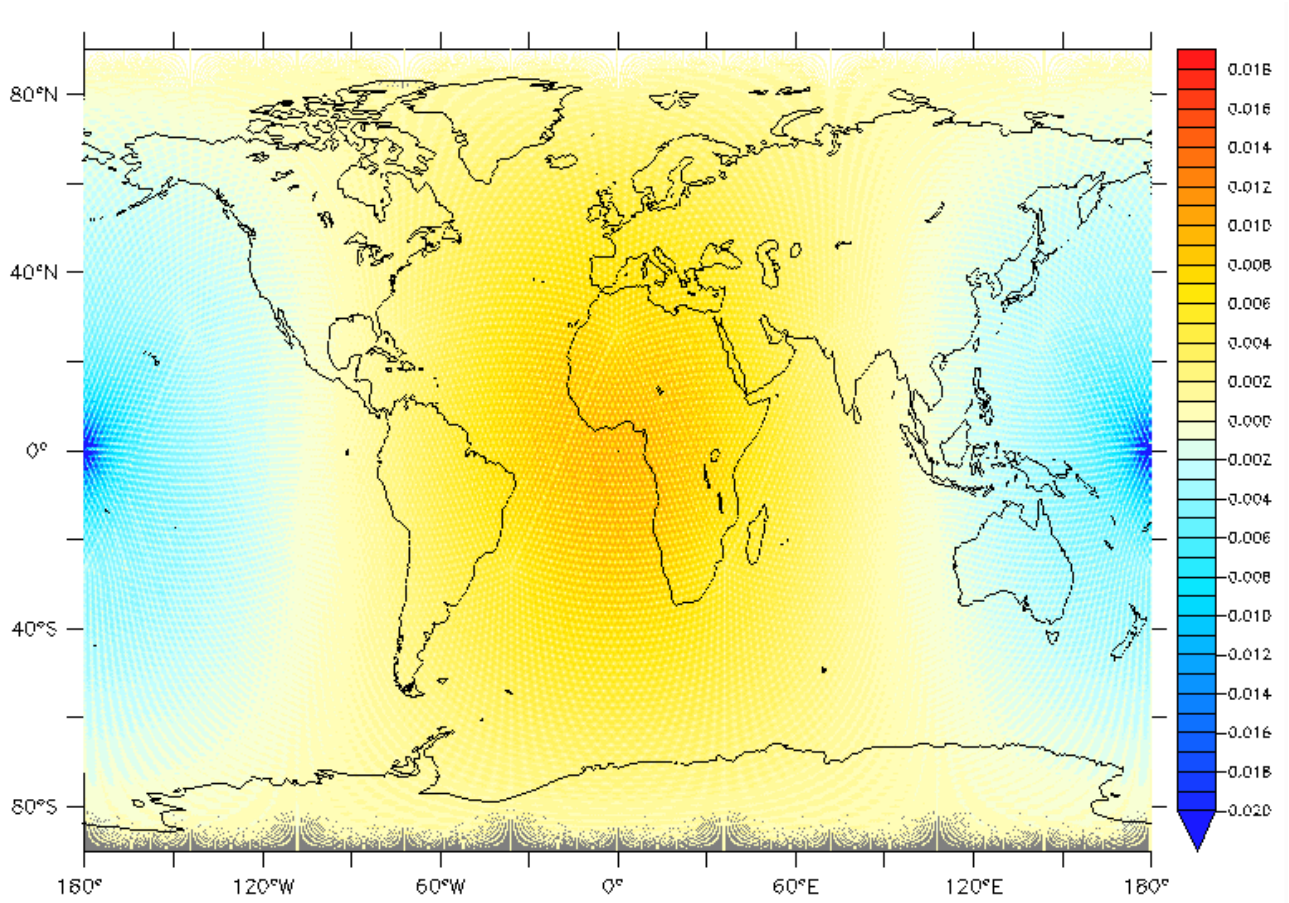

Figure 7: Relative difference for the finite element interpolation from the low resolution to the high resolution icosahedral grid (both described as unstructured)

As a remark, the finite-element interpolation fails when one of the grids is an ORCA grid, probably because of the missing points at the South Pole or of the stretched elements on the continent.

The following figures all refer to the interpolation from the reduced gaussian grid `ssea` to the regular LonLat 144x143 grid `bggd`. The nearest neighbours (single or k=4) interpolations have been performed on the grids either represented as unstructured with a Delaunay mesh and a NodeColumns functionspace or as structured with a StructuredColumns functionspace. Results are almost identical.

