GCHP

Release 13.0.2

GEOS-Chem Support Team

Oct 05, 2021

GETTING STARTED

1 Quickstart Guide 3

2 System Requirements 7

3 Key References 9

4 Downloading GCHP 11

5 Compiling GCHP 13

6 Creating a Run Directory 19

7 Downloading Input Data 23

8 Running GCHP 25

9 Run Directory Configuration 29

10 Configuration Files 37

11 Example Job Scripts 49

12 Building Dependencies with Spack 51

13 Using GCHP Containers 63

14 Plotting GCHP Output 67

15 Stretched-Grid Simulations 69

16 Output Along a Track 73

17 Stretched-Grid Simulation: Eastern US 77

18 Known Bugs 81

19 Support Guidelines 83

20 Contributing Guidelines 85

21 Editing this User Guide 87

22 Git Submodules 89

23 Terminology 91

24 Versioning 93

25 Uploading to Spack 95

Bibliography 97

Index 99

Important: This is a prerelease of the GEOS-Chem High Performance user guide. These pages are the most up-to-date and accurate instructions for GCHP, but they are still a work in progress.

Contributions (e.g., suggestions, edits, revisions) would be greatly appreciated. See *editing this guide* and our *contributing guidelines*. If you find a something hard to understand—let us know!

The GEOS–Chem model is a global 3-D model of atmospheric composition driven by assimilated meteorological observations from the Goddard Earth Observing System (GEOS) of the NASA Global Modeling and Assimilation Office. It is applied by research groups around the world to a wide range of atmospheric composition problems.

- GEOS-Chem Overview
- Narrative description of GEOS-Chem

This site provides instructions for GEOS-Chem High Performance, GEOS-Chem's multi-node variant. We provide two different instruction sets for downloading and compiling GCHP: from a clone of the source code, or using the Spack package manager.

Cloning and building from source code ensures you will have direct access to the latest available versions of GCHP, provides additional compile-time options, and allows you to make your own modifications to GCHP's source code. Spack automates downloading and additional parts of the compiling process while providing you with some standard togglable compile-time options.

Our Quick Start Guide and the downloading, compiling, and creating a run directory sections of the User Guide give instructions specifically for using a clone of the source code. Our dedicated Spack guide describes how to install GCHP and create a run directory with Spack, as well as how to use Spack to install GCHP's dependencies if needed.

CHAPTER

ONE

QUICKSTART GUIDE

This quickstart guide assumes your environment satisfies *GCHP's requirements*. This means you should load a compute environment so that programs like **cmake** and **mpirun** are available before continuing. If you do not have some of GCHP's software dependencies, you can find instructions for installing GCHP's external dependencies in our Spack instructions. More detailed instructions on downloading, compiling, and running GCHP can be found in the User Guide.

1.1 1. Clone GCHP

Download the source code:

```
gcuser:~$ git clone https://github.com/geoschem/GCHP.git ~/GCHP
gcuser:~$ cd ~/GCHP
```

Checkout the GEOS-Chem version that you want to use:

```
gcuser:~/GCHP$ git checkout 13.0.0-beta.1
```

Note: Version 13 is not officially released yet. Until then, the most recent commit to `main` is the most stable version of GCHP. Therefore, we recommend you checkout `main`, rather than a version like `13.0.0-beta.1`. E.g.:

```
$ git checkout main   # recommended until version 13 is officially released
```

Once version 13 is released, we will resume recommending users checkout a specific version.

Initialize and update all the submodules:

```
gcuser:~/GCHP$ git submodule update --init --recursive
```

1.2 2. Create a run directory

Navigate to the `run/` subdirectory. To create a run directory, run `./createRunDir.sh` and answer the prompts:

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

1.3 3. Configure your build

Create a build directory and **cd** into it. A good name for this directory is `build/`, and a good place to put it is the top-level of the source code:

```
gcuser:~/GCHP$ mkdir ~/GCHP/build
gcuser:~/GCHP$ cd ~/GCHP/build
```

Initialize your build directory by running **cmake**, passing it the path to your source code:

```
gcuser:~/GCHP/build$ cmake ~/GCHP
```

Now you can configure *build options*. These are persistent settings that are saved to your build directory. A common build option is `-DRUNDIR`. This option lets you specify one or more run directories that GCHP is "installed" to when you do **make install**. Configure your build so it installs GCHP to the run directory you created in Step 2:

```
gcuser:~/GCHP/build$ cmake . -DRUNDIR="/path/to/your/run/directory"
```

Note: The `.` in the **cmake** command above is important. It tells CMake that your current working directory (i.e., `.`) is your build directory.

1.4 4. Compile and install

Compiling GCHP takes about 20 minutes, but it can varry depending on your system. Next, compile GCHP:

```
gcuser:~/GCHP/build$ make -j
```

Next, install the compiled executable to your run directory (or directories):

```
gcuser:~/GCHP/build$ make install
```

This copies `bin/gchp` and supplemental files to your run directory.

Note: You can update build settings at any time:

1. Navigate to your build directory.
2. Update your build settings with **cmake**. See

3. Recompile with **make -j**. Note that the build system automatically figures out what (if any) files need to be recompiled.
4. Install the rebuilt executable with **make install**.

1.5 5. Configure your run directory

Now, navigate to your run directory:

```
$ cd path/to/your/run/directory
```

Most simulation settings are set in `./runConfig.sh`. You should review this file as it explains most settings. Note that `./runConfig.sh` is actually a helper script that updates other configuration files. Therefore, you need to run it to actually apply the updates:

```
$ vim runConfig.sh              # edit simulation settings here
$ ./runConfig.sh                # applies the updated settings
```

1.6 6. Run GCHP

Running GCHP is slightly different depending on your MPI library (e.g., OpenMPI, Intel MPI, MVAPICH2, etc.) and scheduler (e.g., SLURM, LSF, etc.). If you aren't familiar with running MPI programs on your system, see *Running GCHP* in the user guide, or ask your system administrator.

Your MPI library and scheduler will have a command for launching MPI programs—it's usually something like **mpirun**, **mpiexec**, or **srun**. This is the command that you will use to launch the **gchp** executable. You'll have to refer to your system's documentation for specific instructions on running MPI programs, but generally it looks something like this:

```
$ mpirun -np 6 ./gchp    # example of running GCHP with 6 slots with OpenMPI
```

It's recommended you run GCHP as a batch job. This means that you write a (bash) script that runs your GCHP simulation, and then you submit that script to your scheduler (SLURM, LSF, etc.).

Note: When GCHP runs, partially or to completion, it generates several files including `cap_restart` and `gcchem_internal_checkpoint`. Subsequent runs won't overwrite these files, and instead the run will exit with an error. Because of this it is common to do

```
$ rm -f cap_restart gcchem_internal_checkpoint
```

before starting a GCHP simulation.

Those are the basics of using GCHP! See the user guide, step-by-step guides, and reference pages for more detailed instructions.

CHAPTER

TWO

SYSTEM REQUIREMENTS

2.1 Software Requirements

To use GCHP you need a compute environment with the following software:

- Git
- Make (or GNUMake)
- CMake version 3.13
- Compilers (C, C++, and Fortran):
  - Intel compilers version 18.0.5, or
  - GNU compilers version 8.3
- MPI (Message Passing Interface)
  - OpenMPI 3.0, or
  - IntelMPI, or
  - MVAPICH2, or
  - MPICH, or
  - other MPI libraries might work too
- HDF5
- NetCDF (with C, C++, and Fortran support)
- ESMF version 8.0.0

Your system administrator can tell you what software is available and how to activate it.

If you need to install any of these software yourself, you can do that manually (build from source), but it is faster and easier to do it with Spack. See our guide on *installing GCHP's dependencies with Spack*.

2.2 Hardware Requirements

These are GCHP's hardware requirements. Note that high-end HPC infrastructure is not required to use GCHP effectively. Gigabit Ethernet and 2 nodes is enough for returns on performance compared to GEOS-Chem Classic.

2.2.1 Recommended Minimum Requirements

These recommended minimums are adequate to effectively use GCHP in scientific applications:

- 2 nodes, preferably 24 cores per node
- Gigabit Ethernet (GbE) interconnect or better
- 100 GB memory per node
- 1 TB of storage

2.2.2 Bare Minimum Requirements

These bare minimum requirements are sufficient for running GCHP at C24. The are adequate for try GCHP out, and for learning purposes.

- 6 cores
- 32 GB of memory
- 100 GB of storage for input and output data

2.2.3 Big Compute Recommendations

These hardware recommendations are for users that are interested in tackling large bleeding-edge computational problems:

- A high-performance-computing cluster (or a cloud-HPC service like AWS)
  - 1–50 nodes
  - >24 cores per node (the more the better), preferably Intel Xeon
  - High throughput and low-latency interconnect, preferably InfiniBand if using 500 cores
- Lots of storage. Several TB is sufficient, but tens or hundreds of TB is better.

2.2.4 Other Recommendations

- Hyper-threading may improve simulation throughput, particularly at low core counts
- MPI process should be bound sequentially across cores and nodes (e.g., a simulation with 48-processes with 24 processes per node should bind rank 0 to CPU L#0, rank 1 to CPU L#1, etc. on the first node, and rank 24 to CPU L#0, rank 1 to CPU L#1, etc. on the second node). This should be the default, but it's worth checking if your performance is lower than expected. With OpenMPI the *–report-bindings* argument will show you how processes are ranked and binded.

CHAPTER

THREE

KEY REFERENCES

- GEOS-Chem was first described in [Bey et al., 2001].
- HEMCO is described in [Keller et al., 2014].
- Columnar operators are described in [Long et al., 2015].
- GEOS-Chem High Performance (GCHP) is described in [Eastham et al., 2018].
- GCHP execution on the cloud and MPI considerations are described in [Zhuang et al., 2020].
- Grid-stretching is described in [Bindle et al., 2020].

References

CHAPTER

FOUR

DOWNLOADING GCHP

The GCHP source code is hosted at https://github.com/geoschem/GCHP. Clone the repository:

```
gcuser:~$ git clone https://github.com/geoschem/GCHP.git Code.GCHP
```

The GCHP repository has submodules (other repositories that are nested inside the GCHP repository) that aren't automatically retrieved when you do **`git clone`**. To finish retrieving the GCHP source code, initialize and update the submodules:

```
gcuser:~$ cd Code.GCHP
gcuser:~/Code.GCHP$ git submodule update --init --recursive
```

By default, the source code will be on the `main` branch. Checking out an official release is recommended because they are scientifically-validated versions of the code, and it records the version for your future reference. You can find the list of GCHP releases here. Checkout the version that you want to work with, and update the submodules:

```
gcuser:~/Code.GCHP$ git checkout 13.0.2
gcuser:~/Code.GCHP$ git submodule update --init --recursive
```

Before continuing, it is worth checking that the source code was retrieved correctly. Run **`git status`** to check that there are no differences:

```
gcuser:~/Code.GCHP$ git status
HEAD detached at 13.0.2
nothing to commit, working tree clean
gcuser:~/Code.GCHP$
```

The output of **`git status`** should say that you are at the right version and that there are no modifications (nothing to commit, and a clean working tree).

