Multi-Radar Mosaic Process

October 17, 2017

1 Introduction

In the previous phase, our major works were redesigning the gridding algorithm which was: (1) significantly outperforming; (2) highly optimized for SIMD; and (3) shorter codes (less than a hundred lines) than the existed gridding algorithm in the Radx. From the previous phase, we obtained 3D grids from multiple radar stations - note these 3D grid are completely separated, each grid is surround the corresponding radar station, in other word, radar station locates at (0,0,z) where z is the altitude of the radar station antenna. We have created a class called Cart2Grid to store and handle the 3D grid data. It receives the size of the grid from the user input parameter and creates the grid by using it. The grid comprises three-dimensional vector which contains the actual value of output data such as Reflectivity and Velocity, for each gate of the radar. The atomic operation is used to prevent from intervening by many threads when it comes to update and store the data to the grid, due to the numerous threads generated by TBB.

By applying our high-performance gridding algorithm and multi-threads approach to Radx, we have achieved the improvement of Radx's Gridding performance. However, the gridded data from the previous phase has come from a single radar station. Consequently, what we need to do following gridding process is a mosaic process. Since an individual NEXRAD radar is limited to a range of about 200 miles and each single grid is not large enough to cover an entire hurricane system; hence, it is imperative to combine information from multiple radars to give a regional or national view of reflectivity or precipitation. Consequently, a mosaic product is produced for regions spanning several hundred to several thousands of miles.

We will develop a new application named RadxMultiGrid to extend the capability of Radx from single to multi-radar scenarios. Each RadxMultiGrid process projects gridded data from a radar station, and there will be a master process that collects data from other RadxMultiGrid processes and combines the each grid data into one large grid and then writes it to hard disk.
The rest of this report are organized as following. Section 2 presents the overview of the Mosaic steps and the concept of geographic projections which are used for the Mosaic process; in Section 3, more details of each step is described.

2 Overview

Multi-radar mosaic takes output files from Radx2Grid(Plus), embeds them into a uniform domain defined by user. Typically, this domain is large enough to cover entire study area (e.g. US). This domain is represented as a 3D grid, just like a single radar domain. In this uniform domain, radar station is not located at (0,0) anymore, unless such PCS is specifically defining such that radar station at (0,0).

It has several steps to combine multiple radar stations' data:

1. Converting a radar-centric Cartesian coordinates to longitude and latitude (a.k.a GCS)

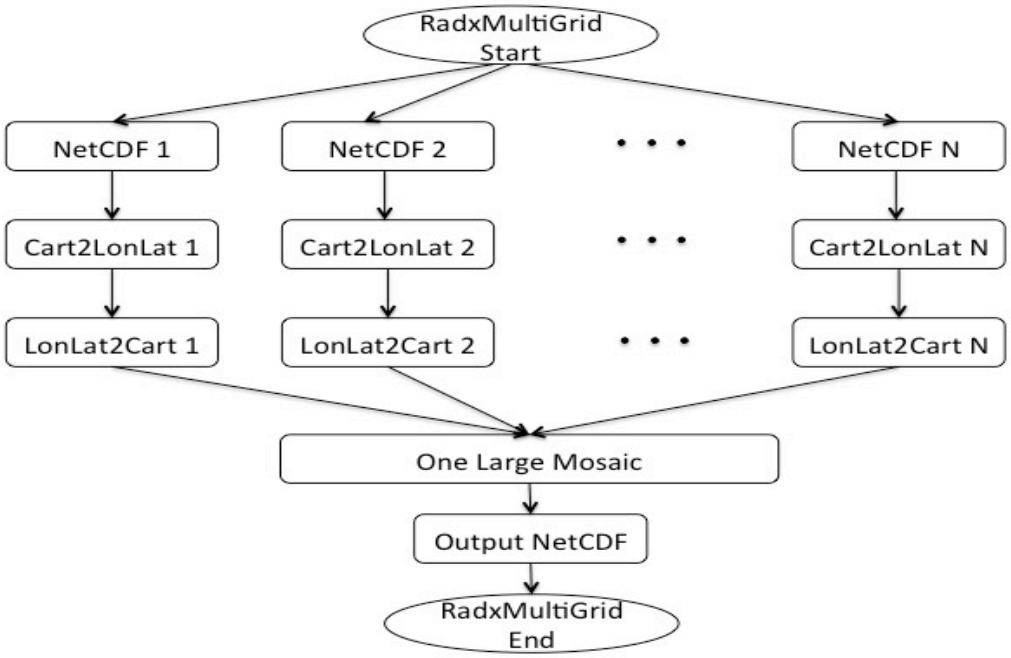

Figure 1: Mosaic Steps

2. Project GCS to a new PCS.
3. Interpolation
4. Write output NetCDF file for combined data.

It is designed as Figure 1.

2.1 Single Radar Grid: Azimuthal Equidistant Projection

When representing a single radar grid data on a map, we essentially establishes an Azimuthal Equidistant Projection (AEQD), where radar station is located in the $(0, 0)$. This is chosen on purpose because AEQD preserves distance along the great circle, essentially preserve true geometries for all radar beams. With such properties we can design a simpler interpolation scheme within single radar grid. To put this projection back to earth, we need first inverse calculate (lon, lat) of all cells in a single-radar grid. As explained in the Glossary, the GCS is a user's choice, which is defined by user's inputs. Figure 2 represent an illustration of the Azimuthal equidistant projection and some azimuthal projections.

Inverse transformation from AEQD to GCS is complex. It requires knowledge of geodesic measurements and often are not analytical. Thus we use a 3rd-party library called Proj4 to complete this task. Proj4 is able to project (GCS→PCS) and inverse project (PCS→GCS) coordinates with an easy interface. We will discuss Proj4 later. With Proj4, we are able to calculate longitude and latitude of all cells in the single-radar-centric grid.