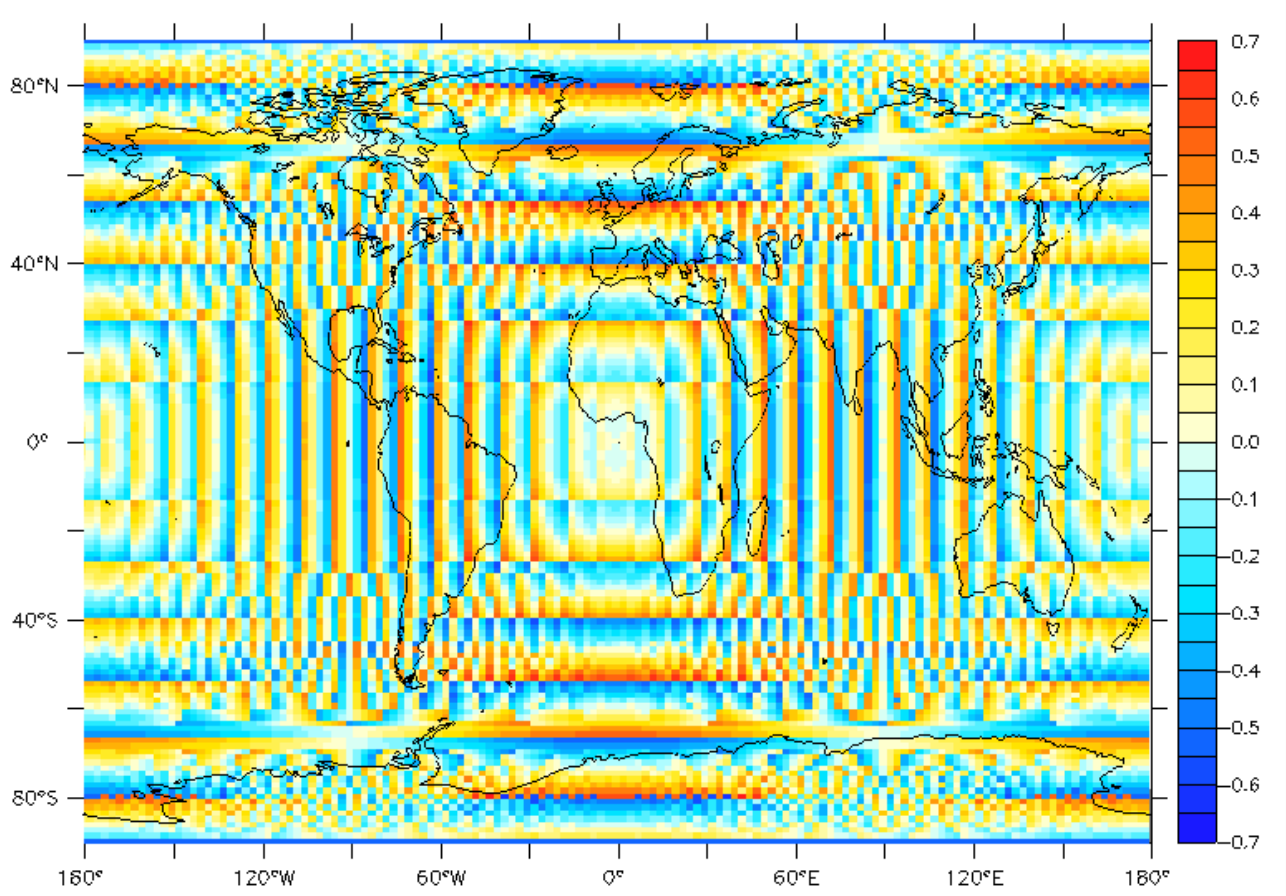

Figure 8: Relative difference for the nearest-neighbour interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (both described as unstructured).