Note: Compiling GCHP and creating a run directory are independent steps, and their order doesn't matter. A small exception is the *RUNDIR* build option, which controls the behaviour of **`make install`**; however, this setting can be reconfigured at any time (e.g., after compiling and creating a run directory).

Here in the User Guide, we describe compiling GCHP before we describe creating a run directory. This is so that conceptually the instructions have a linear flow. The Quickstart Guide uses the opposite ordering to minimize the number of commands.

Note: Another resource for GCHP build instructions is our YouTube tutorial.

CHAPTER

FIVE

COMPILING GCHP

There are two steps to building GCHP. The first is configuring your build, which is done with **cmake**; the second step is compiling, which is done with **make**.

In the first step (build configuration), **cmake** finds GCHP's *software dependencies* on your system, and you can set *build options* like enabling/disabling components, setting paths to run directories, picking between debug or speed-optimizing compiler flags, etc. The second step (running **make**) compiles GCHP according your build configuration.

Important: These instructions assume you have loaded a computing environment that satisfies *GCHP's software requirements* You can find instructions for building GCHP's dependencies yourself in the Spack instructions.

5.1 Create a build directory

A build directory is the working directory for a "build". Conceptually, a "build" is a case/instance of you compiling GCHP. A build directory stores configuration files and intermediate files related to the build. These files and generated and used by CMake, Make, and compilers. You can think a build directory like the blueprints for a construction project.

Create a new directory and initialize it as a build directory by running CMake. When you initialize a build directory, the path to the source code is a required argument:

```
gcuser:~$ cd ~/Code.GCHP
gcuser:~/Code.GCHP$ mkdir build              # create a new directory
gcuser:~/Code.GCHP$ cd build
gcuser:~/Code.GCHP/build$ cmake ~/Code.GCHP  # initialize the current dir as a build dir
-- The Fortran compiler identification is GNU 9.2.1
-- The CXX compiler identification is GNU 9.2.1
-- The C compiler identification is GNU 9.2.1
-- Check for working Fortran compiler: /usr/bin/f95
-- Check for working Fortran compiler: /usr/bin/f95  -- works
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
gcuser:~/Code.GCHP/build$
```

If your **cmake** output is similar to the snippet above, and it says configuring & generating done, then your configuration was successful and you can move on to *compiling* or *modifying build settings*. If you got an error, don't worry, that just means the automatic configuration failed. To fix the error you might need to tweak settings with more **cmake** commands, or you might need to modify your environment and run **cmake** again to retry the automatic configuration.

If you want to restart configuring your build from scratch, delete your build directory. Note that the name and location of your build directory doesn't matter, but a good name is `build/`, and a good place for it is the top-level of your source code.

5.1.1 Resolving initialization errors

If your last step was successful, *skip this section*.

Even if you got a **cmake** error, your build directory was initialized. This means from now on, you can check if the configuration is fixed by running

```
gcuser:~/Code.GCHP/build$ cmake .    # "." because the cwd is the build dir
```

To resolve your errors, you might need to modify your environment (e.g., load different software modules), or give CMake a hint about where some software is installed. Once you identify the problem and make the appropriate update, run **cmake .** to see if the error is fixed.

To start troubleshooting, read the **cmake** output in full. It is human-readable, and includes important information about how the build was set up on your system, and specifically what error is preventing a successful configuration (e.g., a dependency that wasn't found, or a compiler that is broken). To begin troubleshooting you should check that:

- check that the compilers are what you expect (e.g., GNU 9.2, Intel 19.1, etc.)
- check that dependencies like MPI, HDF5, NetCDF, and ESMF were found
- check for obvious errors/incompatibilities in the paths to "Found" dependencies

---

Note: F2PY and ImageMagick are not required. You can safely ignore warnings about them not being found.

Most errors are caused by one or more of the following issues:

- The wrong compilers were chosen. Fix this by explicitly setting the compilers.
- The compiler's version is too old. Fix this by using newer compilers.
- A software dependency is missing. Fix this by loading the appropriate software. Some hints:
  - If HDF5 is missing, does **h5cc -show** or **h5pcc -show** work?
  - If NetCDF is missing, do **nc-config --all** and **nf-config --all** work?
  - If MPI is missing, does **mpiexec --help** work?
- A software dependency is loaded but it wasn't found automatically. Fix this by pointing CMake to the missing software/files with **cmake . -DCMAKE_PREFIX_PATH=/path/to/missing/files**.
  - If ESMF is missing, point CMake to your ESMF install with `-DCMAKE_PREFIX_PATH`
- Software modules that are not compatible. Fix this by loading compatible modules/dependencies/compilers. Some hints:
  - This often shows as an error message saying a compiler is "broken" or "doesn't work"
  - E.g. incompatibility #1: you're using GNU compilers but HDF5 is built for Intel compilers
  - E.g. incompatibility #2: ESMF was compiled for a different compiler, MPI, or HDF5

If you are stumped, don't hesitate to open an issue on GitHub. Your system administrators might also be able to help. Be sure to include `CMakeCache.txt` from your build directory, as it contains useful information for troubleshooting.

Note: If you get a CMake error saying "Could not find XXXX" (where XXXX is a dependency like ESMF, NetCDF, HDF5, etc.), the problem is that CMake can't automatically find where that library is installed. You can add custom paths to CMake's default search list by setting the `CMAKE_PREFIX_PATH` variable.

For example, if you got an error saying "Could not find ESMF", and ESMF is installed to `/software/ESMF`, you would do

```
gcuser:~/Code.GCHP/build$ cmake . -DCMAKE_PREFIX_PATH=/software/ESMF
...
-- Found ESMF: /software/ESMF/include (found version "8.1.0")
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
gcuser:~/Code.GCHP/build$
```

See the next section for details on setting variables like `CMAKE_PREFIX_PATH`.

Note: You can explicitly specify compilers by setting the `CC`, `CXX`, and `FC` environment variables. If the auto-selected compilers are the wrong ones, create a brand new build directory, and set these variables before you initialize it. E.g.:

```
gcuser:~/Code.GCHP/build$ cd ..
gcuser:~/Code.GCHP$ rm -rf build   # build dir initialized with wrong compilers
gcuser:~/Code.GCHP$ mkdir build    # make a new build directory
gcuser:~/Code.GCHP$ cd build
gcuser:~/Code.GCHP/build$ export CC=icc      # select "icc" as C compiler
gcuser:~/Code.GCHP/build$ export CXX=icpc    # select "icpc" as C++ compiler
gcuser:~/Code.GCHP/build$ export FC=icc      # select "ifort" as Fortran compiler
gcuser:~/Code.GCHP/build$ cmake ~/Code.GCHP  # initialize new build dir
-- The Fortran compiler identification is Intel 19.1.0.20191121
-- The CXX compiler identification is Intel 19.1.0.20191121
-- The C compiler identification is Intel 19.1.0.20191121
...
```

5.2 Configure your build

Build settings are controlled by **cmake** commands like:

```
$ cmake . -D<NAME>="<VALUE>"
```

where `<NAME>` is the name of the setting, and `<VALUE>` is the value you are assigning it. These settings are persistent and saved in your build directory. You can set multiple variables in the same command, and you can run **cmake** as many times as needed to configure your desired settings.

Note: The `.` argument is important. It is the path to your build directory which is `.` here.

No build settings are required. You can find the complete list of *GCHP's build settings here*. The most common setting is `RUNDIR`, which lets you specify one or more run directories to install GCHP to. Here, "install" refers to copying the compiled executable, and some supplemental files with build settings, to your run directory/directories.

Note: You can update build settings after you compile GCHP. Simply rerun **make** and (optionally) **make install**, and the build system will automatically figure out what needs to be recompiled.

Since there are no required build settings, so here, we will stick with the default settings.

You should notice that when you run **cmake** it ends with:

```
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
```

This tells you that the configuration was successful, and that you are ready to compile.

5.3 Compile GCHP

You compile GCHP with:

```
gcuser:~/Code.GCHP/build$ make -j    # -j enables compiling in parallel
```

Note: You can add `VERBOSE=1` to see all the compiler commands.

Note: If you run out of memory while compiling, restrict the number of processes that can run concurrently (e.g., use `-j20` to restrict to 20 processes)

Compiling GCHP creates `./bin/gchp` (the GCHP executable). You can copy this executable to your run directory manually, or if you set the *RUNDIR* build option, you can do

```
gcuser:~/Code.GCHP/build$ make install  # Requires that RUNDIR build option is set
```

to copy the executable (and supplemental files) to your run directories.

Now you have compiled GCHP! You can move on to creating a run directory!

5.4 Recompiling

You need to recompile GCHP if you update a build setting or modify the source code. With CMake, you do not need to clean before recompiling. The build system automatically figures out which files need to be recompiled (it's usually a small subset). This is known as incremental compiling.

To recompile GCHP, simply do

```
gcuser:~/Code.GCHP/build$ make -j    # -j enables compiling in parallel
```

and then optionally, **make install**.

Note: GNU compilers recompile GCHP faster than Intel compilers. This is because of how **gfortran** formats Fortran modules files (`*.mod` files). Therefore, if you want to be able to recompile quickly, consider using GNU compilers.

5.5 GCHP build options

These are persistent build setting that are set with **cmake** commands like

```
$ cmake . -D<NAME>="<VALUE>"
```

where <NAME> is the name of the build setting, and <VALUE> is the value you are assigning it. Below is the list of build settings for GCHP.

RUNDIR Paths to run directories where **make install** installs GCHP. Multiple run directories can be specified by a semicolon separated list. A warning is issues if one of these directories does not look like a run directory.

These paths can be relative paths or absolute paths. Relative paths are interpreted as relative to your build directory.

CMAKE_BUILD_TYPE The build type. Valid values are `Release`, `Debug`, and `RelWithDebInfo`. Set this to `Debug` if you want to build in debug mode.

CMAKE_PREFIX_PATH Extra directories that CMake will search when it's looking for dependencies. Directories in `CMAKE_PREFIX_PATH` have the highest precedence when CMake is searching for dependencies. Multiple directories can be specified with a semicolon-separated list.

GEOSChem_Fortran_FLAGS_<COMPILER_ID> Compiler options for GEOS-Chem for all build types. Valid values for <COMPILER_ID> are `GNU` and `Intel`.

GEOSChem_Fortran_FLAGS_<BUILD_TYPE>_<COMPILER_ID> Additional compiler options for GEOS-Chem for build type <BUILD_TYPE>.

HEMCO_Fortran_FLAGS_<COMPILER_ID> Same as `GEOSChem_Fortran_FLAGS_`<COMPILER_ID>, but for HEMCO.

HEMCO_Fortran_FLAGS_<BUILD_TYPE>_<COMPILER_ID> Same as `GEOSChem_Fortran_FLAGS_`<BUILD_TYPE>_<COMPI but for HEMCO.

RRTMG Switch to enable/disable the RRTMG component.

OMP Switch to enable/disable OpenMP multithreading. As is standard in CMake (see if documentation) valid values are `ON`, `YES`, `Y`, `TRUE`, or `1` (case-insensitive) and valid false values are their opposites.

INSTALLCOPY Similar to `RUNDIR`, except the directories do not need to be run directories.

