The C++ fregrid project - towards instantaneous regridding and exchange grid generation

Miguel R. Zuniga, SAIC at NOAA GFDL, Princeton, NJ, September 2023.

Background

Much faster regridding could be an extremely useful feature for climate and weather modeling, and the existence of the feature may lend itself to more convenient workflows and possibly altogether new workflows. The existing regridding capability of GFDL's NCTools, particularly in the fregrid application, is known to be extremely compute intensive and has been targeted for porting on GPUs. Additionally the computational complexity of fregrid was studied in NCTools issue #202 (https://github.com/NOAA-GFDL/FRE-NCtools/issues/202), which suggested approaches for its improvement.

The work task of this report selected the NCTools 2nd order conservative regridding weight generation function named "create_xgrid_2dx2d_order2" as the initial target for improvement. In the course of the work it was possible to easily generalize and expand beyond that function. This report introduces and summarizes those capabilities and other relevant items that were learned.

A high level view of regridding and exchange grid generation.

The process of regridding (or remapping) takes data that is on an underlying source grid and calculates its values on the target grid. Field data that is on the source grid is generally per source grid cell, and target grid cells that overlap a source grid cell will have field data values in those source grid cell(s). As one might imagine, the target field values will depend on the "amount" of overlap. To be slightly more precise, cells of the target grid are "clipped" (with a polygon clipping algorithm) with cells of the source grid. The clipping results on polygons that cover each target cell and the target grid (generally in its entirety). Via a comparison of the area of the polygons and the area of the source cells, field data on the target cells are calculated.

The term "xgrid" in the function name stands for "exchange grid", and it is appropriate because what is described in the preceding paragraph is not just a description of the central calculation for regridding in create_xgrid_2dx2d_order, but it is also the central calculation for determining what are commonly known as the component (atmosphere, land, ocean) exchange grids in NCTools applications such as make_coupler_mosaic.

Grid cell overlaps and near neighbors algorithms

It is intuitively obvious that geographically distant cells should not contribute to the remapping of a field. E.g. a cell over the north pole on the source grid one should not affect the data for a cell covering an equatorial point on the target grid. Conversely, geographic near neighbors may contribute to the remapped values. In the fregrid software, when two polygons do not overlap the polygon clipping algorithm does not return a polygon; when they do overlap it returns a new polygon that forms part of the exchange grid.

Polygon clipping is an expensive calculation and the baseline software avoids calling clip_polygon when it can pre-determine that two cells would not clip to a new polygon. However, as it was determined in NCTools issue #202, the computational complexity of the baseline create_xgrid_2dx2d_order2 function remains "quadratic" despite the avoidance of the many of the clip_polygon calls. Specifically, there is a quadratic (i.e. O(Ns x Nt)) number of latitude (and or longitude) comparisons performed, where Ns and Nt are the number of source grid cells and the number of target grid cells, respectively. This quadratic complexity is the source of the well known high execution times of fregrid remapping weights file generation.

Approaches and primary features developed for faster regridding

There are two approaches developed for faster regridding: one is the porting to GPU what is essentially the baseline algorithm; and the second is the development of search tree data structure to efficiently determine the near neighbors, along with redesigning the baseline algorithm to use the search tree.

C++ and functions common to the two approaches.

C++ is used as the project language and it provides fundamental components of each of the regridding methods. For the software targeting GPUs, it provides the standard parallel copy_if algorithm; for the search tree code it provides a high convenience for library creation.Components of NCTools that were converted to C++ include the entire libfrencutils directory and library, and of course the fregrid directory. One new library (libsearch) created by the project and used by both approaches includes: a 3D axis aligned bounding box class; the function getBoxForSphericalPolygon, which creates a bounding box for polygons on the sphere; and functions such as get_cell_idxs_ccw_4 which help encapsulate the use of the underlying polygon mesh that NCTools grids implicitly assume and use. Additionally the box query search algorithms are encapsulated in classes in that library.

