Working with the High-Performance GEOS-Chem (GCHP)

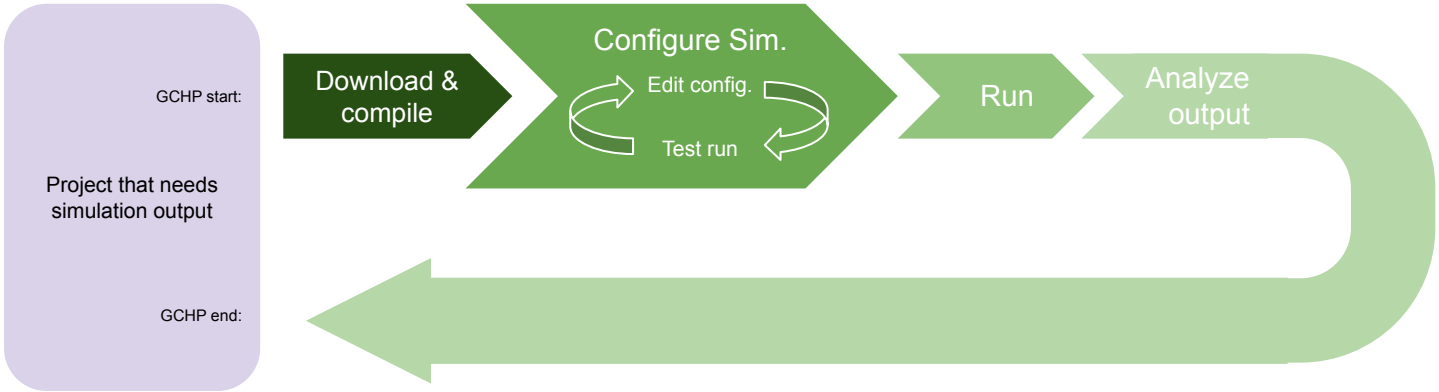
10th International GEOS-Chem Meeting (IGC10)

2022-06-06

Liam Bindle

Video credit: Dandan Zhang (WUSTL)

Typical GCHP Workflow



This clinic has a strong emphasis on **understanding** how to use GCHP.

Focus on intention rather than example code or commands.

