ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Report

High Performance Computing for Weather and Climate: Domain-Decomposition on the Cubed-Sphere

Holstein Christoph
Nick Nathalie
Shekhar Ankit

Swiss Federal Institute of Technology
ETH Zurich

Supervision

Dr. Jeremy McGibbon
Dr. Oliver Fuhrer

August 2021

Acknowledgements

Throughout our project, we gained new knowledge and fundamental insights into High Performance Computing and climate simulations and we would like to acknowledge the people who made this possible.

Our acknowledgment goes to Dr. Oliver Fuhrer, lecturer at ETH Zurich at the Department of Environmental Systems Science and team leader of the unit "Numerical Prediction: Development" at the Federal Office of Meteorology and Climatology MeteoSwiss. His help and support were greatly appreciated at all times of the project and we would like to thank him for the valuable and dedicated inputs.

Our further acknowledgment goes to Dr. Jeremy McGibbon, expert in Boundary Layer Meteorology, Higher-Order Turbulence Closure, Machine Learning and Model Development, for his supervision of the project and professional support. His willingness to give his time was very much appreciated. He provided us with very valuable and constructive inputs and advice.

Contents

List of Figures

1. Introduction and Goal

The cubed-sphere is a popular grid used in global weather and climate models. State-of-the-art weather and climate simulations rely on large and complex software running on supercomputers. This course focuses on programming methods and tools for understanding, developing and optimizing the computational aspects of weather and climate models. Emphasis will be placed on the foundations of parallel computing, practical exercises and emerging trends such as using GPUs.

This project will start from an existing domain-decomposition code written in Python which supports halo-updates, scatter and gather. The goal of the project will be to extend the existing code to 1) generalize the decomposition to allow for uneven distribution of grid points and 2) implement periodic boundary conditions on a single cube face.

2. The Background

At the beginning of the project, an existing code was handed out and provided on which the new implementations should be carried out. The existing code works for squared layouts. There are parts in the existing code that do not run for uneven layouts. It is important to have a code base that also runs for uneven layouts because the aim is to take advantage of uneven layouts for the results. The application was a decomposition (climate models etc.). In Figure 1, a general illustration of our cubed-sphere is depicted. It has six tiles and ranks distributed according to the selected layout. For an uneven layout (such as 5x3) the edges of the red marked ranks in Figure 1 are not well-defined in the existing code. This is because the orientation of the neighbouring tiles of those edges is rotated by either 90Â° or 270Â° (illustrated in Figure 2). Therefore the boundaries at the edges and the communication between them needed to be rewritten. Hence, the aim of the project was to come up with an own mathematical solution.