Create_xgrid_2dx2d for the GPU

This version is called create_xgrid_2dx2d_bf and it uses the C++ standard parallelism function copy_if and inexpensive box-box intersection tests, to determine in a brute force manner, the candidate set of pairs of indices of the cells of the source grid and the cells of the target grid that may overlap. No indices that actually do overlap as determined via the polygon clipping algorithm are missed. Once the indices are determined, the process of polygon clipping and exchange grid cell area calculation occurs. In principle, the number of times clip_polygon is then called is not much greater than the number of cells in the resultant equivalent exchange grid , so it should be proportional to (Ns + Nt), or proportional to the number of cells of the larger input grid.

So the function create_xgrid_2dx2d_bf remains with an O (Ns x Nt) computational complexity, but as coded using copy_if, the candidate pair of indices can be quickly generated in parallel even for fairly large grids. In principle, the software can be compiled to run serially or in parallel on CPUs, but for this project we only present data for Nvidia compilers targeting offloading to Nvidia GPUs.

Create_xgrid_2dx2d using the search tree

The function create_xgrid_2dx2d_st uses bounding boxes and a search tree named the DITree The DITree supports “box queries” : it maintains a collection of boxes and users can query it to efficiently find the boxes in the collection that overlap the input query box.
Create_xgrid_2dx2d_st places the bounding boxes of the cells of the target grid into the search tree, and for query boxes it uses the bounding boxes of the cells of the source grid. For determining overlap of the source and target, the routine has computational complexity of O(Nt x log Nt) + O(Ns x log Nt), where the first term is due to the tree creation, and the second term is due to searching the tree. This complexity is far lower than that of the baseline. Also, as in the previous method, the number of times the clip_polygon function is evaluated should be proportional to the number of cells of the larger input grid.

Results

Hardware and Compilers

The CPU and GPU used were the AMD EPYC 7742 and the Nvidia V100 32GB, respectively. The baseline code as well as the C++ code using the search tree were compiled with GNU g++ 12.3 using “-O2" optimization flags. The C++ code using the brute force method was compiled with Nvidia nvc++ version 23.7, using “-O2 -std=c++20 -stdpar=gpu” optimization flags.

Reproduction tests

Numerous tests were done for regridding weight file generation for cubed-sphere to cubed-sphere grids, cubed-sphere to lat-lon grids, lat-lon to lat-lon grids, and tripolar to lat-lon grids. In all cases the search tree method (create_xgrid_2dx2d_st) bitwise reproduces the baseline. Also, in all cases the brute-force method on GPU (create_xgrid_2dx2d_bf) reproduces the baseline to within tolerance.

Complexity/Scaling tests

Table 1 below compares the total run time of the three fregrid versions for generating the regridding weights files using 2nd order conservative interpolation. The generated weights files are for regridding the cubed-sphere to regular lat-lon grids. Only one CPU core and one GPU is used. The size of the source grids and target grids are each increased by a factor of sixteen in each increasing row. The reported time is the total program run time in seconds.

Table 1 fregrid run time

Grids Mapped	Baseline fregrid	Search tree fregrid	GPU brute-force fregrid
C48 -> R90x180	0.1	0.1	0.4
C192 -> R360x720	11.8	1.0	1.0
C768 -> R1440x2880	2711.3	16.09	48.0
C3072 -> R5760x11520	1) ~1e5 to ~1e6 extrapolated from above. 2) (extreme/parallel fregrid has been seen to take a few hours on Gaea; but Pu Lin reported about 2 minutes with a few hundred CPU cores.)	280.7	Fails on GPU memory allocation. Extrapolates to ~1e4