Step indicator: B C R O

Overview of Clinic

Level 1 – Conceptual Overview

- Working on an HPC
- Conceptual overview of workflow

Level 2 – Basic Usage

- Barebones overview of how to use GCHP

Level 3 – Practical Usage

- Build-time options
- Configuration files
- Run scripts

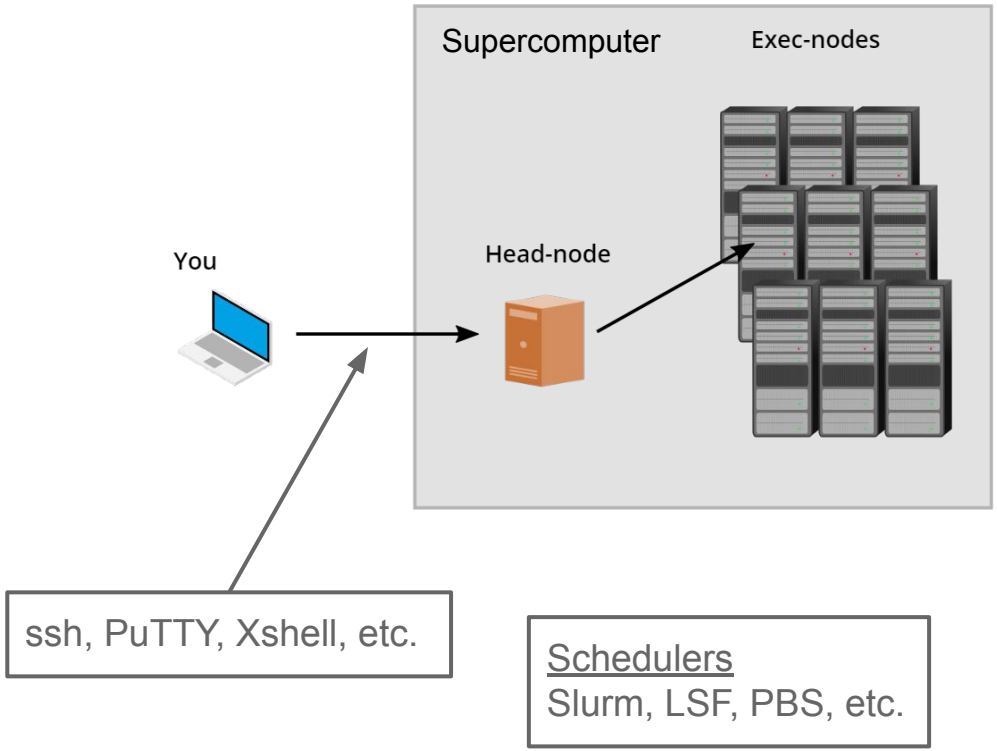
Level 1. Conceptual Overview

How to do work on an HPC

Conceptual overview of how to run GCHP

HPCs are **all different** but kind of **the same**

- HPC ⇔ cluster ⇔ supercomputer
- Log in to head-node via SSH
- Submit *job* to scheduler to do work
- *job* = script executed on exec-nodes



Things you are going to need

- Access to a cluster

```
~$ ssh lbindle@compute.foobar.edu
```

- Software (git, make, cmake, compilers, MPI, NetCDF, ESMF) ⇔ *environment*

```
~$ module load dev-utils
Loaded 'dev-utils' software bundle
~$
```

or

```
~$ source ~/geoschem_deps-2022.03
~$
```

- GEOS-Chem input data (metfields, emissions data, data for chemistry parameterizations, etc.)

Compiling

- Compiler converts source code (Fortran) into an executable program

- Build system = script of compiler commands that compile GCHP source code into program

- cmake – used to configure GCHP build
- make – used to execute GCHP build

```
~/build$ cmake <options> ../path/to/GCHP
```

```
~/build$ make
```

Creating + configure a run directory

- Run directory – config files + output files
- createRunDir.sh – creates a run directory
- Edit the config files in run directory to configure your simulation

```
~/GCHP/run$ ./createRunDir.sh
```

```
~/my-rundir$ vim runConfig.sh
```

or

```
~/my-rundir$ emacs runConfig.sh
```

```
~/my-rundir$ ./runConfig.sh
```

Running

- Parallelized with MPI
- mpirun (or equivalent) – used to launch *n* instances of GCHP program
- Instances are launched on other exec-nodes too

```
~/my-rundir$ mpiexec <options> -n 720 ./gchp
```

or

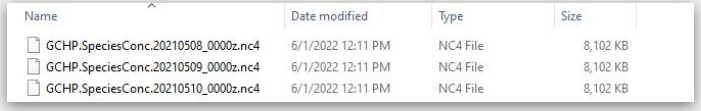
```
~/my-rundir$ mpirun <options> -n 720 ./gchp
```

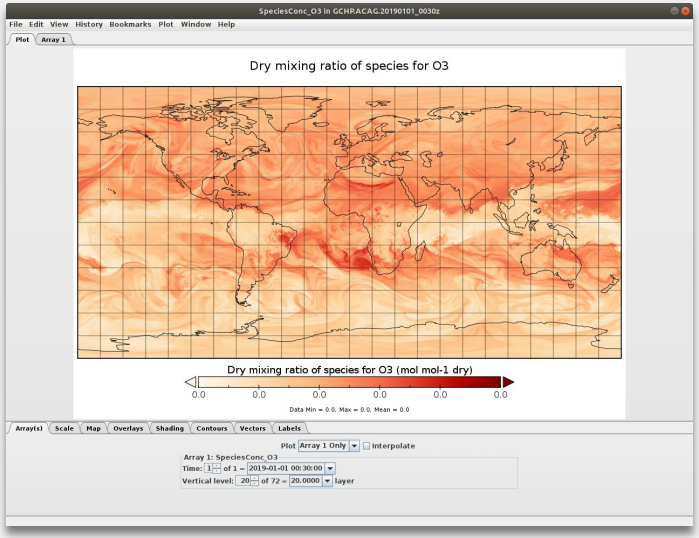
or

```
~/my-rundir$ srun <options> -n 720 ./gchp
```