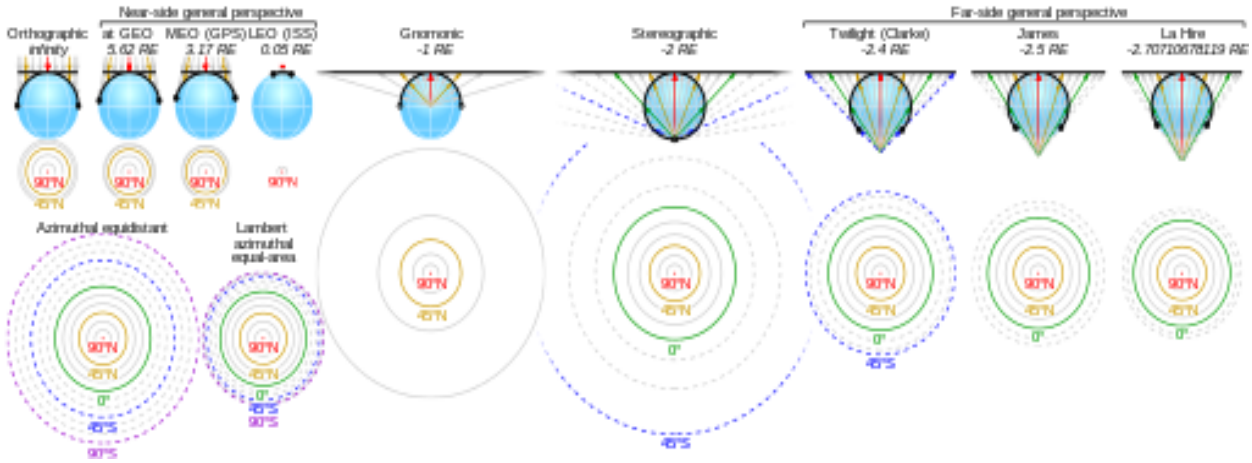

Figure 2: Comparison of the Azimuthal equidistant projection

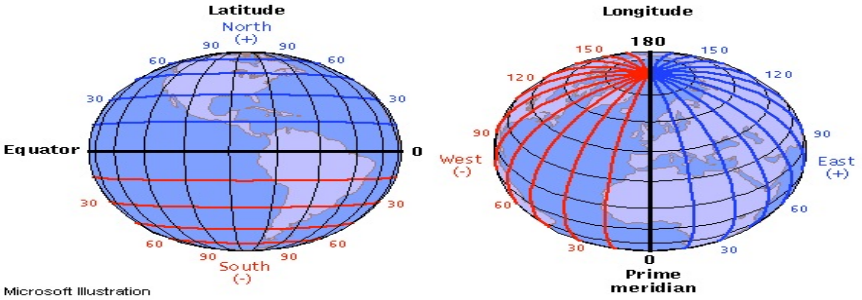

Figure 3: Latitude and Longitude

2.2 Multi-radar Grid

When we represent a large mosaic of multi-radar grid data, we take each cell from single-radar grid [in (lon, lat)], calculate its (x, y) location in the new domain, then find the final destination cell in the new mosaic grid. As we're transform data (lon, lat) into a Cartesian coordinate, we're doing a projection (GCS→PCS). Thus, a PCS definition must be provided by user (we may give a default option for user). To explain this procedure, in this document, we use Lambert Conformal Conic Projection as an example. In the application, user will provide such definition.

Consider in Fig 5, each circle is a bird-eye view of single radar mosaic. Note that since 0 degree azimuth always points to true north, thus, in a LCC projected map, all 0-deg-azimuth beam will virtually converge at the north pole. Thus, on a map, north probably doesn't point upright, and varies at different location. This is called geodesic convergence. One may ask why we choose LCC rather than simply use lon/lat as X/Y. It is because LCC preserves a circular shape of a radar scan, while in a lon/lat map (of course we can do so if it is defined so by users), such circular scan area **MUST** be distorted because 1km in high latitude area occupies more latitude units. To accurately draw a geometry of a circle in lon/lat Cartesian coordinates are difficult. (Fig. 4)

Please refer to `https://en.wikipedia.org/wiki/Map_projection#Conic` for more information about a conic projection, it represents a very common way to map curved earth surface to a plane.

NOTE: Following paragraph is different from what we have discussed during the meeting (Oct/17). Please

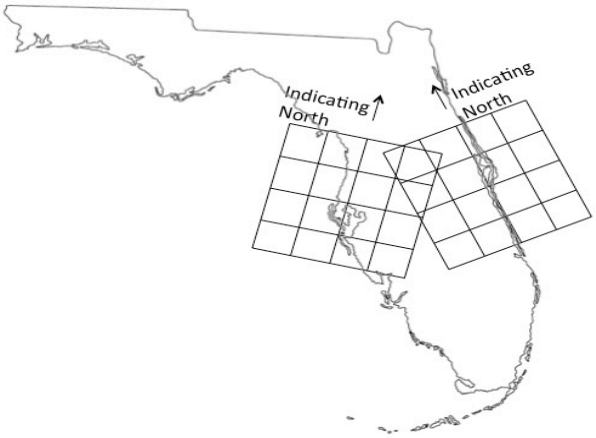

Figure 4: Represented after Convert to Latitude and Longitude

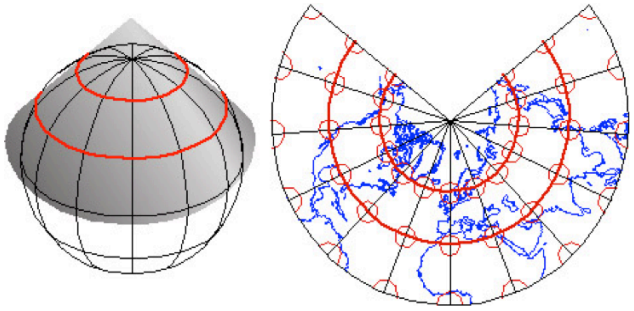

Figure 5: Lambert Conformal Conic Projection

read carefully.