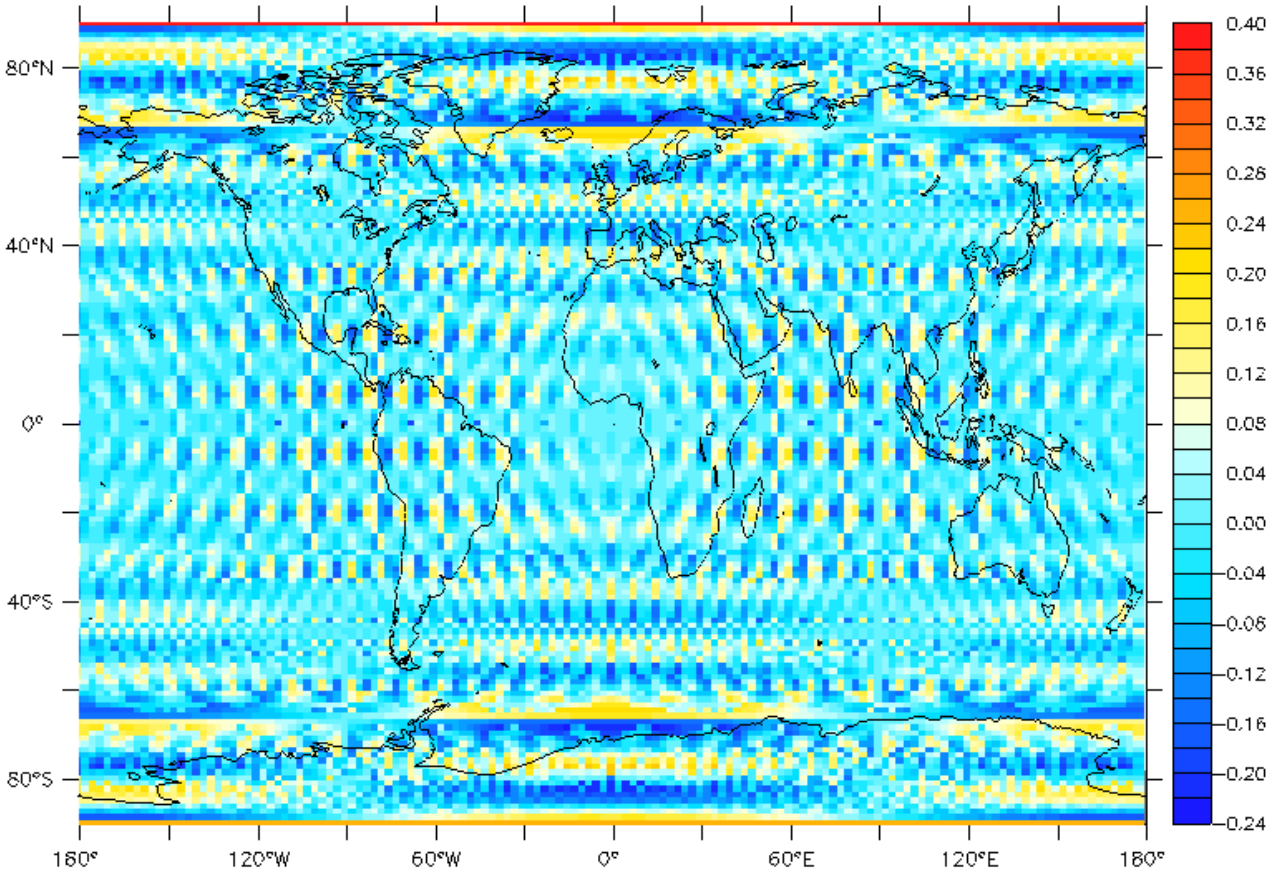

Figure 9: Relative difference for the k-nearest-neighbours (k=4) interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (both described as unstructured).

The bilinear (Fig. 10) and bicubic (Fig. 11) interpolations have been applied on the structured representations with a *StructuredColumns* function space. Both show a concentration of errors near to the coordinates axes intersection, here at 0°,0°. The colorbar has been bounded to avoid hiding other structures.