Working with output data

- NetCDF files are written to OutputDir
- Make plots with Panoply, Python, Matlab, etc.
- Outputs configured in HISTORY.rc





Important directories (folders)

- Run directory
- Build directory
- ExtData

Level 2. Basic Usage

What's needed to run a GCHP simulation

How to configure basic settings

Downloading the source code

- Code available on GitHub
- Use specific GEOS-Chem version
- Uses submodules to include other code like the geos-chem source code

```
~$ git clone https://github.com/geoschem/GCHP
~$ cd GCHP
~/GCHP$ git checkout 13.4.2
~/GCHP$ git submodule update --init --recursive
```

Building GCHP

- Build directory – working folder for a build
- Run cmake in an empty directory ⇒ build directory
- CMake will give an error if there is a problem with your environment

```
~$ source ~/geoschem_deps-2022.03
~$
```

```
~$ mkdir build-dir
~$ cd build-dir
~/build-dir$ cmake ../GCHP
~/build-dir$ make
~/build-dir$ make install
```

Create + configure your run directory

- Run directory = working directory for a simulation
- createRunDir.sh makes a new run directory
- runConfig.sh – common settings

```
~$ cd GCHP/run
~GCHP/run$ ./createRunDir.sh
```

```
~/my-rundir$ vim runConfig.sh
```

or

```
~/my-rundir$ emacs runConfig.sh
```

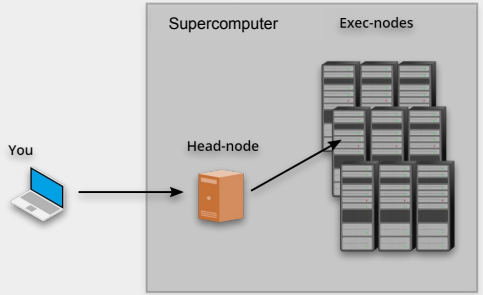
```
~/my-rundir$ ./runConfig.sh   # applies configuration
```

runConfig.sh

- Compute resources
- Grid configuration
- Start date, end date
- Restart file (initial conditions)
- Checkpointing

```
#!/bin/bash

...

TOTAL_CORES=96
NUM_NODES=4
NUM_CORES_PER_NODE=24

...

CS_RES=90
STRETCH_GRID=OFF
STRETCH_FACTOR=2.0
TARGET_LAT=-45.0
TARGET_LON=170.0

...

Start_Time="20220401 000000"
End_Time=   "20220405 000000"
Duration=   "00000004 000000"

...

INITIAL_RESTART=my_restart.nc4

...

Periodic_Checkpoint=OFF
Checkpoint_Freq="1680000"
Checkpoint_Ref_Date=START
Checkpoint_Ref_Time=START

...
```