In any PCS, there is always a (0,0) point defined, and it is always has normal X-Y coordinates. Thus, the mosaic domain will occupy a rectangular space on this X-Y coordinates. **It may not be centered at** $(0, 0)$**, because the** $(0, 0)$ **may locate anywhere**. For example, if a user chooses to use Web Mercator Coordinate System, $(0, 0)$ is located at 0 longitude and 0 latitude, while our research domain is in US, which is definitely not centered $(0, 0)$

The first step is to determine the extent of our mosaic domain on the earth, which presented in a four-number tuple $(min_x, min_y, max_x, max_y)$. We need ensure the mosaic domain . Although user inputs define the projection system, we determine the size of the array which presenting the mosaic domain, using given resolution (e.g. 1km, 0.5km, 0.25km, etc.) To calculate extent, for example, min_x, we get min_x from all radar grids. Note that if $lon_1 < lon_2$, we can not derive $x_1 < x_2$ after projection, You can easily find this by seeing the Figure 5. Thus, we need keep tracking min_x for all single-radar-grids. Same for all other three members.

After finishing this step, we actually obtained PCS coordinates for all cells from all single-radar-grids. The next step is to put them onto mosaic grid. Since we have had extent for the mosaic grid, thus it is trivial to transform an (x, y) to array index (i, j), using cell size and bottom-left coordinates

2.2.1 Handling Radar Intersections

A fact of radar network is radar coverages are overlapping. This ensures an continuous view over entire US. For example, radar KMLB and KTPL overlaps in central Florida area, as shown in Fig6. In this case, we are going to determine how the value in overlapped part can be calculated from the several values. A position may overlapped by several radars, but finally we only need one value. This is a normal reducing (aggregate function). Common aggregate functions include: max, min, mean, inverse-distance-weighted mean. For the range-gated mean, weight is determined by how far a pixel away from radar in its own single-radar-grids. For example, a mosaic cell is observable from two radars, A and B: Radar A grid has the cell at 100km away, Radar B grid has the cell at 10km away, Radar B is weighted more than Radar A. *A detailed function need to be designed later.*

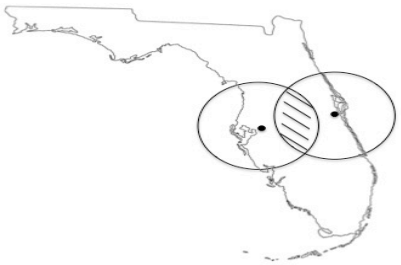

Figure 6: Radar overlap

3 Work flows

3.1 Step 1: Read NetCDF Files from the previous phase

When Radx2Grid is executed, it creates netCDF files that contain 3D Grid data. The netCDF files set up by the gridding process become the input of radar mosaic process and the 3D grid comprises Cartesian coordinates (x, y, z) for each gate. The first work for Mosaicking is to read these NetCDF files and load them on memory.

While the input NetCDF files are loaded into memory, a data structure which caches them is needed. We have devised one (called "layerInfo") that is consist of a set of vectors (vector<float>) that store the Cartesian coordinates, output values such as reflectivity, longitude, latitude and projected data(such as x,y coordinates) after applying Lambert Conformal Conic for each gate. As the Mosaic operates for each layer, the data structure stores complete information such as longitude and latitude and projected value of one layer and maintaining every layer in a Grid as a vector of the structure(vector<layerInfo>) during the Mosaic process.

To reduce the I/O delay, we are going to use a buffer queue. A thread is filling this buffer whenever it reads NetCDF file.

3.2 Step 2: Convert to Latitude and Longitude using Proj4

When the buffer queue is not empty, the grids loaded in the buffer are popped from the buffer. For mosaicking multiple radar grids on a map, we first need to convert the coordinates (x, y, z) to (longitude, latitude, z) for each radar.
A point on the globe is chosen to be special in the sense that mapped distances and azimuths from that point to any other point will be correct. That point, (φ_1 , λ_0), will project to the center of a circular projection, with φ referring to latitude and λ referring to longitude. All points along a given azimuth will project along a straight line from the center, and the angle θ that line subtends from the vertical is the azimuth angle. The distance from the center point to another projected point is given as ρ. By this description, then, the point on the globe specified by (θ, ρ) will be projected to Cartesian coordinates.