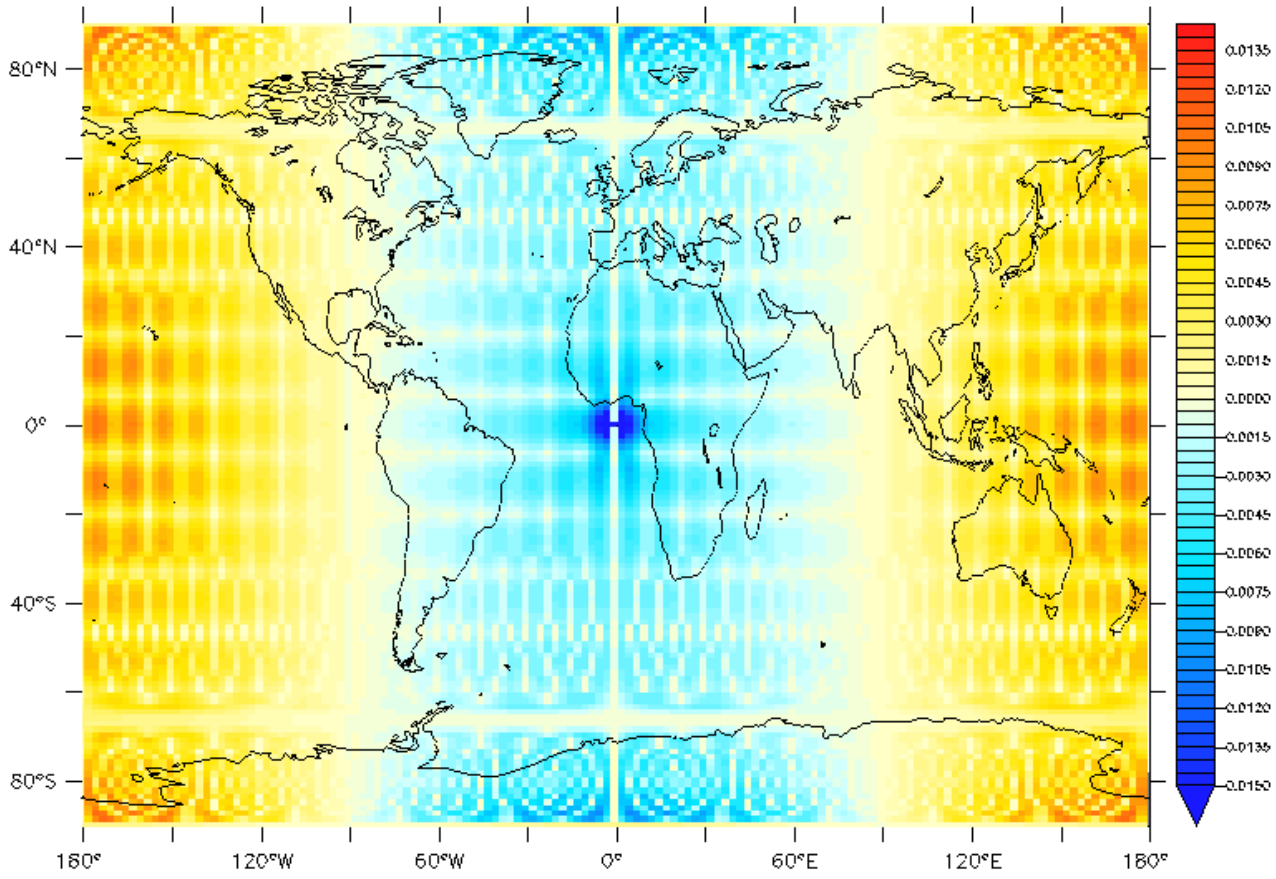

Figure 10: Relative difference for the bilinear interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (entered as structured grids).

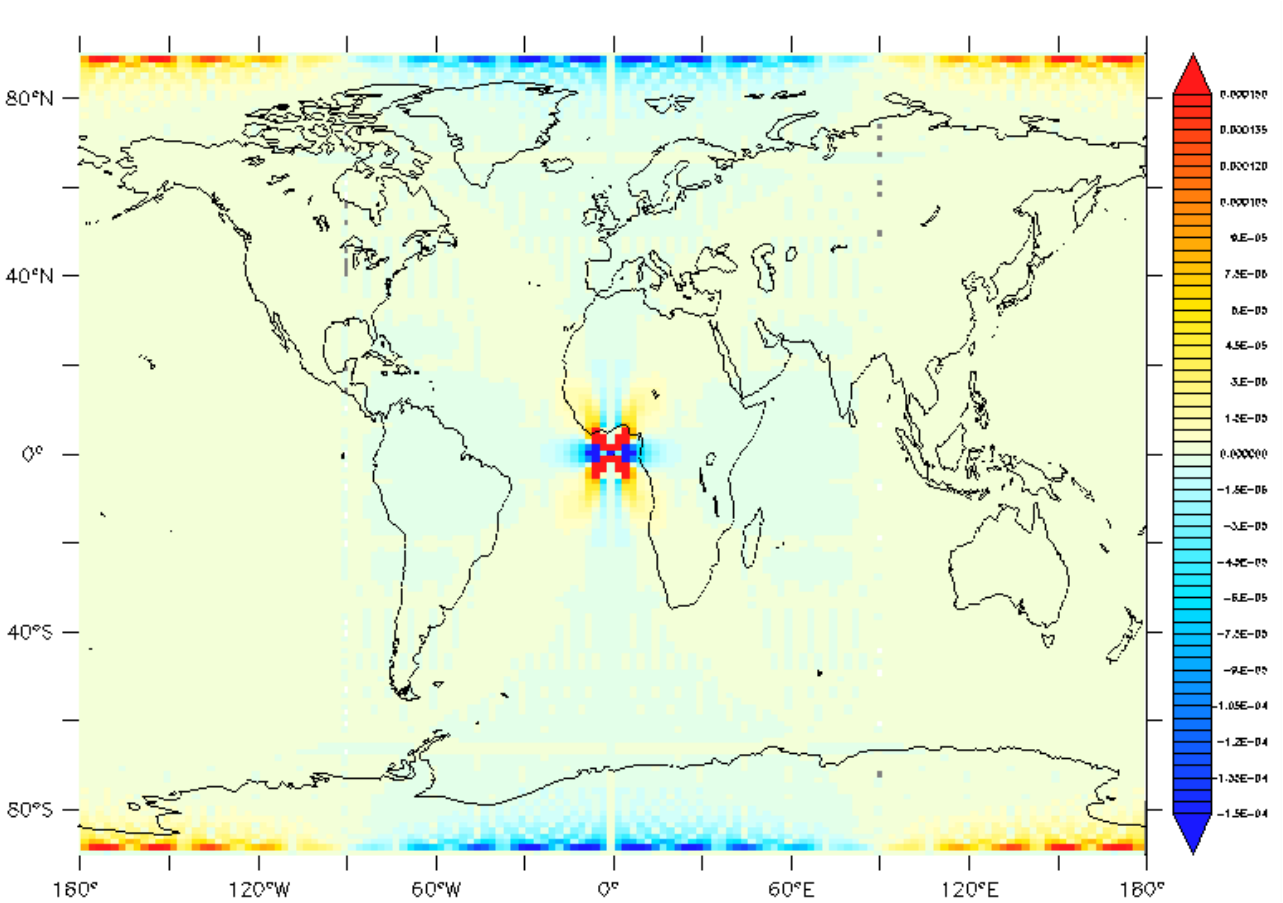

Figure 11: Relative difference for the bicubic interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (entered as structured grids).