CHAPTER

SIX

CREATING A RUN DIRECTORY

Run directories are created with the `createRunDir.sh` script in the `run/` subdirectory of the source code. Run directories are version-specific, so you need to create new run directories for every GEOS-Chem version. The gist of creating a run directory is simple: navigate to the `run/` subdirectory, run `./createRunDir.sh`, and answer the prompts:

```
gcuser:~$ cd Code.GCHP/run
gcuser:~/Code.GCHP/run$ ./createRunDir.sh
... <answer the prompts> ...
```

Important: Use *absolute paths* when responding to prompts.

Create a run directory. If you are unsure what a prompt is asking, see their explanations below, or ask a question on GitHub. After creating a run directory, you can move on to the next section.

6.1 Explanations of Prompts

Below are detailed explanations of the prompts in `./createRunDir.sh`.

6.1.1 Enter ExtData path

The first time you create a GCHP run directory on your system you will be prompted for a path to GEOS-Chem shared data directories. The path should include the name of your `ExtData/` directory and should not contain symbolic links. The path you enter will be stored in file `.geoschem/config` in your home directory as environment variable `GC_DATA_ROOT`. If that file does not already exist it will be created for you. When creating additional run directories you will only be prompted again if the file is missing or if the path within it is not valid.

```
------------------------------------------------------------
Enter path for ExtData:
------------------------------------------------------------
```

6.1.2 Choose a simulation type

Enter the integer number that is next to the simulation type you want to use.

```
------------------------------------------------------------
Choose simulation type:
------------------------------------------------------------
  1. Full chemistry
  2. TransportTracers
```

If creating a full chemistry run directory you will be given additional options. Enter the integer number that is next to the simulation option you want to run.

```
------------------------------------------------------------
Choose additional simulation option:
------------------------------------------------------------
  1. Standard
  2. Benchmark
  3. Complex SOA
  4. Marine POA
  5. Acid uptake on dust
  6. TOMAS
  7. APM
  8. RRTMG
```

6.1.3 Choose meteorology source

Enter the integer number that is next to the input meteorology source you would like to use.

```
------------------------------------------------------------
Choose meteorology source:
------------------------------------------------------------
  1. MERRA2 (Recommended)
  2. GEOS-FP
```

6.1.4 Enter run directory path

Enter the target path where the run directory will be stored. You will be prompted to enter a new path if the one you enter does not exist.

```
------------------------------------------------------------
Enter path where the run directory will be created:
------------------------------------------------------------
```

6.1.5 Enter run directory name

Enter the run directory name, or accept the default. You will be prompted for a new name if a run directory of the same name already exists at the target path.

```
------------------------------------------------------------
Enter run directory name, or press return to use default:
------------------------------------------------------------
```

6.1.6 Enable version control (optional)

Enter whether you would like your run directory tracked with git version control. With version control you can keep track of exactly what you changed relative to the original settings. This is useful for trouble-shooting as well as tracking run directory feature changes you wish to migrate back to the standard model.

```
------------------------------------------------------------
Do you want to track run directory changes with git? (y/n)
------------------------------------------------------------
```

CHAPTER

SEVEN

DOWNLOADING INPUT DATA

Input data for GEOS-Chem is available from the following FTP servers:

- http://geoschemdata.wustl.edu/ExtData/ (preferred)
- http://geoschemdata.computecanada.ca/ExtData/
- http://ftp.as.harvard.edu/gcgrid/data/ExtData/

Notably, you will need four types of input data to run GCHP:

Restart file (initial conditions) These are initial conditions for the simulation. You can find some reasonable default initial conditions in the `GEOSCHEM_RESTARTS/` directory. Default run directories have symlinks to these default restart files.

Meteorological data GCHP is driven by meteorological data from GEOS-FP or MERRA2. The GEOS-FP data is in `GEOS_0.5x0.625/MERRA2/` and the MERRA2 data is in `GEOS_0.25x0.3125/GEOS_FP/`. Default run directories have a symlink (`./MetDir`) to the local copy of this data.

Emissions data Emissions data is in the `HEMCO/` directory. In this directory are `INVENTORY/vYYYY-MM/` subdirectories, where `INVENTORY` is the name of the inventory, and `vYYYY-MM` is the date of the inventory version. Default run directories have a symlink (`./HcoDir`) to the local copy of this data.

Chemistry inputs These are miscellaneous data files for GEOS-Chem. They are in the `CHEM_INPUTS/` directory. Default run directories have a symlink (`./ChemDir`) to the local copy of this data.

You can get a url list for the input data for a simulation by running (in your run directory)

```
$ ./utils/listInputDataFiles 20190101 20190108 --wget-urls > urls.txt
... <answer prompts> ...
```

Replace `20190101` with your desired start date and `20190108` with your desired end date. For climatological data you will be asked to select a year. If the climatology covers your simulation period, you can leave it blank (hit enter). If the climatology doesn't cover your simulation period, pick the closest year. Note that this script does not include restart files, and it's a "best effort" estimation of the input data for your simulation. At runtime you might encounter errors due from missing data files (download these yourself).

You can download the files in `urls.txt` like so (replace `/ExtData` with the path to your local copy of GEOS-Chem input data)

```
$ cp urls.txt /ExtData/urls.txt  # copy url list to local /ExtData directory
$ cd /ExtData                    # cd to root of local /ExtData directory
$ wget -i urls.txt -cxnH --cut-dirs=1   # download all the data
```

CHAPTER

EIGHT

RUNNING GCHP

Note: Another useful resource for instructions on running GCHP is our YouTube tutorial.

This page presents the basic information needed to run GCHP as well as how to verify a successful run and reuse a run directory. A pre-run checklist is included at the end to help prevent run errors. The GCHP "standard" simulation run directory is configured for a 1-hr simulation at c24 resolution and is a good first test case to check that GCHP runs on your system.

8.1 How to run GCHP

You can run GCHP locally from within your run directory ("interactively") or by submitting your run to a job scheduler if one is available. Either way, it is useful to put run commands into a reusable script we call the run script. Executing the script will either run GCHP or submit a job that will run GCHP.

There is a symbolic link in the GCHP run directory called `runScriptSamples` that points to a directory in the source code containing example run scripts. Each file includes extra commands that make the run process easier and less prone to user error. These commands include:

1. Source environment file symbolic link `gchp.env` to ensure run environment consistent with build
2. Source config file `runConfig.sh` to set run-time configuration
3. Delete any previous run output files that might interfere with the new run if present
4. Send standard output to run-time log file `gchp.log`
5. Rename the output restart file to include "restart" and datetime

8.1.1 Run interactively

Copy or adapt example run script gchp.local.run to run GCHP locally on your machine. Before running, open your run script and set nCores to the number of processors you plan to use. Make sure you have this number of processors available locally. It must be at least 6. Next, open file runConfig.sh and set NUM_CORES, NUM_NODES, and NUM_CORES_PER_NODE to be consistent with your run script.

To run, type the following at the command prompt:

```
$ ./gchp.local.run
```

Standard output will be displayed on your screen in addition to being sent to log file `gchp.log`.

8.1.2 Run as batch job

Batch job run scripts will vary based on what job scheduler you have available. Most of the example run scripts are for use with SLURM, and the most basic example of these is `gchp.run`. You may copy any of the example run scripts to your run directory and adapt for your system and preferences as needed.

At the top of all batch job scripts are configurable run settings. Most critically are requested # cores, # nodes, time, and memory. Figuring out the optimal values for your run can take some trial and error. For a basic six core standard simulation job on one node you should request at least 20 min and 32GB of memory. The more cores you request the faster GCHP will run.

To submit a batch job using SLURM:

```
$ sbatch gchp.run
```

To submit a batch job using Grid Engine:

```
$ qsub gchp.run
```

Standard output will be sent to log file `gchp.log` once the job is started unless you change that feature of the run script. Standard error will be sent to a file specific to your scheduler, e.g. `slurm-jobid.out` if using SLURM, unless you configure your run script to do otherwise.

If your computational cluster uses a different job scheduler, e.g. Grid Engine, LSF, or PBS, check with your IT staff or search the internet for how to configure and submit batch jobs. For each job scheduler, batch job configurable settings and acceptable formats are available on the internet and are often accessible from the command line. For example, type **`man sbatch`** to scroll through options for SLURM, including various ways of specifying number of cores, time and memory requested.

8.2 Verify a successful run

There are several ways to verify that your run was successful.

1. NetCDF files are present in the `OutputDir/` subdirectory
2. Standard output file `gchp.log` ends with `Model Throughput` timing information
3. The job scheduler log does not contain any error messages

If it looks like something went wrong, scan through the log files to determine where there may have been an error. Here are a few debugging tips:

- Review all of your configuration files to ensure you have proper setup
- `MAPL_Cap` errors typically indicate an error with your start time, end time, and/or duration set in `runConfig.sh`
- `MAPL_ExtData` errors often indicate an error with your input files specified in either `HEMCO_Config.rc` or `ExtData.rc`
- `MAPL_HistoryGridComp` errors are related to your configured output in `HISTORY.rc`

If you cannot figure out where the problem is please do not hesitate to create a GCHPctm GitHub issue.

8.3 Reuse a run directory

8.3.1 Archive run output

Reusing a GCHP run directory comes with the perils of losing your old work. To mitigate this issue there is utility shell script `archiveRun.sh`. This script archives data output and configuration files to a subdirectory that will not be deleted if you clean your run directory.

Archiving runs is useful for other reasons as well, including:

- Save all settings and logs for later reference after a run crashes
- Generate data from the same executable using different run-time settings for comparison, e.g. c48 versus c180
- Run short runs in quick succession for debugging

To archive a run, pass the archive script a descriptive subdirectory name where data will be archived. For example:

```
$ ./archiveRun.sh 1mo_c24_24hrdiag
```

All files are archived to subfolders in the new directory. Which files are copied and to where are displayed on the screen. Diagnostic files in the `OutputDir/` directory are moved rather than copied so as not to duplicate large files. You will be prompted at the command line to accept this change prior to data move.

8.3.2 Clean a run directory

You should always clean your run directory prior to your next run. This avoids confusion about what output was generated when and with what settings. Under certain circumstances it also avoids having your new run crash. GCHP will crash if:

- Output file `cap_restart` is present and you did not change your start/end times
- Your last run failed in such a way that the restart file was not renamed in the post-run commands in the run script

The example run scripts include extra commands to clean the run directory of the two problematic files listed above. However, you may write your own run script and omit them in which case not cleaning the run directory prior to rerun will cause problems.

To make run directory cleaning simple is utility shell script `cleanRunDir.sh`. To clean the run directory simply execute this script.

```
$ ./cleanRunDir.sh
```

All GCHP output files, including diagnostics files in `OutputDir/`, will then be deleted. Only restart files with names matching `gcchem*` are deleted. This preserve the initial restart symbolic links that come with the run directory.

8.4 Pre-run checklist

Prior to running GCHP, always run through the following checklist to ensure everything is set up properly.

