MISR Time Averaged Project Report
Author: Nurul Atiqah Hamzah
Contact: hamzah3@illinois.edu

I was introduced to the climate marble project when I started working in late June and used the month of July to ramp-up and get myself familiar with all the hdf datasets and the APIs associated with it. Once I was comfortable with producing images for the MISR datasets, I then officially started to work on the time averaged MISR data in August. The summarized timeline of this project is provided below.

Timeline of the MISR time averaged project

Month	Progress
August	• Attempt different approaches such as a single numpy 2d grid and different approaches for re-gridding for the latitude and longitude data • Decided to stick to the approach using pandas dataframe for the grid • Chose nearest-neighbour gridding for the latitude and longitude data
September	• Attempted to work with basemap to produce images for verification of data produced using grid block • Tried averaging the radiance data using the original pixel positions and verified them but decided against approach because it wouldn't work for a larger scale of the MISR data
October	• Refactored code by changing gridding back to a regular division grid instead of nearest neighbor • Changed the grid block code to do all the radiance data extraction automatically after calling insertion rather than having the insertion done with radiance data processed outside of the grid block class • Fixed some errors with radiance data extraction (getting rid of the smallest 2 bits, added in solar zenith data extraction)
November	• Wrote wrapper for one month's worth of data parsing • Tested parsing on single hdf file on ROGER on qsub interactive • Tested insertion for a month's worth of data with qsub

Approach
A pandas dataframe in which each row represents a pixel and the fixed columns are latitude, longitude, path, red radiance, green radiance, blue radiance and the solar zenith that is mapped to the pixel would be used to represent the grid. This is done through the GridBlock class in the grid_block.py file which is using geo_grid.py to regrid the latitude and longitude data from an agp file to the desired grid width and uses misr_tools.py to extract the misr data and sort it for insertion into the grid dataframe. The GridBlock class contains two dataframes, grid dataframe and geolocation dataframe, to keep the radiance data and the latitude and longitude data respectively.

The initialization process flow is as below:

1. Declare an instance of the GridBlock class and pass in desired grid width in degrees [grid_block.py]
2. Initialize the geolocation dataframe by calling GridBlock class method init_geolocation_database
3. Pass in the MISR AGP file associated with the MISR radiance file into insert_misr_geolocation (GridBlock class method). The method will call build_agp_dataset method within geo_grid.py to process the agp data into the grid width. Then, GridBlock will insert the agp data into its geolocation dataframe.
4. Call insert_misr_file (GridBlock class method) with the MISR radiance data file. The GridBlock class will then process the radiance data and realign it with the geolocation data kept in the geolocation dataframe and insert the data, indexed by pixel per row, into the grid dataframe.

Since the agp geolocation data is already cached into the geolocation dataframe, you can continuously feed the grid with MISR files with the same path via step 4. If you wish to average the data by their latitude and longitude as a pivot, call the aggr function in GridBlock and the output data will average every pixel with other pixels with the same latitude and longitude.

Why this approach?
The approach is chosen to enable some flexibility in how we would average the data. By enabling separate processing of each MISR hdf file, I could reduce the amount of computation done. Averaging could also be carried out after every MISR hdf file is inserted into the dataframe. With pandas dataframes, aggregation could use any of the columns as their ‘pivot’ in which the data could be averaged based on their column values, allowing for averaging not only through the geolocation but also using monthly, yearly or any other column in which could be added to the dataframe easily. I could also easily get rid of the fill data by initially replacing the fill data with np.Nan and then having the dataframe delete all rows containing np.Nan values. I also thought that if we could populate the grid dataframe with unaveraged data, we would have a base value dataframe in which we can carry out different types of averaging and save the results separately without having to perform the computations on the hdf files again. However, the cons of this approach are that there a lot of overhead in keeping the column data for

pandas and with every added hdf file, the computations get slower. The grid would also increase with size and introduce scaling problems. Even with enough computation power, memory allocation is still an issue.

Problems Encountered and Possible Solutions

- I tried averaging the misr data based on initial pixel location and found out that the data is misaligned from any hdf file to other hdf files. The misalignment is usually a small number of pixels and exist in both the vertical and horizontal axis. Perhaps this is producing the visible diagonal lines in the output globe image. A viable solution is to use a single block image as a base and correct the alignment of every other image to this base images alignment by shifting the image in both the vertical and horizontal axis by a few pixels. However, this might be problematic because the output averaged data may be misaligned with pixel based latitude and longitude data itself. So perhaps it would better to use the data within the agp files as a solid base and realign every other block image data to the block agp data instead.
- Processing the agp data first and saving it somewhere would reduce overall processing time since an agp file can be mapped to many other MISR files.

Summary

Overall, many approaches were attempted but the pandas dataframe is chosen for its flexibility. However, this approach also produced many problems in verifying the correctness of the averaged radiance output and scaling up with increased insertion of radiance data. For faster and accurate producible results, it would better to not use the pandas dataframe. Perhaps in the future, an alternative that would produce less overhead memory usage than pandas could be used instead.