Resolved Issues and Surprises

1. Function getBoxForSphericalPolygon. The most difficult part of this project was creating the 3D axis aligned bounding boxes for polygons residing on a spherical grid . An algorithm was not found in the literature, and three iterations of the routine were done before the new fregrid could reproduce the baseline fregid without adding ad-hock box expansions (which can cause search inefficiencies).
2. Use of the c++ std:cartesian_product. The std::cartesian_product was planned for use by the function std::copy_if in create_xgrid_2dx2d_bf is a new c++ 2023 feature. It allows the function copy_if to work with tuples or pairs of indices of grids; and we consider its use the natural_way of using std::copy if for near neighbors determination between two grids. Unfortunately, the nvc++ compiler is a C++20 compiler using the C++20 standard library that does not (yet) support the cartesian_product. Experimental or unofficial cartesian_product implementations were obtained from the web and tried but would not compile. Instead, a workaround using copy_if with a 1D std::iota encoding the 2D index pairs was created. In principle the software can be left as is, but it is possible it is more performant with the cartesian_product and it should be tried when ncv++ becomes C++2023 compliant.
3. The development gcc C++ headers and libraries were version 8.3 and not compatible with nvc++. Ultimately compilation was possible by providing an alternate gcc toolchain to nvc++ using the “not too well known” compilation flag “--gcc-toolchain”. As a result of the added complexity a CMakePresets.json was added to help with the compilation on selected targets by future users.

Secondary features developed

1. The functions create_xgrid_2dx2d_bf and create_xgrid_2dx2d_st were modified to also handle the first order conservative interpolation case in addition to the second order case.
2. The C++ software allocates the amount of memory as determined at runtime. (Use of the legacy compile-time variable MAXXGRIDS was removed from the tested C++ code, though MAXXGRIDS is not yet removed from all code in the new libfrencutils.)
3. The C++ project is a multi CMakeLists.txt CMake project, with simple CMakeLists.txt per directory. This allows for simple integration with common IDEs, and simpler project management than alternatives.
4. True unit testing is part of the project. CTest is used and with added productivity enhancement frameworks. Catch2 for easy C++ unit test creation is one of the frameworks.

Ideas for future development of NCTools

The work done in this project is in part with the goal of the eventual simplification, improved maintenance and improved efficiency of the various NCTools and not just fregrid. Obviously any components of NCtools can be targeted for conversion to C++, but here are some specific suggestion that should make good leverage of the existing C++ work and that perhaps should be prioritized :

A. Remaining tools that perform regridding and exchange grid generation should be redesigned to use the new search algorithms. This includes but is not limited to make_coupler_mosaic and remap_land.
B. Components with extensive amounts of lines concerning memory allocation and deallocation, or that use fixed size array buffers, should be modified to use dynamic size C++ classes such as std::vector. This includes those components that use the MAXXGRID variable, and includes some classes in globals.h where array buffers are defined (e.g. interp_config structure)
C. The components of C++ fregrid that perform the actual remapping can be parallelized. The easy way to effectively do this is through profiling and converting the key hotspots to use the parallel std::for_each (or std::for_each_n) algorithms in lieu of normal for loops. Note that it has been observed that although the existing baseline code can use OpenMP threads, not enough of the loops have been covered with OpenMP.
D. The DITree data structure is a simple version of the DETree which was first described in "Ray Queries with Wide Object Isolation and the DE-Tree" M. Zuniga and J. Uhlmann, JGT, 2006. The data structure and its use may be further refined so that the total execution time is greatly reduced - perhaps by a factor of 10! Additionally, it is also possible to parallelize key components. This improvement along with the others mentioned above would allow NCTools to perform seemingly instantaneous regridding and exchange grid generation.

Some ideas for secondary improvements are :

E. Refactoring components to use the C++ Eigen matrix library and the C++23 std:::mdspans. There are numerous benefits of doing this, e.g. improved code efficiency, compactness, easier maintenance, and safety.
F. Adding additional true unit tests.
G. Improving the speed of the search tree.

Code location

The software is located in the cpp directory of the GFDL NCtools github repository, on branch cpp_dev.

Documents associated with the projects:

1. “Modern C++ for GFDL”. M Zuniga

2. “C++ for GFDL?”, M Zuniga (a presentation)