The finite-element interpolation applied to the StructuredColumns representation (Fig. 12) sports a different error pattern, yet the error is still concentrated near to the coordinates axes intersection, here at -180°,0° (the colorbar has been bounded)

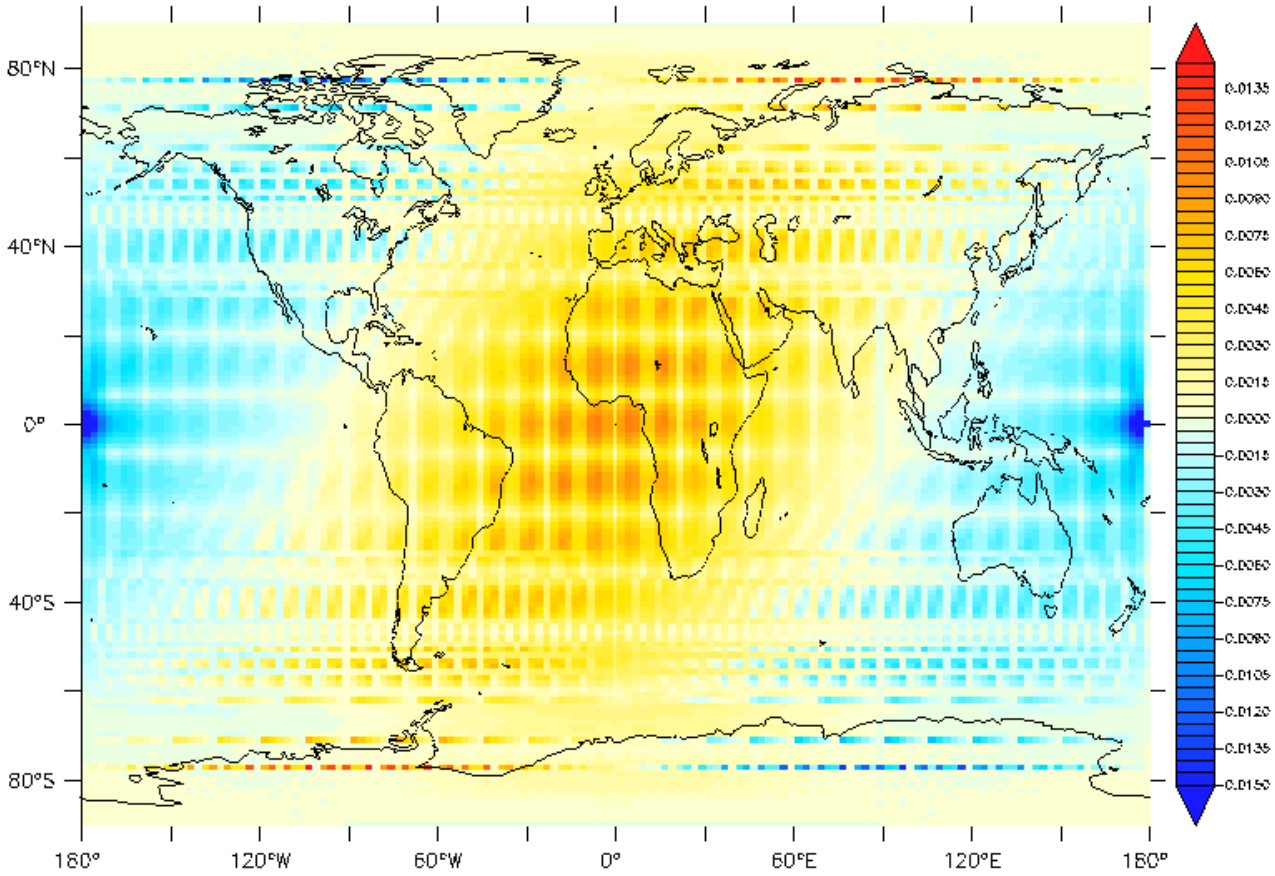

Figure 12: Relative difference for the finite-element interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (entered as structured grids)

Notice that, for these grids, the finite element interpolation on the unstructured representation (Fig 13.) fails with highly oscillating errors along longitudinal bands.