1. Your run directory contains the executable `gchp`.
2. All symbolic links in your run directory are valid (no broken links)
3. You have looked through and set all configurable settings in `runConfig.sh`

4. If running via a job scheduler: you have a run script and the resource allocation in `runConfig.sh` and your run script are consistent (# nodes and cores)
5. If running interactively: the resource allocation in `runConfig.sh` is available locally
6. If reusing a run directory (optional but recommended): you have archived your last run with `./archiveRun.sh` if you want to keep it and you have deleted old output files with `./cleanRunDir.sh`

8.5 Recommended MPI configuration

8.5.1 IntelMPI

```
export I_MPI_ADJUST_GATHERV=3
export I_MPI_ADJUST_ALLREDUCE=12
```

8.5.2 OpenMPI

At high-core counts (e.g., > ~1000 cores) it's recommended to set `WRITE_RESTART_BY_OSERVER: YES` in `GCHP.rc`.

CHAPTER

NINE

RUN DIRECTORY CONFIGURATION

All GCHP run directories have default simulation-specific run-time settings that are set when you create a run directory. You will likely want to change these settings. This page goes over how to do this.

Table of contents

- *Run Directory Configuration*
  - *Configuration files*
  - *Common options*
    - *Compute configuration*
      - *Set number of nodes and cores*
      - *Change domain stack size*
    - *Basic run settings*
      - *Set cubed-sphere grid resolution*
      - *Set stretching parameters*
      - *Turn on/off model components*
      - *Change model timestep*
      - *Set simulation start and end dates*
    - *Inputs*
      - *Change initial restart file*
      - *Turn on/off emissions inventories*
      - *Add new emissions files*
    - *Outputs*
      - *Output diagnostics data on a lat-lon grid*
      - *Output restart files at regular or irregular frequency*
      - *Turn on/off diagnostics*
      - *Set diagnostic frequency, duration, and mode*
      - *Add a new diagnostics collection*
      - *Generate monthly mean diagnostics*

        * *Debugging*
            - *Enable maximum print output*

9.1 Configuration files

GCHP is controlled using a set of configuration files that are included in the GCHP run directory. Files include:

1. `CAP.rc`
2. `ExtData.rc`
3. `GCHP.rc`
4. `input.geos`
5. `HEMCO_Config.rc`
6. `HEMCO_Diagn.rc`
7. `input.nml`
8. `HISTORY.rc`
9. `logging.yml`

Several run-time settings must be set consistently across multiple files. Inconsistencies may result in your program crashing or yielding unexpected results. To avoid mistakes and make run configuration easier, bash shell script `runConfig.sh` is included in all run directories to set the most commonly changed config file settings from one location. Sourcing this script will update multiple config files to use values specified in file.

Sourcing `runConfig.sh` is done automatically prior to running GCHP if using any of the example run scripts, or you can do it at the command line. Information about what settings are changed and in what files are standard output of the script. To source the script, type the following:

```
$ source runConfig.sh
```

You may also use it in silent mode if you wish to update files but not display settings on the screen:

```
$ source runConfig.sh --silent
```

While using `runConfig.sh` to configure common settings makes run configure much simpler, it comes with a major caveat. If you manually edit a config file setting that is also set in `runConfig.sh` then your manual update will be overrided via string replacement. Please get very familiar with the options in `runConfig.sh` and be conscientious about not updating the same setting elsewhere.

You generally will not need to know more about the GCHP configuration files beyond what is listed on this page. However, for a comprehensive description of all configuration files used by GCHP see the Configuration Files section of this user manual.

9.2 Common options

9.2.1 Compute configuration

Set number of nodes and cores

To change the number of nodes and cores for your run you must update settings in two places: (1) `runConfig.sh`, and (2) your run script. The `runConfig.sh` file contains detailed instructions on how to set resource parameter options and what they mean. Look for the Compute Resources section in the script. Update your resource request in your run script to match the resources set in `runConfig.sh`.

It is important to be smart about your resource allocation. To do this it is useful to understand how GCHP works with respect to distribution of nodes and cores across the grid. At least one unique core is assigned to each face on the cubed sphere, resulting in a constraint of at least six cores to run GCHP. The same number of cores must be assigned to each face, resulting in another constraint of total number of cores being a multiple of six. Communication between the cores occurs only during transport processes.

While any number of cores is valid as long as it is a multiple of six (although there is an upper limit per resolution), you will typically start to see negative effects due to excessive communication if a core is handling less than around one hundred grid cells or a cluster of grid cells that are not approximately square. You can determine how many grid cells are handled per core by analyzing your grid resolution and resource allocation. For example, if running at C24 with six cores each face is handled by one core (6 faces / 6 cores) and contains 576 cells (24x24). Each core therefore processes 576 cells. Since each core handles one face, each core communicates with four other cores (four surrounding faces). Maximizing squareness of grid cells per core is done automatically within `runConfig.sh` if variable `NXNY_AUTO` is set to `ON`.

Further discussion about domain decomposition is in `runConfig.sh` section `Domain Decomposition`.

Change domain stack size

For runs at very high resolution or small number of processors you may run into a domains stack size error. This is caused by exceeding the domains stack size memory limit set at run-time and the error will be apparent from the message in your log file. If this occurs you can increase the domains stack size in file `input.nml`. The default is set to 20000000.

9.2.2 Basic run settings

Set cubed-sphere grid resolution

GCHP uses a cubed sphere grid rather than the traditional lat-lon grid used in GEOS-Chem Classic. While regular lat-lon grids are typically designated as Lat Lon (e.g. 45), cubed sphere grids are designated by the side-length of the cube. In GCHP we specify this as CX (e.g. C24 or C180). The simple rule of thumb for determining the roughly equivalent lat-lon resolution for a given cubed sphere resolution is to divide the side length by 90. Using this rule you can quickly match C24 with about 4x5, C90 with 1 degree, C360 with quarter degree, and so on.

To change your grid resolution in the run directory edit the `CS_RES` integer parameter in `runConfig.sh` section `Internal Cubed Sphere Resolution` to the cube side length you wish to use. To use a uniform global grid resolution make sure that `STRETCH_GRID` is set to `OFF`.

Set stretching parameters

GCHP has the capability to run with a stretched grid, meaning one portion of the globe is stretched to fine resolution. Set stretched grid parameter in `runConfig.sh` section Internal Cubed Sphere Resolution. See instructions in that section of the file.

Turn on/off model components

You can toggle all primary GEOS-Chem components, including type of mixing, from within `runConfig.sh`. The settings in that file will update `input.geos` and `HEMCO_Config.rc` automatically. Look for section `Turn Components On/Off`. Please note that you can turn on/off all base emissions from `runConfig.sh` but you cannot toggle HEMCO extensions or individual base emissions. For this you must manually edit `HEMCO_Config.rc`.

Change model timestep

Model timesteps, both chemistry and dynamic, are configured within `runConfig.sh`. They are set to match GEOS-Chem Classic default values for low resolutions for comparison purposes but can be updated, with caution. Timesteps are automatically reduced for high resolution runs. Read the documentation in `runConfig.sh` section `Timesteps` for setting them.

Set simulation start and end dates

Set simulation start and end in `runConfig.sh` section `Simulation Start, End, Duration, # runs`. Read the comments in the file for a complete description of the options. Typically a "CAP" runtime error indicates a problem with start, end, and duration settings. If you encounter an error with the words "CAP" near it then double-check that these settings make sense.

9.2.3 Inputs

Change initial restart file

All GCHP run directories come with symbolic links to initial restart files for commonly used cubed sphere resolutions. The appropriate restart file is automatically chosen based on the cubed sphere resolution you set in `runConfig.sh`.

You may overwrite the default restart file with your own by specifying the restart filename in `runConfig.sh` section `Initial Restart File`. Beware that it is your responsibility to make sure it is the proper grid resolution.

Unlike GEOS-Chem Classic, HEMCO restart files are not used in GCHP. HEMCO restart variables may be included in the initial species restart file, or they may be excluded and HEMCO will start with default values. GCHP initial restart files that come with the run directories do not include HEMCO restart variables, but all output restart files do.

Turn on/off emissions inventories

Because file I/O impacts GCHP performance it is a good idea to turn off file read of emissions that you do not need. You can turn individual emissions inventories on or off the same way you would in GEOS-Chem Classic, by setting the inventories to true or false at the top of configuration file `HEMCO_Config.rc`. All emissions that are turned off in this way will be ignored when GCHP uses `ExtData.rc` to read files, thereby speeding up the model.

For emissions that do not have an on/off toggle at the top of the file, you can prevent GCHP from reading them by commenting them out in `HEMCO_Config.rc`. No updates to `ExtData.rc` would be necessary. If you alternatively comment out the emissions in `ExtData.rc` but not `HEMCO_Config.rc` then GCHP will fail with an error when looking for the file information.

Another option to skip file read for certain files is to replace the file path in `ExtData.rc` with `/dev/null`. However, if you want to turn these inputs back on at a later time you should preserve the original path by commenting out the original line.

Add new emissions files

There are two steps for adding new emissions inventories to GCHP:

1. Add the inventory information to `HEMCO_Config.rc`.
2. Add the inventory information to `ExtData.rc`.
3. To add information to `HEMCO_Config.rc`, follow the same rules as you would for adding a new emission inventory to GEOS-Chem Classic. Note that not all information in `HEMCO_Config.rc` is used by GCHP. This is because HEMCO is only used by GCHP to handle emissions after they are read, e.g. scaling and applying hierarchy. All functions related to HEMCO file read are skipped. This means that you could put garbage for the file path and units in `HEMCO_Config.rc` without running into problems with GCHP, as long as the syntax is what HEMCO expects. However, we recommend that you fill in `HEMCO_Config.rc` in the same way you would for GEOS-Chem Classic for consistency and also to avoid potential format check errors.

Staying consistent with the information that you put into `HEMCO_Config.rc`, add the inventory information to `ExtData.rc` following the guidelines listed at the top of the file and using existing inventories as examples. You can ignore all entries in `HEMCO_Config.rc` that are copies of another entry since putting these in `ExtData.rc` would result in reading the same variable in the same file twice. HEMCO interprets the copied variables, denoted by having dashes in the `HEMCO_Config.rc` entry, separate from file read.

A few common errors encountered when adding new input emissions files to GCHP are:

1. Your input file contains integer values. Beware that the MAPL I/O component in GCHP does not read or write integers. If your data contains integers then you should reprocess the file to contain floating point values instead.
2. Your data latitude and longitude dimensions are in the wrong order. Lat must always come before lon in your inputs arrays, a requirement true for both GCHP and GEOS-Chem Classic.
3. Your 3D input data are mapped to the wrong levels in GEOS-Chem (silent error). If you read in 3D data and assign the resulting import to a GEOS-Chem state variable such as `State_Chm` or `State_Met`, then you must flip the vertical axis during the assignment. See files `Includes_Before_Run.H` and setting `State_Chm%Species` in `Chem_GridCompMod.F90` for examples.
4. You have a typo in either `HEMCO_Config.rc` or `ExtData.rc`. Error in `HEMCO_Config.rc` typically result in the model crashing right away. Errors in `ExtData.rc` typically result in a problem later on during ExtData read. Always try running with all debug prints on in your first test with new emissions. See separate section on this page about turning on debug prints, or check the debug print section of `runConfig.sh` for instructions. Another useful strategy is to find config file entries for similar input files and compare them against the entry for your new file. Directly comparing the file metadata may also lead to insights into the problem.

9.2.4 Outputs

Output diagnostics data on a lat-lon grid

See documentation in the `HISTORY.rc` config file for instructions on how to output diagnostic collection on lat-lon grids. If outputting on a lat-lon grid you may also output regional data instead of global.