Since we're using Proj4 library here, we need define the projection string. In Proj4, such string is defined like this

```
"+proj=aeqd +lat_0=60 +lon_0=-90 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
```

"aeqd" defines the transformation method, which means "Azimuthal Equidistant Projection". lat_0 and lon_0 defines where (x_0, y_0) locates on the map. x_0, y_0 defines origin offsets, which is usually $(0, 0)$, unless in a zonal projection system like UTM. +ellps and +datum defines the GCS parameters. Common choice of is "+ellps=WGS84 +datum=WGS84" (for global map) or "+ellps=GRS80 +datum=NAD83" (for US map)

An easy way to find Proj4 definition string is to look it up at `http://spatialreference.org/`, then modify parameters. Full parameter list can be found here: `http://proj4.org/parameters.html`

Hence, as we reversely use Azimuthal Equidistant projection, we can convert the coordinates (x_0, y_0, z_0) of each cell in the grid to (longitude, latitude, z_0).
Once we convert the coordinates of one layer of one grid, we can reuse the coordinates information for the rest of the layers in the same grid. After this step is done, latitude and longitude of every cell in a grid are stored in the structure(vector<layerInfo>) and the radar data will be represented as Fig4 on a map. Converting the Cartesian coordinates to latitude and longitude does not need to be atomic operation and TBB will spawn a number of threads to handle this step.

3.3 Step 3: Project to The Mosaic Grid

Once we reach this step, we will have longitude and latitude for all points. We then use Proj4 again to project it the mosaic grid.

A mosaic grid's PCS may be defined like this:

```
"+proj=lcc +lat_1=20 +lat_2=60 +lat_0=40 +lon_0=-90 +x_0=500000.0 +y_0=0 +ellps=GRS80 +datum=NAD83
+units=m +no_defs"
```

Here is the explanation: +proj=lcc defines it is a Lambert Conformal Conic projection. In LCC, there are two secant latitude with earth, that is lat_1 and lat_2. Center of the projection defined at (lon_0, lat_0), which is (-90, 40). x_0 is false easting at 500000(m, defined in units). It means entire coordinate shifts (+500000, 0). You can image that all x are added by 500000 after LCC transformation.

It is IMPORTANT to keep ellps and datum same in Step 2 and Step 3. See Glossary.

We should provide a user an option to automatically determine an projection system from them. Here is the step:

1. Use +proj=lcc, +ellps=grs80, +datum=nad83, and other options from example above

2. Find max and min of latitude of all radars, define 1/3 and 2/3 point between $[min_{lat} - 2, max_{lat} + 2]$, these are lat_1 and lat_2. Also define $lat\_0 = (min + max)/2$
3. Find max and min of longitude of all radars, define $lon\_0 = (min + max)/2$. Now we essentially put the mosaic domain as small as possible and centered at where our radars are cover.

When converting latitude and longitude to (x, y) coordinates for each grid, TBB can generate threads to calculate the Cartesian coordinates.

3.3.1 Calculate the final value

Now, we need to calculate the actual value of each cell of the domain. Consider we want to fill the value in the cell Fig8. Then, we need to search the cells that surround the target and get values and calculate the distance, Fig9. There are many ways to calculate and fill the value in the final mosaic cell. We should consider calculating and filling the mosaic cell needs to be atomic.

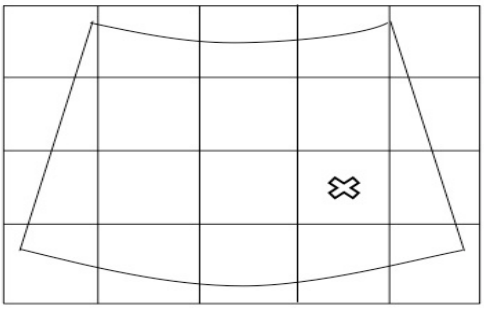

Figure 7: Target cell

Nearest one This is the fastest but less accurate way to fill the value. We can fill with the value which is the shortest distance among surrounded cells.