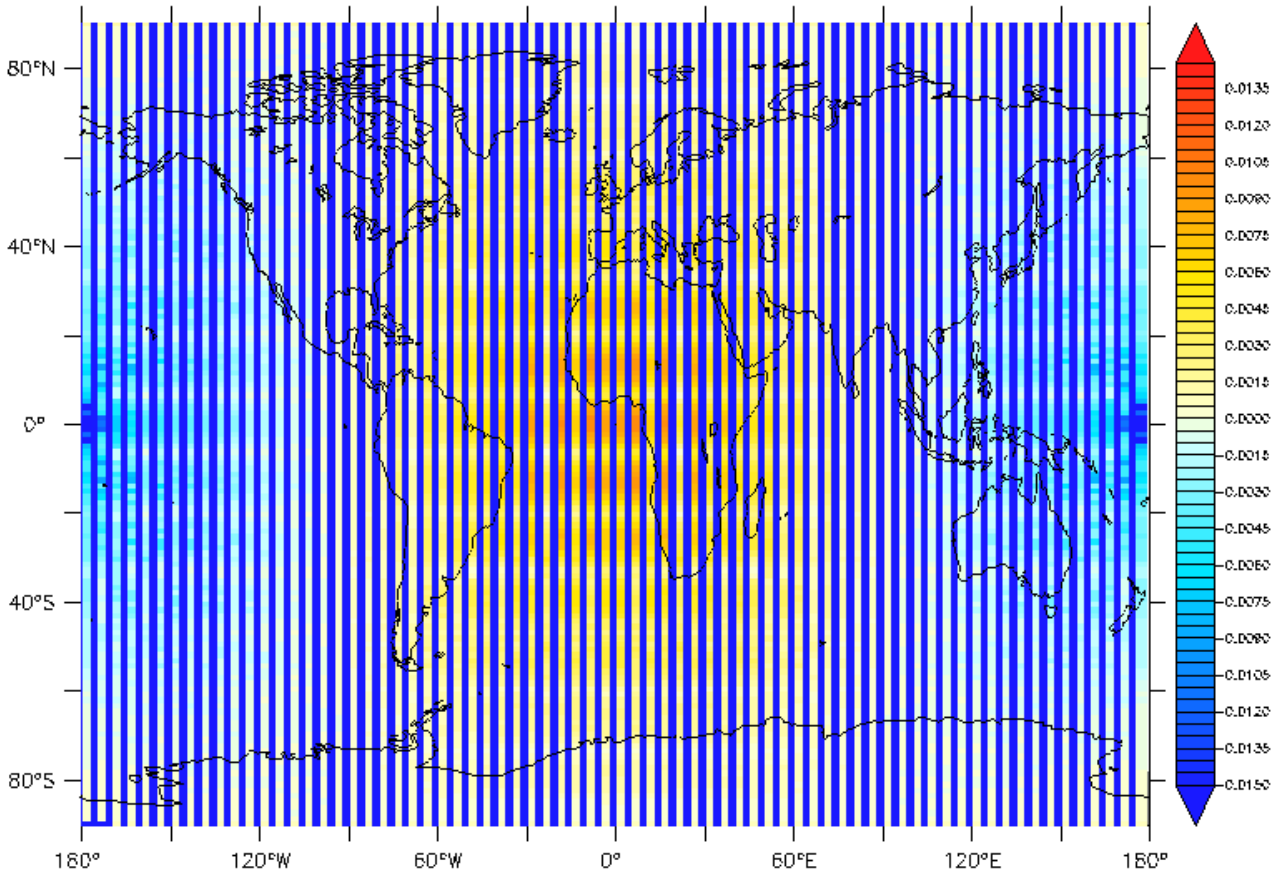

Figure 13: Relative difference for the finite-element interpolation from the reduced gaussian ssea grid to the regular lonlat 144x143 grid (entered as unstructured grids)

6. Performances

As a very basic scalability test, we have performed the construction of the matrices for the back and forth bilinear interpolations between regular grids of different resolutions on the 6 cores of an Intel Xeon E-2186M cpu (Dell Precision 7730 laptop). Timing are easily obtained by setting the environment variable `ATLAS_TRACE=1` (times from the *StructuredInterpolation2D::do_setup* sections)

	1/20° to 2°	2° to 1/20°		1/12° to 1°	1° to 1/12°
Serial	0.00503547s	12.58810s		0.01947480s	4.430450s
2 MPI procs	0.00239123s	6.56235s		0.00933619s	2.325970s
4 MPI procs	0.00136010s	3.50134s		0.00506786s	1.237620s
6 MPI procs	0.00106058s	2.61592s		0.00371443s	0.931651s

7. Conclusions

The Atlas design is certainly appealing and could provide a useful portable toolkit for the best usage of new heterogeneous architectures.
Nevertheless, its usage as a software foundation for the interpolation OASIS in the short term cannot be

recommended.

Some basic capabilities are still missing in current version 0.21, in particular the handling of missing/masked values and the whole chapter of conservative interpolations. Some constraints, as the matching parallel distributions of the source and target grids are too strict. All these aspects are the object of current developments, but no precise roadmap has been announced.