Output restart files at regular or irregular frequency

The MAPL component in GCHP has the option to output restart files (also called checkpoint files) prior to run end. The frequency of restart file write may be at regular time intervals (regular frequency) or at specific programmed times (irregular frequency). These periodic output restart files contain the date and time in their filenames.

Enabling this feature is a good idea if you plan on doing a long simulation and you are not splitting your run into multiple jobs. If the run crashes unexpectedly then you can restart mid-run rather than start over from the beginning.

Update settings for checkpoint restart outputs in `runConfig.sh` section `Output Restarts`. Instructions for configuring both regular and irregular frequency restart files are included in the file. Note that because months have an irregular number of days, outputting monthly checkpoint files requires using the irregular checkpoint option.

Turn on/off diagnostics

To turn diagnostic collections on or off, comment ("#") collection names in the "COLLECTIONS" list at the top of file `HISTORY.rc`. Collections cannot be turned on/off from `runConfig.sh`.

Set diagnostic frequency, duration, and mode

All diagnostic collections that come with the run directory have frequency and duration auto-set within `runConfig.sh`. The file contains a list of time-averaged collections and instantaneous collections, and allows setting a frequency and duration to apply to all collections listed for each. Time-avraged collections also have a monthly mean option (see separate section on this page about monthly mean). To avoid auto-update of a certain collection, remove it from the list in `runConfig.sh`, or set `AutUpdate_Diagnostics` to `OFF`. See section `Diagnostics` within `runConfig.sh` for examples.

Add a new diagnostics collection

Adding a new diagnostics collection in GCHP is the same as for GEOS-Chem Classic netcdf diagnostics. You must add your collection to the collection list in `HISTORY.rc` and then define it further down in the file. Any 2D or 3D arrays that are stored within GEOS-Chem objects `State_Met`, `State_Chm`, or `State_Diag`, may be included as fields in a collection. `State_Met` variables must be preceded by "Met_", `State_Chm` variables must be preceded by "Chem_", and `State_Diag` variables should not have a prefix. Collections may have a combination of 2D and 3D variables, but all 3D variables must have the same number of levels. See the `HISTORY.rc` file for examples.

Once implemented, you can either incorporate the new collection settings into `runConfig.sh` for auto-update, or you can manually configure all settings in `HISTORY.rc`. See the `Diagnostics` section of `runConfig.sh` for more information.

Generate monthly mean diagnostics

You can toggle monthly mean diagnostics on/off from within `runConfig.sh` in the diagnostics configurations section. If you do this then ALL time-averaged diagnostic collections listed in `runConfig.sh` will have monthly diagnostics enabled. As for collection frequency and duration, you can remove collections from the list in `runConfig.sh` to prevent auto-update, or set `AutoUpdate_Diagnostics` to `OFF`, and then manually update settings for those collections in `HISTORY.rc` prior to your run. This would allow you to have some time-averaged diagnostics be monthly mean and some not. If using monthly diagnostics, be sure to always start and end your run on the first day and time of the month.

9.2.5 Debugging

Enable maximum print output

Besides compiling with `CMAKE_BUILD_TYPE=Debug`, there are a few settings you can configure to boost your chance of successful debugging. All of them involve sending additional print statements to the log files.

1. Set Turn on debug printout? in input.geos to T to turn on extra GEOS-Chem print statements in the main log file.
2. Set the Verbose and Warnings settings in `HEMCO_Config.rc` to maximum values of 3 to send the maximum number of prints to `HEMCO.log`.
3. Set `CAP.EXTDATA` option `root_level` in `logging.yml` to `DEBUG` to send root thread MAPL ExtData (input) prints to `allPEs.log`.
4. Set `CAP.EXTDATA` option `level` in `logging.yml` to `DEBUG` to send all thread MAPL ExtData (input) prints to `allPEs.log`.

None of these options require recompiling. Be aware that all of them will slow down your simulation. Be sure to set them back to the default values after you are finished debugging.

CHAPTER

TEN

CONFIGURATION FILES

GCHP is controlled using a set of resource configuration files that are included in the GCHP run directory, most of which are denoted by suffix `.rc`. These files contain all run-time information required by GCHP. Files include:

10.1 HISTORY.rc

`HISTORY.rc` is the file that configures GCHP's output. It has the following format

```
EXPID:  OutputDir/GCHP
EXPDSC: GEOS-Chem_devel
CoresPerNode: 30
VERSION: 1

<DEFINE GRID LABELS>

<DEFINE ACTIVE COLLECTIONS>

<DEFINE COLLECTIONS>
```

EXPID This is the file prefix for all collections. `OutputDir/GCHP` means that collections will be to a direcotry named `OutputDir/`, and the file names will start with *GCHP*.

CoresPerNode The number of cores per node for your GCHP simulation.

EXPDSC Leave this as it is.

VERSION Leave this as it is.

The format and description of *<DEFINE GRID LABELS>*, *<DEFINE ACTIVE COLLECTIONS>*, and and *<DEFINE COLLECTIONS>* sections are given below.

10.1.1 Defining Grid Labels

You can specify custom grids for you output. For example, a regional 0.05°x0.05° grid covering North America. This way your collections are regridded online. There are two advantages to doing this

1. It eliminates the need to regrid your simulation data in a post-processing step.
2. It saves disk space if you are interested in regional output.

You can define as many grids as you want. A collection can define `grid_label` to select a custom grid. If a collection does not define `grid_label` the simulation's grid is assumed.

Below is the format for the `<DEFINE GRID LABELS>` section in `HISTORY.rc`.

```
GRID_LABELS:   MY_FIRST_GRID       # My custom grid for C96 output
               MY_SECOND_GRID      # My custom grid for global 0.5x0.625 output
               MY_THIRD_GRID       # My custom grid for regional 0.05x0.05 output
::
   MY_FIRST_GRID.GRID_TYPE:   Cubed-Sphere
   MY_FIRST_GRID.IM_WORLD:    96
   MY_FIRST_GRID.JM_WORLD:    576      # 576=6x96

   MY_SECOND_GRID.GRID_TYPE:  LatLon
   MY_SECOND_GRID.IM_WORLD:   360
   MY_SECOND_GRID.JM_WORLD:   181
   MY_SECOND_GRID.POLE:       PC       # pole-centered
   MY_SECOND_GRID.DATELINE:   DC       # dateline-centered

   MY_THIRD_GRID.GRID_TYPE:   LatLon
   MY_THIRD_GRID.IM_WORLD:    80
   MY_THIRD_GRID.JM_WORLD:    40
   MY_THIRD_GRID.POLE:        XY
   MY_THIRD_GRID.DATELINE:    XY
   MY_THIRD_GRID.LON_RANGE:    0 80    # regional boundaries
   MY_THIRD_GRID.LAT_RANGE:  -30 10
```

SPEC NAMES

GRID_TYPE The type of grid. Valid options are `Cubed-Sphere` or `LatLon`.

IM_WORLD The number of grid boxes in the i-dimension. For a `LatLon` grid this is the number of longitude grid-boxes. For a `Cubed-Sphere` grid this is the cubed-sphere size (e.g., 48 for C48).

JM_WORLD The number of grid boxes in the j-dimension. For a `LatLon` grid this is the number of latitude grid-boxes. For a `Cubed-Sphere` grid this is six times the cubed-sphere size (e.g., 288 for C48).

POLE Required if the grid type is `LatLon`. `POLE` defines the latitude coordinates of the grid. For global lat-lon grids the valid options are `PC` (pole-centered) or `PE` (polar-edge). Here, "center" or "edge" refers to whether the grid has boxes that are centered on the poles, or whether the grid has boxes with edges at the poles. For regional grids `POLE` should be set to `XY` and the grid will have boxes with edges at the regional boundaries.

DATELINE Required if the grid type is `LatLon`. `DATELINE` defines the longitude coordinates of the grid. For global lat-lon grids the valid options are `DC` (dateline-centered), `DE` (dateline-edge), `GC` (grenwich-centered), or `GE` (grenwich-edge). If `DC` or `DE`, then the longitude coordinates will span (-180°, 180°). If `GC` or `GE`, then the longitude coordinates will span (0°, 360°). Similar to `POLE`, "center" or "edge" refer to whether the grid has boxes that are centered at -180° or 0°, or whether the grid has boxes with edges at -180° or 0°. For regional grids `DATELINE` should be set to *XY* and the grid will have boxes with edges at the regional boundaries.

LON_RANGE Required for regional `LatLon` grids. `LON_RANGE` defines the longitude bounds of the regional grid.

LAT_RANGE Required for regional `LatLon` grids. `LAT_RANGE` defines the latitude bounds of the regional grid.

10.1.2 Defining Active Collections

Collections are activated by defining them in the `COLLECTIONS` list. For instructions on defining collections, see *Defining Collections*.

Below is the format for the <DEFINE ACTIVE COLLECTIONS> section of `HISTORY.rc`.

```
COLLECTIONS:   'MyCollection1',
               'MyCollection2',
::
```

This example activates collections named "MyCollection1" and "MyCollection2".

10.1.3 Defining Collections

A collection is

```
MyCollection1.template:    '%y4%m2%d2_%h2%n2z.nc4',
MyCollection1.format:      'CFIO',
MyCollection1.frequency:   010000
MyCollection1.duration:    240000
MyCollection1.mode:        'time-averaged'
MyCollection1.fields:      'SpeciesConc_O3  ',  'GCHPchem',
                           'SpeciesConc_NO  ',  'GCHPchem',
                           'SpeciesConc_NO2 ',  'GCHPchem',
                           'Met_BXHEIGHT    ',  'GCHPchem',
                           'Met_AIRDEN      ',  'GCHPchem',
                           'Met_AD          ',  'GCHPchem',
::
<DEFINE MORE COLLECTIONS ...>
```

Output file configuration

template This is the file name suffix for the collection. The path to the collection's files is obtained by concatenating `EXPID` with the collection name and the value of `template`.

format Defines the file format of the collection. Valid values are `'CFIO'` for CF compliant NetCDF (recommended), or `'flat'` for GrADS style flat files.

duration Defines the frequency at which files are generated. The format is `HHMMSS`. For example, `1680000` means that a file is generated every 168 hours (7 days).

timeStampStart *[optional]* Only used if `mode` is `'time-averaged'`. If `.true.` the file is timestamped according to the start of the accumulation interval (which depends on `frequency`, `ref_date`, and `ref_time`). If `.false.` the file is timestamped according to the middle of the accumulation interval. If `timeStampStart` is not set then the default value is false.

Sampling configuration

mode Defines the sampling method. Valid values are `'time-averaged'` or `'instantaneous'`.

frequency Defines the time frequency of collection's data. Said another way, this defines the time separation (time step) of the time coordinate for the collection. The format is `HHMMSS`. For example, `010000` means that the collection's time coordinate will have a 1-hour time step. If `frequency` is less than `duration` multiple time steps are written to each file.

acc_interval *[optional]* Only valid if `mode` is `'time-averaged'`. This specifies the length of the time average. By default it is equal to `frequency`.

ref_date *[optional]* The reference date from which the frequency is based. The format is `YYYYMMDD`. For example, a frequency of `1680000` (7 days) with a reference date of *20210101* means that the time coordinate will be weeks since 2021-01-01. The default value is the simulation's start date.

ref_time *[optional]* The reference time from which the frequency is based. The format is `HHMMSS`. The default value is `000000`. See `ref_date`.

fields Defines the list of fields that this collection should use. The format (per-field) is `'FieldName', 'GridCompName',`. For example, `'SpeciesConc_O3', 'GCHPchem',` specifies that this collection should include the *SpeciesConc_O3* field from the *GCHPchem* gridded component.