Max or Min Among the values, we can fill with Maximum or Minimum value.

Mean We can calculate the mean of every value and fill it to the final cell.

Linear interpolation We can weigh the value to fill. Then the final value can be the sum of value divided by distance divided by the inverse of the sum of distance.

$$V = \frac{\sum \frac{v_i}{d_i}}{\sum \frac{1}{d_i}}$$

Bicubic Using this method, we can represent the final mosaic the most smoothly. However, it also the slowest one.

$$V = \frac{\sum \frac{v_i}{d_i^3}}{\sum \frac{1}{d_i^3}}$$

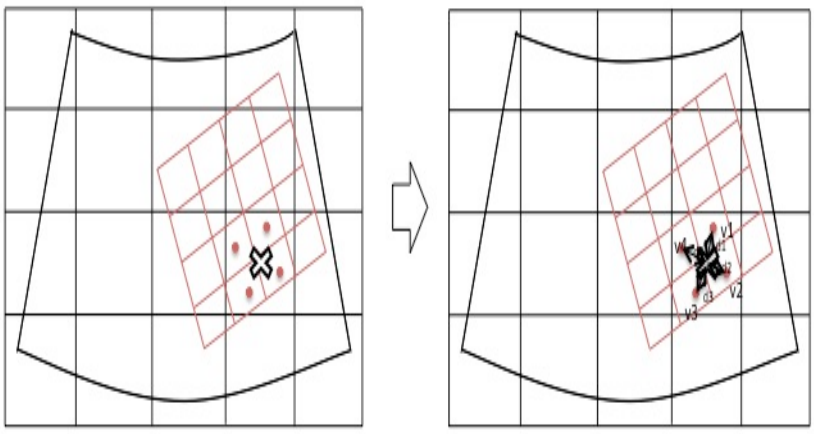

Figure 8: Get Surrounded Values

3.4 Step 4: Write Output NetCDF

We need to write the mosaic output from the previous step into one output file. For the output file, we create the NetCDF file as an output for mosaic process.

4 Glossary

- Geographic Coordinate System (GCS): a geographic coordinate system defines a point of the Earth using longitude, latitude and altitude above mean-sea-level. A GCS contains three components: a ellipsoid model describe earth with major and minor axis, a geodesic datum establishes reference points with defined longitude and latitude, and a geoid describes earth surface. A GCS is a model to measure longitude and latitude of a point of the earth. Since different GCS uses different parameters in three components, a point on the earth may have different longitudes and latitudes in different GCS. There in no "correct" GCS in the world, GCS are chosen to minimize local error. For a given (lon, lat) in U.S., one should expect an "error" range about 50–500 meters across common GCS. Further reading: `https://en.wikipedia.org/wiki/Geodetic_datum`; `https://en.wikipedia.org/wiki/Reference_ellipsoid`; `https://en.wikipedia.org/wiki/World_Geodetic_System`
- Projected Coordinate System (PCS): a projected coordinate system defines a point on the Earth through a map in 2D Cartesian coordinate system. Commonly, we usually call a PCS as a "projection". Altitude is generally omitted, because it describes sea surface using 2D plane. There are two components in a PCS: a GCS defines longitude and latitude, and a transformation method. Every (lon, lat) in GCS, through the transformation method, will be **projected** to (x, y) position. The projected figure is a "map". Since GCS is a user's choice, the final location. **It is very important to notice that, there is NO PCS can completely preserve original geometry of the earth surface. Any PCS can only preserve at most one characters: 1. shape (circle, polygon, etc.); 2. area size (hold same size, but shape must change); 3. distance along the great**

circle (only the distance along the great circle). Correspondingly, we call them: 1. conformal projection, 2. equal-area projection, and 3. equidistant projection. A PCS may preserve none of them.