The overall design of the APIs is probably too oriented toward the coherent implementation of a full set of treatments all based on Atlas and its grid and fields representations. A flexible tool as OASIS has to be, needs to be able to account for geometries and representations coming from existing model that cannot be modified. This has been a key concept in the design of the interfaces of OASIS or ESMF or XIOS which input a model grid or mesh description with a minimal required information.
This feature will be of particular concern when implementing conservative interpolations: surface interpolation for climate models must conserve the energy integrals as they are computed by the single models and this requires to enter the exact coordinates of the "cells" that every model associates to a grid point and the surfaces of these cells as they are computed in the model.
For the moment it isn't clear how these aspects will be dealt with in Atlas.

Another relevant issue for the interpolations, especially for the conservative methods, is the treatment of singularities in polar regions or along discontinuous edges (as in reduced grids). Wrong assumptions and approximations can lead to catastrophic errors and this is an active research domain leading to different choices in different couplers.
Similarly, there are different strategies for evaluating local gradients in the second-order conservative interpolation. We don't know, for the moment, what has been foreseen for Atlas.

Because of the lack of treatment of masked points, the quality of currently implemented interpolations has not thoroughly been compared to other methods. Nevertheless we remark some spatial structures in the error patterns that would deserve a specific analysis.

Finally, from a practical point of view, the current state of the Fortran APIs is neither complete (w.r.t. C++ APIs) nor particularly flexible. No roadmap has been announced for the extension of the Fortran APIs and we don't know if they are going to be progressively abandoned when IFS will move most of its internal procedures toward C++, as for the new dynamical finite volume core.
Also from a practical point of view, there is, up to date, a painful lack of documentation and the only way of finding any example is to scroll through the non regression test procedures and the sandbox codes. Once again, only a small subset of it is coded in Fortran.

8. Acknowledgments

This work is carried on in the framework of the project IS-ENES3 which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084.

Appendix A: Installation of Atlas

All the tests have been carried on on a 6 cores Intel Xeon E-2186M cpu of a Dell Precision 7730 laptop under Fedora Core 26 64 bits.

The software environment is loaded with the following modulefiles:
1) compiler/intel/18.0.5.274
2) mpi/intelmpi/2018.4.274
3) lib/phdf5/1.10.4_impi
4) lib/netcdf-fortran/4.4.4_phdf5_1.10.4

and cmake has been upgraded to version 3.17.2

The main documentation for atlas installation can be found on
https://sites.ecmwf.int/docs/atlas/

The required dependencies are:
CMake --- For use and installation see http://www.cmake.org/
ecbuild --- ECMWF library of CMake macros
eckit (with MPI support) --- C++ support library

And the recommended dependencies are:
fckit --- For enabling Fortran interfaces
python --- only when Fortran bindings are required)
cgal --- if Delaunay triangulation of unstructured grids is needed

Installation instructions for ecbuild are on
https://sites.ecmwf.int/docs/atlas/getting_started/install_ecbuild/
only the `CMAKE_INSTALL_PREFIX` key is needed.
The `ecbuild_ROOT` environment variable should point to that prefix and `$ecbuild_ROOT/bin` has to be added to the shell `PATH`.

Installation instructions for eckit are on
https://sites.ecmwf.int/docs/atlas/getting_started/install_eckit/
Notice that the `CC`, `CXX`, `MKL_ROOT` and `MPI_ROOT` have to be set before invoking cmake and that, if only atlas needs to be built on top of eckit, a lighter version can be compiled:

```
cmake ../ -DCMAKE_MODULE_PATH=$ecbuild_ROOT/share/cmake/ecbuild \
          -DCMAKE_INSTALL_PREFIX=xxx \
           -DENABLE_MPI=ON           \
           -DENABLE_TESTS=OFF        \
          -DENABLE_ECKIT_SQL=OFF    \
          -DENABLE_ECKIT_CMD=OFF    \
          -DENABLE_ARMADILLO=OFF    \
          -DENABLE_VIENNACL=OFF     \
          -DENABLE_CUDA=OFF         \
          -DENABLE_AEC=OFF          \
          -DENABLE_XXHASH=OFF       \
          -DENABLE_LZ4=OFF          \
          -DENABLE_JEMALLOC=OFF     \
          -DENABLE_BZIPS2=OFF       \
          -DCMAKE_DISABLE_FIND_PACKAGE_Doxygen=ON
```

The `eckit_ROOT` environment variable should point to the installation prefix and `$eckit_ROOT/bin` has to be added to the shell `PATH`.

Installation instructions for eckit are on
https://sites.ecmwf.int/docs/atlas/getting_started/install_fckit/

Notice that the CC, CXX and Fortran have to be set before invoking cmake and that, if only atlas needs to be built on top of fckit, a lighter version can be compiled .

```
cmake ../ -DCMAKE_INSTALL_PREFIX=xxx \
          -DENABLE_TESTS=OFF
```

The fckit_ROOT environment variable should point to the installation prefix and $fckit_ROOT/bin has to be added to the shell PATH.