Running GCHP



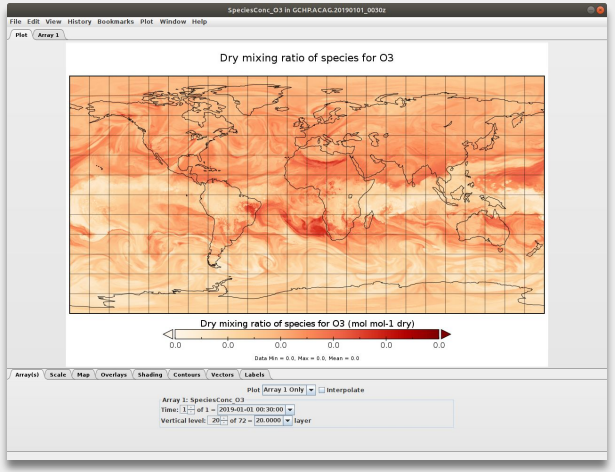
- Should be done on **exec-nodes**
- **Don’t forget** to:
  (1) load software, (2) execute ‘runConfig.sh’
- Compute resources need to be **consistent**:
  job, runConfig.sh, mpirun

```
~$ source ~/geoschem_deps-2022.03
~$
```

```
~/my-rundir$ ./runConfig.sh  # apply configuration
~/my-rundir$ mpirun -n 90 ./gchp
```

Working with output data

- Panoply is convenient for quickly viewing output
- For Python, use pcolormesh() with grid corners
- For Matlab, use pcolorm() with grid corners (via corner_lats and corner_lons)




```python
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import xarray as xr

ds = xr.open_dataset('GCHP.SpeciesConc.20210508_0000z.nc4')

plt.figure()
ax = plt.axes(projection=ccrs.EqualEarth())
ax.coastlines()
ax.set_global()

norm = plt.Normalize(1e-8, 7e-8)

for face in range(6):
    x = ds.corner_lons.isel(nf=face)
    y = ds.corner_lats.isel(nf=face)
    v = ds.SpeciesConc_O3.isel(time=0, lev=23, nf=face)
    ax.pcolormesh(x, y, v, norm=norm, transform=ccrs.PlateCarree())

plt.show()
```

Level 3. Practical Usage

How to use GCHP effectively for your work

Overview of configuration

Job scripts (run scripts)

Downloading source code

- Already covered

```
~$ git clone https://github.com/geoschem/GCHP
~$ cd GCHP
~/GCHP$ git checkout 13.4.2
~/GCHP$ git submodule update --init --recursive
```

Building GCHP

- Build settings configured with `-D<NAME>=”<VALUE>”`

- Build settings:

RUNDIR	OMP
APM	TOMAS
RRTMG	LUO_WETDEP
GTMM	

- Can’t find software? Use `CMAKE_PREFIX_PATH`

First time:

```
~/build-dir$ cmake ../path/to/GCHP
```

Subsequent calls, e.g.:

```
~/build-dir$ cmake . -DRUNDIR=/path/to/rundir
```

```
~/build-dir$ cmake . -DRUNDIR=”/path/to/rundir1;/path/to/rundir2”
```

Creating + configuring a run directory

- Already covered run directory creation

```
~$ cd GCHP/run
~GCHP/run$ ./createRunDir.sh
```

- Most useful config files:

`runConfig.sh`	Common settings
`HISTORY.rc`	Output files
`ExtData.rc`	Input data
`HEMCO_Config.rc`	Emission parameterizations
`logging.yaml`	Log messages

```
~GCHP/run$ vim runConfig.sh
```

```
~GCHP/run$ vim HISTORY.rc
```

```
~GCHP/run$ vim ExtData.rc
```

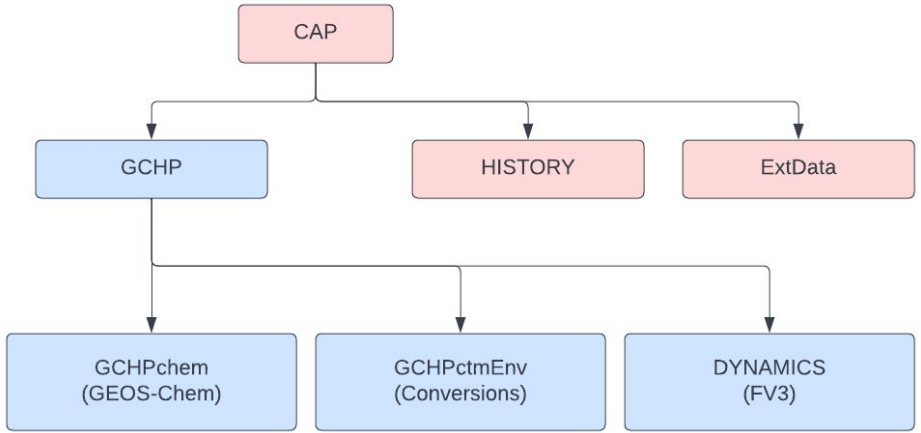
```
~GCHP/run$ vim HEMCO_Config.rc
```

```
~GCHP/run$ vim logging.yaml
```