Fields from multiple gridded components can be included in the same collection. However, a collection must not mix fields that are defined at the center of vertical levels and the edges of vertical levels (e.g., *Met_PMID* and *Met_PEDGE* cannot be included in the same collection).

Variables can be renamed in the output by adding `'your_custom_name',` at the end. For example, `'SpeciesConc_O3', 'GCHPchem', 'ozone_concentration',` would rename the SpeciesConc_O3 field to "ozone_concentration" in the output file.

Output grid configuration

grid_label *[optional]* Defines the grid that this collection should be output on. The lable must match on of the grid labels defined in *<DEFINE GRID LABELS>*. If `grid_label` isn't set then the collection uses the simulation's horizontal grid.

conservative *[optional]* Defines whether or not regridding to the output grid should use ESMF's first-order conservative method. Valid values are `0` or `1`. It is recommended you set this to `1` if you are using `grid_label`. The default value is `0`.

levels *[optional]* Defines the model levels that this collection should use (i.e., a subset of the simulation levels). The format is a space-separated list of values. The lowest layer is 1 and the highest layer is 72. For example, `1 2 5` would select the first, second, and fifth level of the simulation.

track_file *[optional]* Defines the path to a 1D track file along which the collection is sampled. See *Output Along a Track* for more info.

recycle_track *[optional]* Only valid if a `track_file` is defined. Specifies that the track file should be reused every day. If `.true.` the dates in the track file are automatically forced to the simulation's current date. The default value is false.

Other configuration

end_date *[optional]* A date at which the collection is deactivated (turned off). By default there is no end date.

end_time *[optional]* Time at which the collection is deactivated (turned off) on the `end_date`.

10.1.4 Example `HISTORY.rc` configuration

Below is an example `HISTORY.rc` that configures two output collection

1. 30-min instantaneous concentrations of O3, NO, NO2, and some meteorological parameters for the lowest 10 model levels on a 0.1°x0.1° covering the US. Each file contains one day of data.
2. 24-hour time averages of O3, NO, and NO2 concentrations, NO emissions, and some meteorological parameters. The horizontal grid is the simulation's grid. All vertical levels are use. Each file contains one week worth of data, and files are generated relative to 2017-01-01.

```
EXPID:  OutputDir/GCHP
EXPDSC: GEOS-Chem_devel
CoresPerNode: 6
VERSION: 1

GRID_LABELS: RegionalGrid_US
::
   RegionalGrid_US.GRID_TYPE: LatLon
   RegionalGrid_US.IM_WORLD:   640
   RegionalGrid_US.JM_WORLD:   290
   RegionalGrid_US.POLE:        XY
   RegionalGrid_US.DATELINE:    XY
   RegionalGrid_US.LON_RANGE: -127 -63
   RegionalGrid_US.LAT_RANGE:   23  52

COLLECTIONS: 'Inst30minGases',
      'DailyAvgGasesAndNOEmissions',
::
Inst30minGases.template:    '%y4%m2%d2_%h2%n2z.nc4',
Inst30minGases.format:      'CFIO',
Inst30minGases.frequency:   003000
Inst30minGases.duration:    240000
Inst30minGases.mode:        'instantaneous'
Inst30minGases.grid_label:  RegionalGrid_US
Inst30minGases.levels:      1 2 3 4 5 6 7 8 9 10 11 12 13 14
Inst30minGases.fields:     'SpeciesConc_O3  ',  'GCHPchem',
                           'SpeciesConc_NO  ',  'GCHPchem',
                           'SpeciesConc_NO2 ',  'GCHPchem',
                           'Met_BXHEIGHT    ',  'GCHPchem',
                           'Met_AIRDEN      ',  'GCHPchem',
                           'Met_AD          ',  'GCHPchem',
                           'Met_PS1WET      ',  'GCHPchem',
::
DailyAvgGasesAndNOEmissions.template:     '%y4%m2%d2_%h2%n2z.nc4',
DailyAvgGasesAndNOEmissions.format:       'CFIO',
DailyAvgGasesAndNOEmissions.ref_date:     20170101
DailyAvgGasesAndNOEmissions.frequency:    240000
DailyAvgGasesAndNOEmissions.duration:    1680000
DailyAvgGasesAndNOEmissions.mode:         'time-averaged'
DailyAvgGasesAndNOEmissions.fields:       'SpeciesConc_O3  ',  'GCHPchem',
                                          'SpeciesConc_NO  ',  'GCHPchem',
                                          'SpeciesConc_NO2 ',  'GCHPchem',
                                          'EmisNO_Total    ',  'GCHPchem',
                                          'EmisNO_Aircraft ',  'GCHPchem',
                                          'EmisNO_Anthro   ',  'GCHPchem',
                                          'EmisNO_BioBurn  ',  'GCHPchem',
                                          'EmisNO_Lightning',  'GCHPchem',
                                          'EmisNO_Ship     ',  'GCHPchem',
                                          'EmisNO_Soil     ',  'GCHPchem',
                                          'EmisNO2_Anthro  ',  'GCHPchem',
                                          'EmisNO2_Ship    ',  'GCHPchem',
                                          'EmisO3_Ship     ',  'GCHPchem',
                                          'Met_BXHEIGHT    ',  'GCHPchem',
```

(continues on next page)

(continued from previous page)

```
                                        'Met_AIRDEN      ',  'GCHPchem',
                                        'Met_AD          ',  'GCHPchem',
::
```

10.2 HEMCO_Config.rc, HEMCO_Diagn.rc

10.2.1 HEMCO_Config.rc

Like `input.geos`, information about the `HEMCO_Config.rc` file is the same as for GEOS-Chem Classic with a few exceptions. Refer to the HEMCO documentation for an overview of the file.

Some content of the `HEMCO_Config.rc` file is ignored by GCHP. This is because MAPL ExtData handles file input rather than HEMCO in GCHP.

Items at the top of the file that are ignored include:

- ROOT data directory path
- METDIR path
- DiagnPrefix
- DiagnFreq
- Wildcard

In the BASE EMISSIONS section and beyond, columns that are ignored include:

- sourceFile
- sourceVar
- sourceTime
- C/R/E
- SrcDim
- SrcUnit

All of the above information is specified in file `ExtData.rc` instead with the exception of diagnostic prefix and frequency. Diagnostic filename and frequency information is specified in `HISTORY.rc`.

10.2.2 HEMCO_Diagn.rc

Like in GEOS-Chem Classic, the `HEMCO_Diagn.rc` file is used to map between HEMCO containers and output file diagnostic names. However, while all uncommented diagnostics listed in `HEMCO_Diagn.rc` are output as HEMCO diagnostics in GEOS-Chem Classic, only the subset also listed in `HISTORY.rc` are output in GCHP. See the HEMCO documentation for an overview of the file.

10.3 ExtData.rc

`ExtData.rc` contains input variable and file read information for GCHP. Explanatory information about the file is located at the top of the configuration file in all run directories. The file format is the same as that used in the GEOS model, and GMAO/NASA documentation for it can be found at the ExtData component page on the GEOS-5 wiki.

The following two parameters are set at the top of the file:

Ext_AllowExtrat Logical toggle to use data from nearest year available. This is set to true for GCHP. Note that GEOS-Chem Classic accomplishes the same effect but with more flexibility in `HEMCO_Config.rc`. That functionality of `HEMCO_Config.rc` is ignored in GCHP.

DEBUG_LEVEL Turns MAPL ExtData debug prints on/off. This is set to 0 in GCHP (off), but may be set to 1 to enable. Beware that turning on ExtData debug prints greatly slows down the model, and prints are only done from the root thread. Use this when debugging problems with input files.

The rest of the file contains space-delimited lines, one for each variable imported to the model from an external file. Columns are as follows in order as they appear left to right in the file:

Export Name Name of imported met field (e.g. ALBD) or HEMCO emissions container name (e.g. GEIA_NH3_ANTH).

Units Unit string nested within single quotes. '1' indicates there is no unit conversion from the native units in the netCDF file.

Clim Enter Y if the file is a 12 month climatology, otherwise enter N. If you specify it is a climatology ExtData the data can be on either one file or 12 files if they are templated appropriately with one per month.

Conservative Enter Y the data should be regridded in a mass conserving fashion through a tile file. `F;{VALUE}` can also be used for fractional regridding. Otherwise enter N to use the non-conervative bilinear regridding.

Refresh Time Template Possible values include:

- -: The field will only be updated once the first time ExtData runs
- 0: Update the variable at every step. ExtData will do a linear interpolation to the current time using the available data.
- %y4-%m2-%h2T%h2:%n2:00: Set the recurring time to update the file. The file will be updated when the evaluated template changes. For example, a template in the form %y4-%m2-%d2T12:00:00 will cause the variable to be updated at the start of a new day (i.e. when the clock hits 2007-08-02T00:00:00 it will update the variable but the time it will use for reading and interpolation is 2007-08-02T12:00:00).

Offset Factor Factor the variable will be shifted by. Use none for no shifting.

Scale Factor Factor the variable will be scaled by. Use none for no scaling.

External File Variable The name of the variable in the netCDF data file, e.g. ALBEDO in met fields.

External File Template Path to the netCDF data file. If not using the data, specify `/dev/null` to reduce processing time. If there are no tokens in the template name ExtData will assume that all the data is on one file. Note that if the data on file is at a different resolution that the application grid, the underlying I/O library ExtData uses will regrid the data to the application grid.

10.4 input.geos

Information about the `input.geos` file is the same as for GEOS-Chem Classic with a few exceptions. See the `input.geos` file wiki page for an overview of the file.

The `input.geos` file used in GCHP is different in the following ways:

- Start/End datetimes are ignored. Set this information in `CAP.rc` instead.
- Root data directory is ignored. All data paths are specified in `ExtData.rc` instead with the exception of the FAST-JX data directory which is still listed (and used) in `input.geos`.
- Met field is ignored. Met field source is described in file paths in `ExtData.rc`.
- GC classic timers setting is ineffectual. GEOS-Chem Classic timers code is not compiled when building GCHP.

Other parts of the GEOS-Chem Classic `input.geos` file that are not relevant to GCHP are simply not included in the file that is copied to the GCHP run directory.

10.5 CAP.rc, GCHP.rc, input.nml

10.5.1 CAP.rc

`CAP.rc` is the configuration file for the top-level gridded component called CAP. This gridded component can be thought of as the primary driver of GCHP. Its config file handles general runtime settings for GCHP including time parameters, performance profiling routines, and system-wide timestep (hearbeat). Combined with output file `cap_restart`, `CAP.rc` configures the exact dates for the next GCHP run.

ROOT_NAME Sets the name MAPL uses to initialize the ROOT child gridded component component within CAP. CAP uses this name in all operations when querying and interacting with ROOT. It is set to GCHP.