Tesselation is needed for Delaunay triangulation of unstructured generic meshes, therefore cgal is a prerequisite
Version 4.10.1 can be installed via dnf under FC26. Notice, however, that current version is 5.0.3

```
[root ~]# dnf install CGAL
[root ~]# dnf install CGAL-devel
```

Installation instructions for Atlas itself are on
https://sites.ecmwf.int/docs/atlas/getting_started/installation/
Notice that the CC, CXX and Fortran have to be set before invoking cmake

```
cmake ../ -DCMAKE_INSTALL_PREFIX=xxx \
          -DENABLE_TESSELATION=ON \
          -DENABLE_DOCS=ON \
           -DENABLE_SANDBOX=ON
```

The -DENABLE_DOCS=ON option is not needed for production installation and the -DENABLE_SANDBOX has been useful only for the current analysis
The atlas_ROOT environment variable should point to the installation prefix and $atlas_ROOT/bin has to be added to the shell PATH.

If the installation has succeeded, the atlas --info command should provide a full report on the installation.

Even if cmake is the preferred compilation environment, in particular if wrapped by ecbuild macros, applications using Atlas can be compiled with simple Makefiles accessing the module directories under the Atlas and fckit installation roots and the atlas, eckit and fckit libraries in the corresponding lib installation directories.

Appendix B: Needed tweaks

In order to write the NetCDF output of fields represented on a StructuredColumns functionspace in a consistent way with fields represented on a NodeColumns functionspace, we rely on the xy coordinated of the points or of the nodes, gathered on the master process, the one writing the NetCDF output.
As in version 0.21, the gather of the xy coordinates fails for the StructuredColumns functionspace because the shape of the xy field has not been declared.
We had to add the
`field_xy_.set_variables( 2 );`
instruction at line 547 of
`src/atlas/functionspace/detail/StructuredColumns_setup.cc`

As in version 0.21, interpolation can be set up and performed in parallel only if the domain decomposition of the target grid matches the domain decomposition of the source grid.
The C++ APIs expose a matching partitioner object both for Meshes and for FunctionSpaces. The latter is strictly necessary when working with StructuredColumns functionspaces, since the connectivity is inferred and no Mesh object is generated.
The Fortran API, in version 0.21, only exposes the `atlas_MatchingMeshPartitioner` object.
We had to create a public `atlas_MatchingFunctionSpacePartitioner` object in `src/atlas_f/grid/atlas_Partitioner_module.F90`, declare its constructor interfacing the C wrapper of
`atlas__grid__MatchingFunctionSpacePartitioner__new` and propagate its public declaration to the `atlas_module` in `src/atlas_f/atlas_module.F90`

Because of the still limited handling of polymorphism in F2003, some redundant coding is needed even when the syntax is common to all concrete implementations.
As an example, the base `atlas_Functionspace` type from
`src/atlas_f/functionspace/atlas_FunctionSpace_module.F90`
does not contain any field, but only some private or generic procedures.
The `gather` procedure is defined in all the extending functionspace types, but the PointCloud and, therefore, it is not defined in the base type.
In Fortran, a variable declared as
`CLASS(atlas_FunctionSpace), ALLOCATABLE :: functionspace`
can be later "cast" into a concrete type with a mold allocation, as in

```
SELECT CASE(TRIM(fs_in%name()))
CASE('NodeColumns')
  ALLOCATE(atlas_functionspace_NodeColumns::functionspace)
  functionspace = field_res%functionspace()
CASE('StructuredColumns')
  ALLOCATE(atlas_functionspace_StructuredColumns::functionspace)
  functionspace = field_res%functionspace()
END SELECT
```

yet, the Fortran compiler does not accept any reference to typebound procedures that are not declared in the class, unless they are referenced in a `SELECT TYPE` construct.
Since the `TYPE IS` switch does not allow for more than one type, we had to introduce redundant constructs as in:

```
SELECT TYPE(functionspace)
TYPE IS(atlas_functionspace_StructuredColumns)
   CALL functionspace%gather(lonlat,lonlat_g)
   CALL functionspace%gather(field_res,field_res_g)
   CALL functionspace%gather(field_true,field_true_g)
   CALL functionspace%gather(field_diff,field_diff_g)
TYPE IS(atlas_functionspace_NodeColumns)
   CALL functionspace%gather(lonlat,lonlat_g)
```

```
      CALL functionspace%gather(field_res,field_res_g)
      CALL functionspace%gather(field_true,field_true_g)
      CALL functionspace%gather(field_diff,field_diff_g)
  END SELECT
```

A thorough design of Fortran APIs should either use abstract classes with deferred procedures (making, nevertheless a bit more cumbersome the use of broader classes as routine arguments) or enrich as much as possible the base class with stub typebound procedures interfaces that the concrete extensions will override, provided that the generic interfaces can still be resolved by the compiler.