ESE90: Progress Report

Edward Speer
CliMA Land Team
FA '24

December 11, 2024

1 Overview

Under ESE90, I worked for the CliMA land team under the supervision of Dr. Renato Braghiere and the rest of the team. The goal of the CliMA Land project is to build a process-based land surface model in a high level programming language, with an unprecedented level of modularity, and flexibility. The CliMA Land model is designed to fit together with CliMA's ocean and atmosphere models to form a comprehensive Earth System Model. A large part of CliMA's mission is to bring modern data science and machine learning techniques into the ream of process-based earth modeling and to make the models more accessible to a wider range of users, from students to researchers to policy makers.

Previously, I have worked on several aspects of the Land Surface model, including the canopy radiative transfer scheme and the site level validation of the model on flux tower data. In this report, I will discuss the areas of the model where I made new progress in the Fall term, being the software pipeline for global calibration, and the use of well documented, replicable artifacts In the data processing pipelines of the model.

2 Global Calibration

Global calibration of the CliMA land model is a crucial step in the model development process. The CliMA Land model has a very large number of parameters that need to be specified in each grid cell over the Earth's surface for the model to run. While some of these parameters have large datasets from empirical studies available, many of the parameters do not, and are hard to estimate. The global calibration process is a way to estimate these parameters using machine learning techniques. This term, I worked with the team to apply one such machine learning technique, Ensemble Kalman Filtering, to the global calibration of the bucket hydrology model in the CliMA Land model. This work is a major step in the ongiong effort to perform calibration of the full land model, and put in place a software framework which will make bringing the full model calibration online much easier.

The implementation of Ensemble Kalman Filtering used is an existing CliMA package called `EnsembleKalmanProcesses.jl`. This package requires a certain interface to be implemented to allow the model calibration. The basic interface requirements we implemented are:

- Vector of parameters - The parameters to be calibrated are stored in a single vector which is passed to the model.
- Output/Observation vector - The model must return a vector of outputs which correspond to a vector of stacked observations. This is used to compare the model output to the observations.
- Function - There must be a single function which takes in the vector of parameters and returns the output vector, wtacked properly for comparison with the observations.

Our initial pipeline attempted to learn the following parameters:

- κ`_soil` - Soil hydraulic conductivity
- ρ`c_soil` - Soil volumetric heat capacity
- `f_bucket` - Fraction of bucket capacity at which evaporation begins
- `W_f` - Capacity of the land bucket
- `p` - Exponent used in beta decay
- `z_0m` - Roughness length for momentum

by using the following set of output variables::

- `shf` - Sensible heat flux from the land surface
- `lhf` - Latent heat flux from the land surface

For comparing our model output to observations, I built a pipeline which selects random latitude/longitude pairs on the land surface, read in ERA5 reanalysis data for the selected points, and then compares the model output to the ERA5 data. ERA5 data is a high resolution reanalysis dataset produced by the European Centre for Medium-Range Weather Forecasts (ECMWF). The ERA5 data is used as a proxy for the true land surface fluxes, and the model output is compared to the ERA5 data to determine how well the model is performing on any given iteration of the calibration process.

To select random land locations, I first read in the Land/Sea mask used in CliMA Land model, reshaping it to the same resolution as out global calibration grid. I then collect all of the lat/lon pairs of the land points into a Julia array and randomly select a subset of these points to use in the calibration process.

The ERA5 data is stored in NetCDF files, and I used the Julia package `NCDatasets.jl` to read in the data. The ERA5 data is stored as a length 12 array at each grid point, with each element of the array corresponding to an average taken over a different month. I used the `Dates.jl` package to select data for the appropriate time period matching the driver data used for the simulation, and then select the appropriate grid points closes to the random points selected for the calibration.

Once the locations are selected, the ERA5 data is read in, and the model is set up, the calibration process is placed in the hands of the Ensemble Kalman package. The Ensemble Kalman package

uses a set of ensemble members to estimate the parameters of the model. The ensemble members are generated by perturbing the initial guess of the parameters (the priors), and running the model with the perturbed parameters. The model output is then compared to the observations, and the ensemble members are updated based on the comparison. The ensemble members are then used to update the priors, and the process is repeated until the parameters converge to a set of values which best fit the observations or until the maximum number of iterations is reached. This process is run on GPUs on the Caltech HPC cluster, and results are stored and reported using the `ClimaDiagnostics` package.