ROOT_CF Resource configuration file for the ROOT component. It is set to `GCHP.rc`.

HIST_CF Resource configuration file for the MAPL HISTORY gridded component (another child gridded component of CAP). It is set to `HISTORY.rc`.

BEG_DATE Simulation begin date in format YYYYMMDD hhmmss. This parameter is overrided in the presence of output file `cap_restart` containing a different start date.

END_DATE Simulation end date in format YYYYMMDD hhmmss. If BEG_DATE plus duration (JOB_SGMT) is before END_DATE then simulation will end at BEG_DATE + JOB_SGMT. If it is after then simulation will end at END_DATE.

JOB_SGMT Simulation duration in format YYYYMMDD hhmmss. The duration must be less than or equal to the difference between start and end date or the model will crash.

HEARTBEAT_DT The timestep of the ESMF/MAPL internal clock, in seconds. All other timesteps in GCHP must be a multiple of HEARTBEAT_DT. ESMF queries all components at each heartbeat to determine if computation is needed. The result is based upon individual component timesteps defined in `GCHP.rc`.

MAPL_ENABLE_TIMERS Toggles printed output of runtime MAPL timing profilers. This is set to YES. Timing profiles are output at the end of every GCHP run.

MAPL_ENABLE_MEMUTILS Enables runtime output of the programs' memory usage. This is set to YES.

PRINTSPEC Allows an abbreviated model run limited to initializat and print of Import and Export state variable names. Options include:

- 0 (default): Off

- 1: Imports and Exports only
- 2: Imports only
- 3: Exports only

USE_SHMEM This setting is deprecated but still has an entry in the file.

REVERSE_TIME Enables running time backwards in CAP. Default is 0 (off).

10.5.2 GCHP.rc

`GCHP.rc` is the resource configuration file for the ROOT component within GCHP. The ROOT gridded component includes three children gridded components, including one each for GEOS-Chem, FV3 advection, and the data utility environment needed to support them.

NX, NY Number of grid cells in the two MPI sub-domain dimensions. NX * NY must equal the number of CPUs. NY must be a multiple of 6.

GCHP.GRID_TYPE Type of grid GCHP will be run at. Should always be Cubed-Sphere.

GCHP.GRIDNAME Descriptive grid label for the simulation. The default grid name is PE24x144-CF. The grid name includes how the pole is treated, the face side length, the face side length times six, and whether it is a Cubed Sphere Grid or Lat/Lon. The name PE24x144-CF indicates polar edge (PE), 24 cells along one face side, 144 for 24*6, and a cubed-sphere grid (CF). Many options here are defined in MAPL_Generic.

Note: Must be consistent with IM and JM.

GCHP.NF Number of cubed-sphere faces. This is set to 6.

GCHP.IM_WORLD Number of grid cells on the side of a single cubed sphere face.

GCHP.IM Number of grid cells on the side of a single cubed sphere face.

GCHP.JM Number of grid cells on one side of a cubed sphere face, times 6. This represents a second dimension if all six faces are stacked in a 2-dimensional array. Must be equal to IM*6.

GCHP.LM Number of vertical grid cells. This must be equal to the vertical resolution of the offline meteorological fields (72) since MAPL cannot regrid vertically.

GCHP.STRETCH_FACTOR Ratio of configured global resolution to resolution of targeted high resolution region if using stretched grid.

GCHP.TARGET_LON Target longitude for high resolution region if using stretched grid.

GCHP.TARGET_LAT Target latitude for high resolution region if using stretched grid.

IM Same as GCHP.IM and GCHP.IM_WORLD.

JM Same as GCHP.JM.

LM Same as GCHP.LM.

GEOChem_CTM If set to 1, tells FVdycore that it is operating as a transport model rather than a prognostic model.

AdvCore_Advection Toggles offline advection. 0 is off, and 1 is on.

DYCORE Should either be set to OFF (default) or ON. This value does nothing, but MAPL will crash if it is not declared.

HEARTBEAT_DT The timestep in seconds that the DYCORE Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

SOLAR_DT The timestep in seconds that the SOLAR Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

IRRAD_DT The timestep in seconds that the IRRAD Component should be called. ESMF checks this value during its timestep check. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

RUN_DT The timestep in seconds that the RUN Component should be called.

GCHPchem_DT The timestep in seconds that the GCHPchem Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

RRTMG_DT The timestep in seconds that RRTMG should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

DYNAMICS_DT The timestep in seconds that the FV3 advection Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

SOLARAvrg, IRRADAvrg Default is 0.

GCHPchem_REFERENCE_TIME HHMMSS reference time used for GCHPchem MAPL alarms.

PRINTRC Specifies which resource values to print. Options include 0: non-default values, and 1: all values. Default setting is 0.

PARALLEL_READFORCING Enables or disables parallel I/O processes when writing the restart files. Default value is 0 (disabled).

NUM_READERS, NUM_WRITERS Number of simultaneous readers. Should divide evenly unto NY. Default value is 1.

BKG_FREQUENCY Active observer when desired. Default value is 0.

RECORD_FREQUENCY Frequency of periodic restart file write in format HHMMSS.

RECORD_REF_DATE Reference date(s) used to determine when to write periodic restart files.

RECORD_REF_TIME Reference time(s) used to determine when to write periodic restart files.

GCHOchem_INTERNAL_RESTART_FILE The filename of the internal restart file to be written.

GCHPchem_INTERNAL_RESTART_TYPE The format of the internal restart file. Valid types include pbinary and pnc4. Only use pnc4 with GCHP.

GCHPchem_INTERNAL_CHECKPOINT_FILE The filename of the internal checkpoint file to be written.

GCHPchem_INTERNAL_CHECKPOINT_TYPE The format of the internal checkstart file. Valid types include pbinary and pnc4. Only use pnc4 with GCHP.

GCHPchem_INTERNAL_HEADER Only needed when the file type is set to pbinary. Specifies if a binary file is self-describing.

DYN_INTERNAL_RESTART_FILE The filename of the DYNAMICS internal restart file to be written. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_RESTART_TYPE The format of the DYNAMICS internal restart file. Valid types include pbinary and pnc4. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_CHECKPOINT_FILE The filename of the DYNAMICS internal checkpoint file to be written. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_CHECKPOINT_TYPE The format of the DYNAMICS internal checkpoint file. Valid types include pbinary and pnc4. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_HEADER Only needed when the file type is set to pbinary. Specifies if a binary file is self-describing.

RUN_PHASES GCHP uses only one run phase. The GCHP gridded component for chemistry, however, has the capability of two. The two-phase feature is used only in GEOS.

HEMCO_CONFIG Name of the HEMCO configuration file. Default is `HEMCO_Config.rc` in GCHP.

STDOUT_LOGFILE Log filename template. Default is `PET%%%%%.GEOSCHEMchem.log`. This file is not actually used for primary standard output.

STDOUT_LOGLUN Logical unit number for stdout. Default value is 700.

MEMORY_DEBUG_LEVEL Toggle for memory debugging. Default is 0 (off).

WRITE_RESTART_BY_OSERVER Determines whether MAPL restart write should use o-server. This must be set to YES for high core count (>1000) runs to avoid hanging during file write. It is NO by default.

10.5.3 input.nml

input.nml controls specific aspects of the FV3 dynamical core used for advection. Entries in input.nml are described below.

&fms_nml Header for the FMS namelist which includes all variables directly below the header.

print_memory_usage Toggles memory usage prints to log. However, in practice turning it on or off does not have any effect.

domain_stack_size Domain stack size in bytes. This is set to 20000000 in GCHP to be large enough to use very few cores in a high resolution run. If the domain size is too small then you will get an "mpp domain stack size overflow error" in advection. If this happens, try increasing the domain stack size in this file.

&fv_core_nml Header for the finite-volume dynamical core namelist. This is commented out by default unless running on a stretched grid. Due to the way the file is read, commenting out the header declaration requires an additional comment character within the string, e.g. `#&fv#_core_nml`.

do_schmidt Logical for whether to use Schmidt advection. Set to .true. if using stretched grid; otherwise this entry is commented out.

stretch_fac Stretched grid factor, equal to the ratio of grid resolution in targeted high resolution region to the configured run resolution. This is commented out if not using stretched grid.

target_lat Target latitude of high resolution region if using stretched grid. This is commented out if not using stretched grid.

target_lon Target longitude of high resolution region if using stretched grid. This is commented out if not using stretched grid.

Much of the labor of updating the configuration files has been eliminated by run directory shell script `runConfig.sh`. It is important to remember that sourcing `runConfig.sh` will overwrite settings in other configuration files, and you therefore should never manually update other configuration files unless you know the specific option is not available in `runConfig.sh`.

That being said, it is still worth understanding the contents of all configuration files and what all run options include. This page details the settings within all configuration files and what they are used for.

10.6 File descriptions

The following table lists the core functions of each of the configuration files in the GCHP run directory. See the individual subsections on each file for additional information.

CAP.rc Controls parameters used by the highest level gridded component (CAP). This includes simulation run time information, name of the Root gridded component (GCHP), config filenames for Root and History, and toggles for certain MAPL logging utilities (timers, memory, and import/export name printing).

ExtData.rc Config file for the MAPL ExtData component. Specifies input variable information, including name, regridding method, read frequency, offset, scaling, and file path. All GCHP imports must be specified in this file. Toggles at the top of the file enable MAPL ExtData debug prints and using most recent year if current year of data is unavailable. Default values may be used by specifying file path `/dev/null`.

GCHP.rc Controls high-level aspects of the simulation, including grid type and resolution, core distribution, stretched-grid parameters, timesteps, and restart file configuration.

input.geos Primary config file for GEOS-Chem. Same function as in GEOS-Chem Classic except grid, simulation start/end, met field source, timers, and data directory information are ignored.

HEMCO_Config.rc Contains emissions information used by HEMCO. Same function as in GEOS-Chem Classic except only HEMCO name, species, scale IDs, category, and hierarchy are used. Diagnostic frequency, file path, read frequency, and units are ignored, and are instead stored in GCHP config file `ExtData.rc`. All HEMCO variables listed in `HEMCO_Config.rc` for enabled emissions must also have an entry in `ExtData.rc`.

HEMCO_Diagn.rc Contains information mapping `HISTORY.rc` diagnostic names to HEMCO containers. Same function as in GEOS-Chem Classic except that not all items in `HEMCO_Diagn.rc` will be output; only emissions listed in `HISTORY.rc` will be included in diagnostics. All GCHPctm diagnostics listed in `HISTORY.rc` that start with Emis, Hco, or Inv must have a corresponding entry in `HEMCO_Diagn.rc`.

input.nml Namelist used in advection for domain stack size and stretched grid parameters.

HISTORY.rc Config file for the MAPL History component. Configures diagnostic output from GCHP.

CHAPTER

ELEVEN

EXAMPLE JOB SCRIPTS

These are example job scripts for GCHP batch jobs. These examples are taken from the `runScriptSamples/`. See that directory for more information and examples.

Important: These are examples. You need to write your own job scripts, but these are good templates to start from.

Please share yours! Submit a pull-request on GitHub.

11.1 Examples for Various Schedulers

These are simple examples for various schedulers. They are set up to use 2 nodes, and are suitable for C48 or C90 resolution.