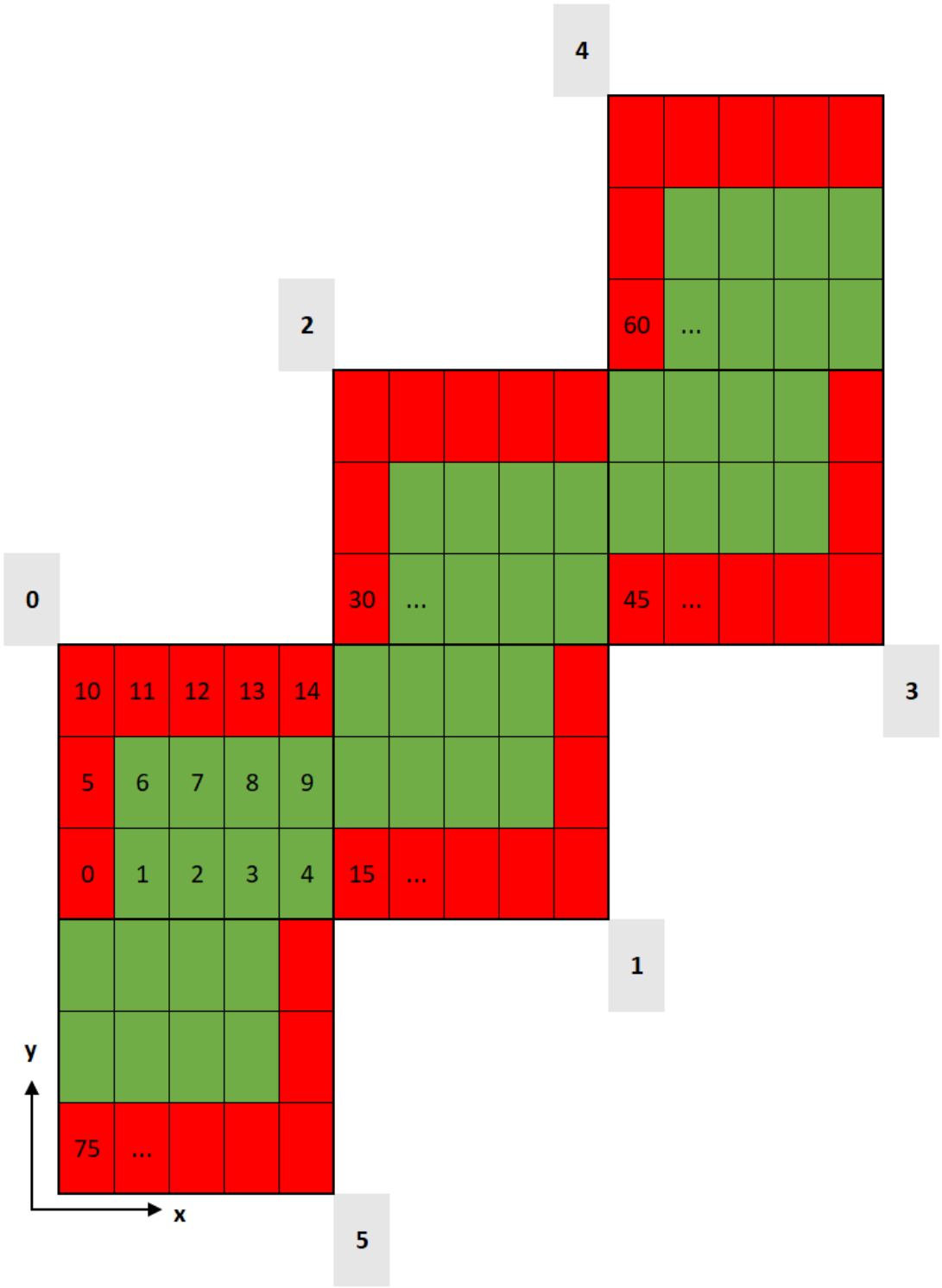

Figure 2.1: Illustration of step stair-case cube layout (5x3) with ranks marked that worked/not-worked with the existing code (in green/red). We modified the existing code base for the code base that handled the red ranks.