This calibration process requires running the model many times with many sets of perturbed parameters, and so it is quite computationally expensive and time consuming. Future improvements may involve running the calibration process on multiple GPUs in parallel to speed the process. If the calibration were made faster, a higher number of ensembles and more iterations could be used on a greater number of parameters, which could significantly improve the accuracy of the calibrated model. Before this happens, however, the main priority is to extend this pipeline to the calibration of the full Land model, which will require slightly more complex data processing as well as the prescription of spatially varying parameters over the land surface within the calibration pipeline.

3 Replicable Artifacts

The CliMA Land model consumes a large amount of data from a number of different sources using a number of different methods of data processing. This data is used to drive the model, to prescribe parameters to the model, to compare the model output to observations, and to calibrate the model. It became clear a while ago that there was a need to standardize and document the data processing pipelines used by the model, and to centralize the storage of the various data artifacts handled by the model. For this purpose, `ClimaArtifacts.jl` was created. This term, I worked with the team to move a lot of the artifacts used in CliMA Land into `ClimaArtifacts`, and to document the data processing pipelines used to generate these artifacts.

The first artifact I converted to a `ClimaArtifact` was the foliage clumping index data derived from MODIS data. The foliage clumping index is a measure of the degree to which the foliage in a canopy is clumped together, and is used in the canopy radiative transfer scheme in the CliMA Land model. MODIS is a remote sensing satellite which provides a number of different data products including the foliage clumping index.

MODIS data is stored in NetCDF files, but is stored in a sinusoidal projection. This sinusoidally projected data is not directly usable in CliMA global runs which use a rectilinear grid. Luckily, there is another CliMA package called `GriddingMachine.jl` which offers a processed version of the MODIS data in a rectilinear grid. I used this package to read in the MODIS data, and then converted it to a `ClimaArtifact`, and downloaded the artifact to the Caltech HPC cluster where it may be used by any CliMA model.

The second artifact I converted to a `ClimaArtifact` was the fluxtower data used for site level validation of the Clima Land model. The fluxtower data is stored in large CSV files for each site, which contain a number of variables used to both drive, prescribe site-level parameters to, and validate the model. This data is downloaded from the Ameriflux website. To generate the data

artifacts, I wrote a simple Julia script which trims the CSV files down to the desired year (2010). I then converted the data to a `ClimaArtifact` and uploaded it to the Caltech HPC cluster.

The final artifact I converted to a `ClimaArtifact` was a test dataset I had previously generated to test my TwoStream radiative transfer scheme implementation. This dataset is a simple CSV file containing sets of input parameters and the corresponding fraction of absorbed photosynthetically active radiation (fAPAR) output by the TwoStream implementatiom. This expected output is generated from a reference TwoStream implementation, PySellersTwoStream written in Python2 by Tristan Quaife. To generate the artifact, I first forked the PySellersTwoStream repository, and then updated it to run under Python3. I then wrote a Python3 script which runs the code in the fork over a large combinatorial space of input parameters, and then saves the output to a CSV file. I then wrote a Julia script which calls the Python3 script and then saves the generated CSV outputs to a `ClimaArtifact`.

The conversion of these artifacts to `ClimaArtifacts` is a major step in the ongoing effort to standardize and document the data processing pipelines of the CliMA Land model. It is important to undertake these efforts in order to ensure that the model and data artifacts used are easliy reproducible and accessible, and that all data is being used and shared in accordance with license agreements.

4 Pull Requests

Here are the pull requests documenting the details and code for all of the changes I made to CliMA repos this term:

- `ClimaLand`:
  - All calibration work - #835
  - Incorporate clumping index `ClimaArtifact` #863
  - Incorporate TwoStr test data `ClimaArtifact` #910
  - Incorporate fluxtower data `ClimaArtifact` #914
- `ClimaArtifacts`:
  - Create clumping index artifact: #52
  - Create TwoStr test data artifact: #62
  - Create fluxtower data artifact: #65