PyEnsemble: A Python Ensemble Manager for the GEOS-Chem Adjoint Model

Andre Perkins[1], Dr. Daven Henze[2]

1. SOARS, University of Wisconsin-Madison

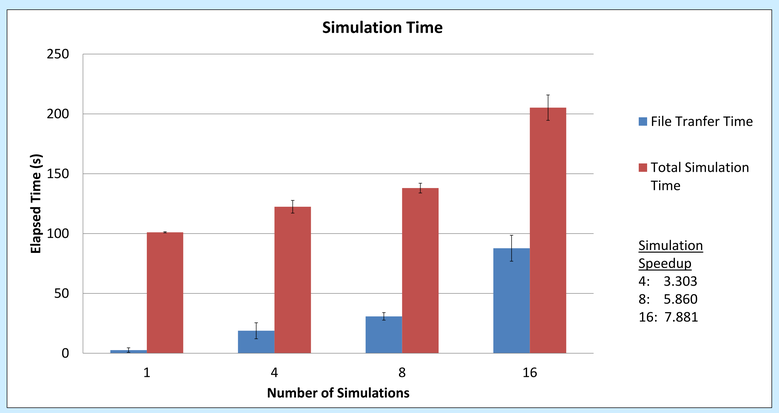
2. University of Colorado-Boulder

Project Motivation

The GEOS-Chem Adjoint model , developed using the OpenMP framework, allows for a single simulation to be executed per node on a distributed-memory computing cluster. Most of today's high-performance computing resources can have anywhere from tens to thousands of these nodes available for use in a job. However, having to manually set up individual simulations can make it impossible to fully utilize a large percentage of the resources available on such a system. Therefore, to make large ensembles of GEOS-Chem simulations feasible for projects the PyEnsemble tool was created.

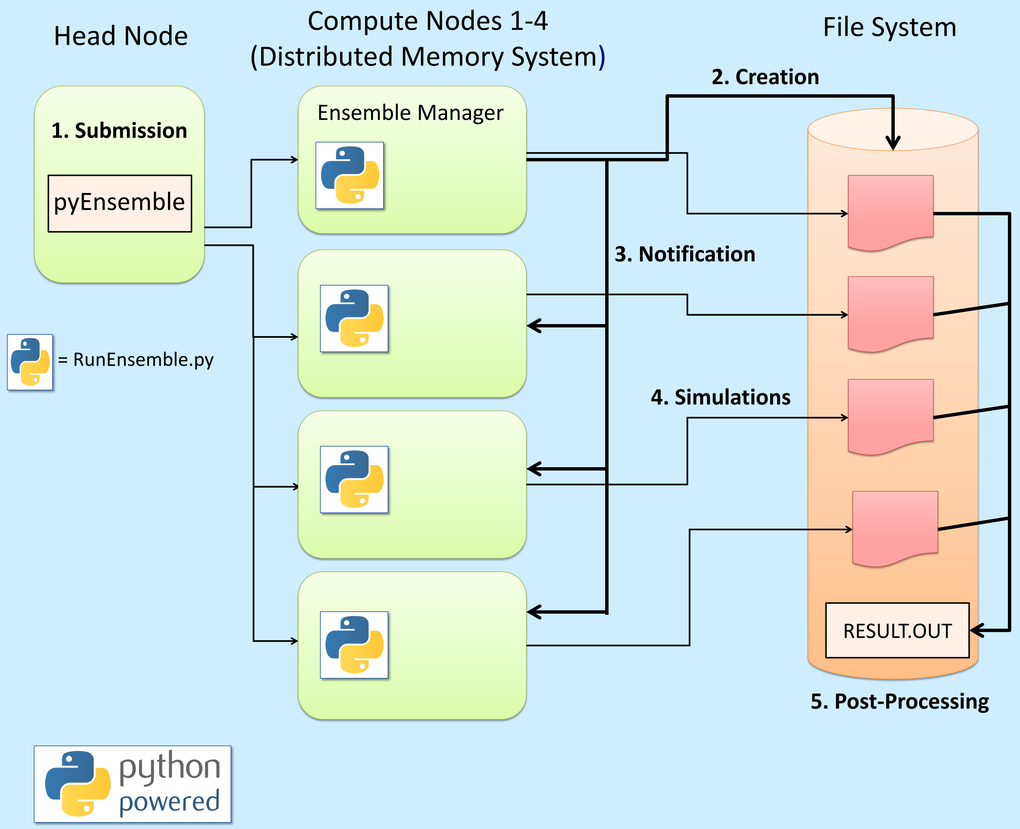
PyEnsemble Performance

- Successful 16 simulation ensemble with post-processing
- Successful 100 simulation ensemble using test simulations
- Storage space can be used up quickly
  - A single GEOS-Chem Adjoint model simulation can generate gigabytes of temporary data
- Current speed bottleneck related to file transfers of directory creation
  - Could take advantage of parallel file I/O on capable systems
- Degraded speedup displayed below is a result of short test simulation time (1-day duration)
  - Normal simulation time takes hours, this gives much better speedup



This graph shows total elapsed time for different ensembles of simulations averaged over five ensemble runs. Blue shows the time spent creating the directory structure. Red depicts the total ensemble run time. Error bars represent one standard deviation from the mean. Speedup values give the degree to which the ensemble run speeds up the process compared to sequential runs.

Python Ensemble Manager



PyEnsemble Execution Flowchart

1. User submits PyEnsemble batch script to the computer queuing system. This script specifies the location of the desired simulation and number of simulations to run.
2. RunEnsemble.py is distributed across the specified computing nodes, designating a single node as the 'Ensemble Manager.' The manager copies a simulation directory for each node, along with generating unique input files for each directory.
3. The manager node notifies other nodes of respective run directory locations.
4. Each node performs a GEOS-Chem Adjoint simulation.
5. Nodes send data back to the manager node, whereupon desired output is compiled and written to a file.

Implementation Details

- Developed using Python v2.7.3
- Library utilization
  - Standard Python libraries: shutil, os, logging, etc.
  - MPI4py facilitates communication across computing nodes
  - Numpy is used for array handling and matrix calculations in the input file generator and post-processing stage
- Main script components: RunEnsemble.py, InputGen.py, PostProcess.py
- Modular nature to allow for easy removal or editing of components

Future Work

- Provide a Python virtual environment to run PyEnsemble
  - No need for admin privileges to install and use the scripts
  - Prevents newer or older versions of the libraries from affecting program execution
- Test PyEnsemble on larger computing clusters
- Upload scripts and documentation to GitHub for distribution

Acknowledgements

I would like to thank my science mentor Daven Henze, and his group at CU-Boulder for their help and support with this project, I would like to thank my writing mentor Jana Milford for her thoughtful reviews of my work, and I would like to thank UCAR and SOARS for providing the opportunity to perform this summer research.

References

Bey, I., and Coauthors, 2001: Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation. Journal of Geophysical Research-Atmospheres, 106, 23073-23095.

Henze, D. K., A. Hakami, and J. H. Seinfeld, 2007: Development of the adjoint of GEOS-Chem. Atmospheric Chemistry and Physics, 7, 2413-2433.

```
#Run simulation code example
try:
    logger.debug('Starting simulation on %s for %s' % (name, dir) )
    #same as typing command run > log into terminal
    os.system('%s > %s' % (os.path.join(dir, 'run'),
         os.path.join(dir,'log') ) )

except OSError, e:
    logger.error("Failed to start simulation on %s: %s" % (name, e) )
```