GCHP structure

- ExtData – Input data (ExtData.rc)
- HISTORY – Outputs (HISTORY.rc)
- GCHP – GCHP model (GCHP.rc)

Structure of GCHP
(Gridded Components)

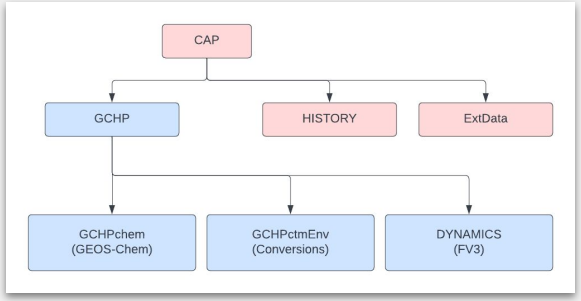


HISTORY.rc

- Described in detail on ReadTheDocs
- Outputs can be on a lat-lon grid
- Custom collections are *very* useful

```
GRID_LABELS:   MY_CUSTOM_GRID
::

MY_CUSTOM_GRID.GRID_TYPE:  LatLon
MY_CUSTOM_GRID.IM_WORLD:   360
MY_CUSTOM_GRID.JM_WORLD:   181
MY_CUSTOM_GRID.POLE:       PC
MY_CUSTOM_GRID.DATELINE:   DC


COLLECTIONS:   'CustomCollection',
::

CustomCollection.template:    '%y4%m2%d2_%h2%n2z.nc4',
CustomCollection.format:      'CFIO',
CustomCollection.frequency:   010000
CustomCollection.duration:    240000
CustomCollection.grid_label:  MY_CUSTOM_GRID
CustomCollection.mode:        'time-averaged'
CustomCollection.fields:      'SpeciesConc_O3   ',   'GCHPchem',
                              'SpeciesConc_NO   ',   'GCHPchem',
                              'SpeciesConc_NO2  ',   'GCHPchem',
                              'Met_BXHEIGHT     ',   'GCHPchem',
                              'Met_AIRDEN       ',   'GCHPchem',
                              'Met_AD           ',   'GCHPchem',
::
```

logging.yaml



- **Useful for troubleshooting** your simulation
- Levels: ERROR, WARNING, INFO, DEBUG
- root_level: messages only printed once
- See `allPEs.log`

```
...
loggers:

   CAP:
       level: WARNING
       root_level: INFO
   MAPL:
       handlers: [mpi_shared]
       level: WARNING
       root_level: INFO
   CAP.EXTDATA:
       handlers: [mpi_shared]
       level: WARNING
       root_level: WARNING
       propagate: false
   GCHPctmEnv:
       handlers: [mpi_shared]
       level: WARNING
       root_level: INFO
   GCHP:
       handlers: [mpi_shared]
       level: WARNING
       root_level: INFO
```

Running GCHP

- Run simulations in segments that take ~1 day (via `Duration` in `runConfig.sh`)
- `cap_restart` defines the start time
- GCHPchem_INTERNAL_RESTART_FILE defines restart file
- Always* used a job script!