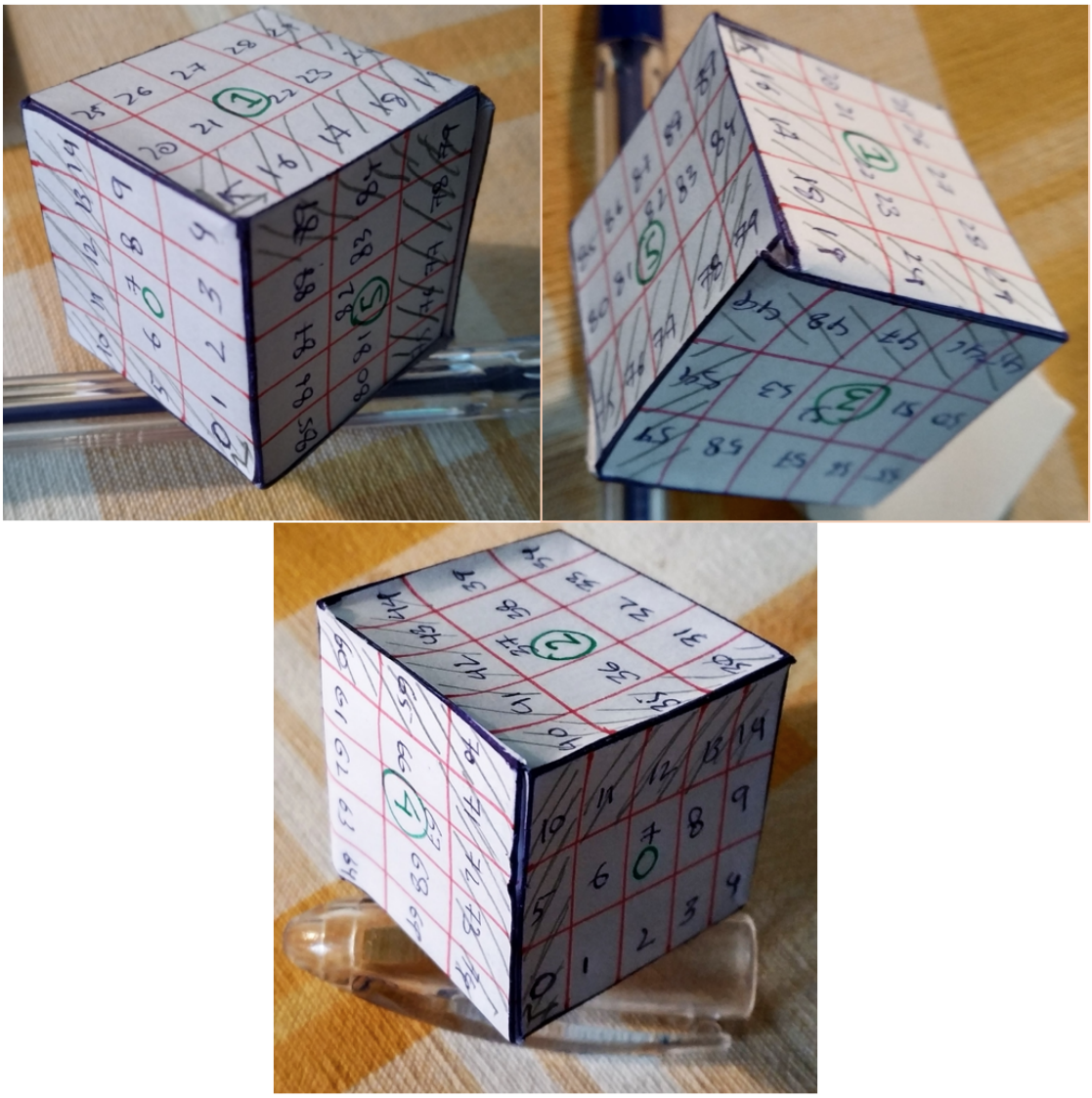

Figure 2.2: Illustration of cube-sphere of uneven layout (here 5x3 shown in red grids). The tile numbers are shown in green circles. The rank number in each tile is shown in blue letters. The sharing boundary (corresponds to red shaded boundary in Figure 1) and their corresponding ranks are shaded (with pencil).

3. The solution

There are in total four functions which have been implemented and will be explained in the following section.

The first two functions return the boundary sequence. That is, it will output how many boundaries have to be created for the edge of a single rank. The lr-boundary-seq function returns how many boundaries are shared on the left and right edges. For the upper and lower edges (ul) the same procedure applies. It has been cut into two pieces (left-right and upper-lower) because once we are dealing with continuation in the x-axis, and another time in the y-axis. These functions above are applied if the problem at hand deals with rotations at the boundaries over the edges (red ranks). Otherwise, the results would be trivial and already implemented in the existing code. In the end, a list of the boundary sequence across all ranks on a respective edge on a tile is returned.

The next functions return what ranks are on the other side of the edge and which of those the data of the respective rank is shared with. It is again split into left-right and upper-lower functions. If we consider the left edge of tile 0, the corresponding edge would be the top edge of tile 4. First, it has to be determined which ranks are potential sharers on the other tile (to-ranks-pot). Additionally, using the boundary sequence of the function before and the position of the original rank, the shared ranks are returned in a list (to-ranks-seq). These ranks share a boundary with the original rank.

In the next step these functions are executed in parts of the original code (class CubedSpherePartitioner) to create the respective boundaries. In a squared layout, only one boundary per edge gets defined, which is not necessarily the case for uneven layouts. So instead, a list of boundaries for that respective edge gets defined by length to-ranks-seq and filled with the returned values from our written functions.

In Figure 3 an example of the whole process is shown until the list of boundaries is returned, considering a layout of 5x3 and the left edge of rank 5. The shared ranks are 71, 72 and 73.

```
class157@nid02768:~/HPC4WC/projects/2021/group05/fv3gfs-util/tests/our_tests> pytho
n partitioner_test.py
layout = (5, 3)
rank =  5
Calculations for the left edge
boundary_seq = [2, 3, 2]
to_ranks_pot =  [74 73 72 71 70]
to_ranks_seq =  [73 72 71]
List of Boundaries:
[SimpleBoundary(from_rank=5, to_rank=73, n_clockwise_rotations=1, boundary_type=0),
 SimpleBoundary(from_rank=5, to_rank=72, n_clockwise_rotations=1, boundary_type=0),
 SimpleBoundary(from_rank=5, to_rank=71, n_clockwise_rotations=1, boundary_type=0)]
```

Figure 3.1: Execution of the written functions. Starting with a cube layout of 5x3 and rank 5. Returns 3 boundaries with different to-rank.

Finally, other small parts of the partitioner.py file needed to be changed to allow a calculation for uneven layouts, which are not mentioned in detail in this report.

4. Conclusions and Outlook

4.1 Conclusions

This project was started from an existing domain-decomposition code written in Python which supports halo-updates, scatter and gather. The goal of the project was to extend the existing code to 1) generalize the decomposition to allow for uneven distribution of grid points and 2) implement periodic boundary conditions on a single cube face.

The uneven distribution part has been successfully implemented and the code extended. Since our goal was to allow the calculations of shared ranks for any given layout, a more general approach was needed. In this context, it is recommended to always have a cube out of paper at hand to visualize the different steps better and for the capacity for spatial thinking. The domain composition now returns the correct results for any layout possible.

4.2 Outlook and next steps

The next major step would be to make additional changes to the code in other areas such that returned boundary lists can work with the halo update and the communication of data between ranks. This was the part which was not possible to implement. The corresponding changes would mainly affect the files "communicator.py" and "boundary.py". At the moment these files work with a single boundary object instead of a list that is returned in the scope of the project.