*running interactively is useful when you are configuring or troubleshooting a simulation

```
~/my-rundir$ ./runConfig.sh  # apply configuration
~/my-rundir$ mpirun -n 90 ./gchp
```

`cap_restart` e.g.:

```
20190213 000000
```

`GCHP.rc` e.g.:

```
...
GCHPchem_INTERNAL_RESTART_FILE:     initial_GEOSChem_rst.c24_fullchem.nc
...
```

Writing a job script

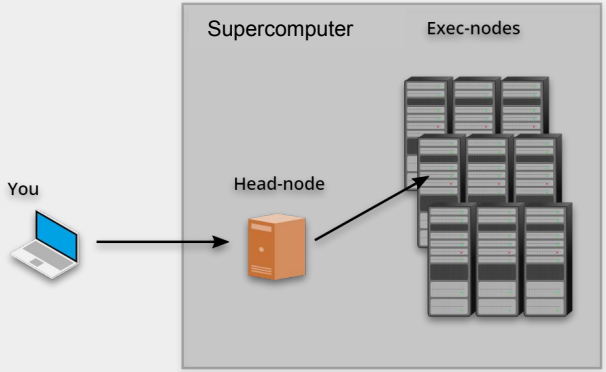
- What it needs to do:

  1. Specifies resource requirements
  2. Load software
  3. Pre-run configuration (restart file + start time)
  4. Launch GCHP
  5. (Optional) Post-run checks

- Samples in `runScriptSamples/`

- Updated runScriptSamples in 14.0 do this scripting for you!

```
#BSUB -q rvmartin
#BSUB -n 72
#BSUB -W 2:00
#BSUB -R "rusage[mem=120GB] span[ptile=36]"
#BSUB -a 'docker(geoschem/gchp:13.0.0-beta.1-13-g924e47f)

. /spack/share/spack/setup-env.sh
module load openmpi-4.0.1-gcc-9-sdj47y5

./runConfig.sh

mpiexec -n 72 ./gchp > runlog.txt
```



Changes coming in 14.0

- Start time + restart file automatically updated (credit: Lizzie Lundgren)

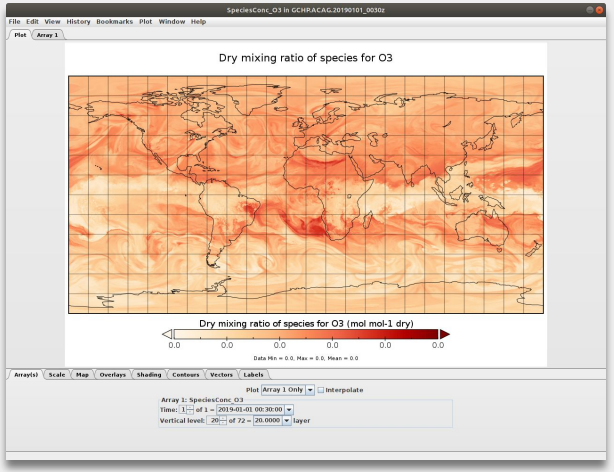
- How to use

  1. `$ cp runScriptSamples/gchp.local.run .`
  2. Add scheduler directives at top
  3. Edit launch command as necessary

```
# Define log name to include simulation start date
start_str=$(echo $(cat cap_restart) | sed 's/ /_/g')
log=gchp.${start_str}z.log

# Load environment
if [ ! -L gchp.env ] || [ ! -e gchp.env ]; then
   echo "ERROR: gchp.env symbolic link not valid. Set using setEnvironment.sh."
   exit 1
else
   source gchp.env > ${log}
fi

# Update config files, set restart symlink, and do sanity checks
source setCommonRunSettings.sh --verbose >> ${log}
source setRestartLink.sh >> ${log}
source checkRunSettings.sh >> ${log}

# Run GCHP with # processors specified in config file setCommonRunSettings.sh
nCores=$( grep -oP 'TOTAL_CORES=\s*\K\d+' setCommonRunSettings.sh )

#-----------------------------------------------------
# Edit this line to run GCHP on your system
#-----------------------------------------------------
time mpirun -np ${nCores} ./gchp 2>&1 | tee -a ${log}

# Rename and move restart file and update restart symlink if new start time ok
new_start_str=$(echo $(cat cap_restart) | sed 's/ /_/g')
if [[ "${new_start_str}" = "${start_str}" || "${new_start_str}" = "" ]]; then
   echo "ERROR: cap_restart either did not change or is empty."
   exit 1
else
    N=$(grep "CS_RES=" setCommonRunSettings.sh | cut -c 8- | xargs )
    mv gcchem_internal_checkpoint Restarts/GEOSChem.Restart.${new_start_str}z.c${N}.nc4
    source setRestartLink.sh 2>&1 | tee -a ${log}
fi
```

Working with output data

- Already covered




```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import xarray as xr

ds = xr.open_dataset('GCHP.SpeciesConc.20210508_0000z.nc4') instructions

plt.figure()
ax = plt.axes(projection=ccrs.EqualEarth())
ax.coastlines()
ax.set_global()

norm = plt.Normalize(1e-8, 7e-8)

for face in range(6):
    x = ds.corner_lons.isel(nf=face)
    y = ds.corner_lats.isel(nf=face)
    v = ds.SpeciesConc_O3.isel(time=0, lev=23, nf=face)
    ax.pcolormesh(x, y, v, norm=norm, transform=ccrs.PlateCarree())

plt.show()
```

Next steps: ideas for where to focus your effort

Getting started

- Tutorials on YouTube
- Documentation on RTD
- New sample run scripts

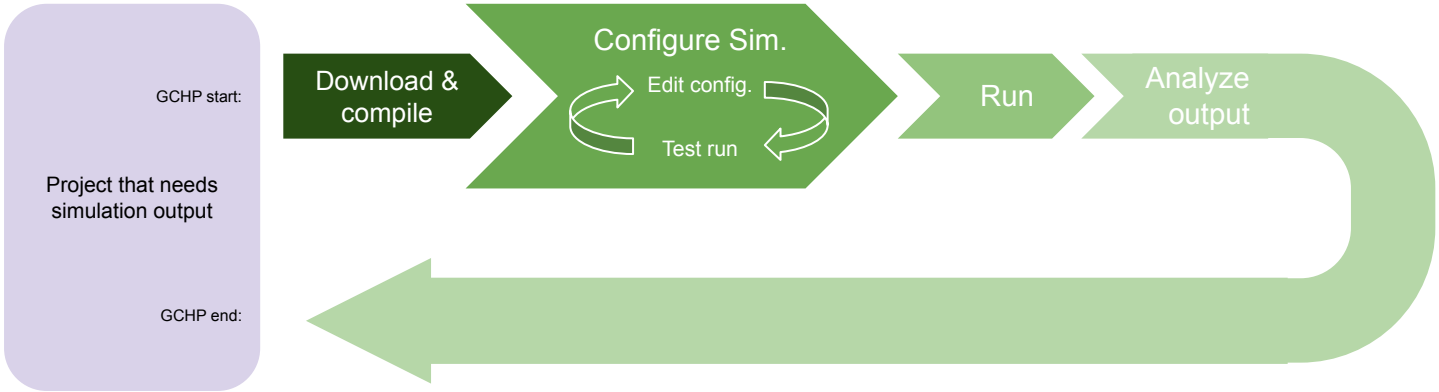
Productive practices

- Customize HISTORY.rc
- Configure + test at low resolution, crank up the resolution when ready
- Configure 1 thing at a time
- logging.yaml for troubleshooting
- Reading instructions is boring but it’s productive

Tools to learn for success

- bash
- NetCDF Operators (NCO)
- Python, Matlab, or etc.
- Climate Data Operators (CDO)

Questions?


GCHP start:
Project that needs simulation output
GCHP end:
Download & compile
Configure Sim.
Edit config.
Test run
Run
Analyze output