- For PBS-based clusters: `simple_batch_job.pbs.sh`
- For Slurm-based clusters: `simple_batch_job.slurm.sh`
- For LSF-based clusters: `simple_batch_job.lsf.sh`

11.2 Examples for Various HPCs

These are simple examples for various systems. They are set up to use 2 nodes, and are suitable for C48 or C90 resolution.

- For Pleiades (NASA Advanced Supercomputing): `simple_batch_job.pbs.sh`
- For Cannon (Harvard): `simple_batch_job.slurm.sh`
- For Compute1 (WUSTL): `simple_batch_job.lsf.sh`

11.3 Operational Examples

These are "full-fledged" examples. They are more complicated, but they demonstrate what operational GCHP batch jobs look like. Initially, it's probably best to err on the side of simplicity, and build your own automated functionality with time.

- Auto-requeuing C360 simulation (Compute1): `c360_requeuing.sh`
- 1 month benchmark simulation (Cannon): `gchp.benchmark.run`

CHAPTER

TWELVE

BUILDING DEPENDENCIES WITH SPACK

The Spack Package Manager may be used to download and build GCHP and all of its required external software depenencies, including CMake, MPI, ESMF, and NetCDF libraries. The only essential prerequisite for using Spack is a local C/C++/Fortran Compiler such as GNU Compiler Collection. You can use this local compiler to later install a different compiler version through Spack. **GCHP requires GNU Compilers version 8.3 or Intel compilers version 18.0.5**. You must install a newer compiler through Spack if your pre-installed compiler does not meet these requirements.

There are 3 stages to setting up GCHP and/or its dependencies with Spack: setting up Spack and configuring existing libraries, installing GCHP and/or its dependencies, and setting up an environment file for loading installed Spack packages.

All Spack users should first read the section on setting up Spack and configuring Spack to recognize existing dependencies on your system that you would like Spack to use for building GCHP or other dependencies (such as ESMF or Slurm):

- *Setting up Spack and existing dependencies*

You can optionally read about the most important Spack commands and tips on how they work:

- *Spack syntax and handy commands*

Read one of the following sections depending on which portions of the GCHP software stack you want to install using Spack:

- *Install individual software dependencies*
- *Install all dependencies in one command*
- *Install both GCHP and all of its dependencies in one command*

Read how to load Spack libraries in an environment file for running or building GCHP or ESMF:

- *Loading Spack libraries for use with GCHP and/or ESMF*

12.1 Setting up Spack, installing new compilers, and specifying pre-installed dependencies

To begin using Spack, clone the latest version by typing `git clone https://github.com/spack/spack.git`. Execute the following commands to initialize Spack's environment (replacing `/path/to/spack` with the path of your *spack* directory). Add these commands to an environment initialization script for easy reuse.

```
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh
```

If you do not already have a copy of your preferred text editor, you can use Spack to install and load one before proceeding (e.g. `spack install emacs; spack load emacs`).

12.1.1 Installing compilers

Ensure Spack recognizes any existing compiler on your system by typing `spack compilers`. You can use this compiler to build a new one.

Setting up and using GNU compilers

GNU compilers are free, open source, and easily installable through Spack. Execute the following at the command prompt to install version 9.3.0 of the GNU Compiler Collection:

```
$ spack install gcc@9.3.0
```

The installation of `gcc` may take a long time. Once it is complete, you'll need to add it to Spack's list of compilers using the following command:

```
$ spack compiler find $(spack location -i gcc@9.3.0)
```

Make sure that spack recognizes your new compiler by typing `spack compilers`, which should display a list of compilers including GCC 9.3.0.

Setting up and using Intel compilers

If you would like to use Intel compilers, whether pre-installed on your system or which you would like to install through Spack using an existing license, follow the instructions for pre-installed Intel compilers or for Intel compilers you want to install through Spack on the official Spack documentation site. In the instructions below, simply replace `%gcc@9.3.0` with `%intel` to use your Intel compilers to build libraries.

12.1.2 Specifying preinstalled dependencies

Just as Spack can use an existing compiler on your system, it can also use your existing installations of dependencies for GCHP rather than building new copies. **This is essential for interfacing your cluster's Slurm with a Spack-built GCHP and its dependencies**. For any preinstalled dependency you want Spack to always use, you must specify its path on your system and that you want Spack to always use this preinstalled package rather than building a new version. The code below shows how to do this by editing `$HOME/.spack/packages.yaml` (you may need to create this file):

```
packages:
 slurm:
  paths:
   slurm: /path/to/slurm
  buildable: False
```

12.1.3 Finding and specifying your system's Slurm installation

If you plan on submitting jobs through the Slurm job scheduler, you'll need to specify the location of Slurm on your system for Spack. Spack expects a specific directory format for your external Slurm installation path: it must contain both an `include/` directory and a `lib64/` directory. Depending on your cluster's Slurm configuration, these directories may or may not already be in the same parent directory. Additionally, finding these individual directories can prove challenging. Each of the following commands can help lead you to the correct `include/` and `lib64/` directories on your cluster:

```
$ whereis slurm
$ whereis libpmi
$ whereis libpmi2
$ whereis srun
$ whereis sinfo
```

You may or may not receive any output from each of these commands, but hopefully at least one of these commands reveals a high level Slurm directory (e.g. `opt/slurm`), an `include/` directory, and/or a `lib64/` directory. You may encounter multiple directories with the name `include` or `lib64`; the correct `include/` directory should contain `.h` files like `pmi.h` and `slurm.h`, while the `lib64/` directory should contain `libpmi.so`, `libpmi2.so`, and/or `libpmix.so`.

If you have a high level Slurm directory that contains correct `include/` and `lib64/` directories, then you can use the path to that high level directory in your `$HOME/.spack/packages.yaml` file.

If your revealed `include/` and `lib64/` directories are not located in the same parent directory, you'll need to create a new directory (called `slurm_hub` in this example, though you can name it anything and put it anywhere) with symlinks to your `include/` and `lib64/` directories:

```
$ mkdir slurm_hub
$ cd slurm_hub
$ ln -s /path/to/include include
$ ln -s /path/to/lib64 lib64
```

Put the path to `slurm_hub` in the correct location of your `$HOME/.spack/packages.yaml` file, and Spack will consider this directory to be your Slurm directory.

12.2 Spack syntax and handy commands

This section describes the most important parts of Spack's syntax as well as various useful Spack commands.

When describing a Spack package, the `@version` syntax specifies the package version (e.g. `gchp@13.0.2`). If you do not specify a specific version of a package during installation, Spack will either install the latest available version of that package or Spack will choose a version that it knows to be compatible with other dependencies in your software stack.

The `package1 ^package2` syntax tells Spack that `package2` is a dependency of `package1`. Packages will often have locked dependency requirements such that you cannot add totally new packages as dependencies or avoid using specific packages, but you can still use this syntax to specify information on how to build certain dependencies (e.g. `spack install gchp ^netcdf-fortran@4.5.2%gcc@9.3.0`). Other dependency requirements can be fulfilled by multiple packages (e.g. you can specify an MPI implementation to use through `spack install gchp ^openmpi` or `spack install gchp ^intel-mpi`). If you want to specify build options for a package, make sure that option immediately follows the package's name in a Spack command (e.g. `spack install gchp +ofi ^openmpi +pmi` is correct because `+ofi` is an option for `gchp` and `+pmi` is an option for `openmpi`, but `spack install gchp ^openmpi +pmi +ofi` will fail because `+ofi` is not an option for `openmpi`).

The `%compiler` syntax specifies which compiler to use to build a package. You can specify the compiler version as well (e.g. `gchp%gcc@9.3.0`). Spack should build all child dependencies of a package with the same compiler you specify for the parent package, but compiler versions may or may not be consistent if you first specify a compiler version further down the dependency tree (e.g. `spack install gchp%gcc@9.3.0 ^openmpi` will build all dependencies of GCHP with `gcc@9.3.0` including OpenMPI, but `spack install gchp ^openmpi%gcc@9.3.0` only requires Spack to build OpenMPI and its dependencies with `gcc@9.3.0` so other dependencies of GCHP may be built with another compiler version).

Every build of a package through Spack receives its own unique hash identifier. This hash can be useful when you install multiple copies of the same package that are not readily distinguishable (e.g. they only differ by one dependency version). You can use the `-l` option in `spack find` to see the beginning of the hash ID for each package. You only need this short code (rather than the full 32 character hash) to identify a package for Spack. Use `/hash_id` to specify a package build based on its hash (you don't need to include the package's name, e.g. a GCHP installation with the hash `2qmcc6kq3hwm3vasey63y6h4l77pzw2o` can be loaded using `spack load /2qmcc6`).

All of the above syntax examples can be added to any of the commands below for any `package_name`.

`spack info package_name` will return information on package `package_name`, including available versions and build variant options, if this package is available through Spack. This info is pulled from data in your clone of the Spack Git repository. You can get the latest information (as well as any updates to packages) by updating your Spack repo using `git pull`.

`spack spec package_name` will return a list of all dependencies (including package versions, compilers, and build variants) that would be installed if you were to use `spack install package_name`.

`spack install package_name` is the core Spack installation command. Many examples of `install` commands are found in other sections on this page. One important option for `spack install` is `--only dependencies`, which installs all dependencies for a package without installing the package itself. Note that if you use `spack install --only dependencies package_name` you will not be able to later load all dependencies for `package_name` using `spack load package_name`; you will have to load dependencies individually.

`spack uninstall package_name` lets you uninstall specific packages. You'll be prompted with additional options to append after `uninstall` if the package you're attempting to remove has dependents remaining on your system. `spack uninstall -a pattern` can be used to uninstall all packages matching a certain spec pattern (e.g. `spack uninstall %gcc@9.3.0` will uninstall all packages that were compiled with `gcc@9.3.0`).

`spack compiler remove compiler_name` will remove a compiler from Spack's list of available compilers.

`spack find package_name` will show you all installed versions of `package_name`. `spack find` will show you all installed packages, and `spack find --loaded` will show you all loaded packages.

`spack location -i package_name` will return the installation path of `package_name`.

`spack load package_name` will load a package and all of its runtime dependencies.

`spack unload package_name` will unload a specific package. Note that some environment variable changes will not be undone, and dependencies will not be unloaded. `spack unload` (without specifying a package) will unload all loaded packages, similar to `module purge`.

12.3 Installing individual dependencies with Spack

This section describes how to use Spack to build GCHP's individual dependencies. While these dependencies can be used to then install GCHP directly using Spack, this section is mainly intended for those looking to manually download and compile GCHP as described in the User Guide.

12.3.1 Installing basic dependencies

You should now install Git and CMake using Spack:

```
$ spack install git@2.17.0%gcc@9.3.0
$ spack install cmake@3.16.1%gcc@9.3.0
```

Installing without Slurm support

If you do not intend to use a job scheduler like Slurm to run GCHP, use the following commands to install MPI and NetCDF-Fortran. Otherwise, scroll down to see necessary modifications you must make to include Slurm support.

OpenMPI

```
$ spack install openmpi@4.0.4%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^openmpi@4.0.4
```

Intel MPI

```
$ spack install intel-mpi%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^intel-mpi
```

MVAPICH2

```
$ spack install mvapich2%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^mvapich2
```

Installing with Slurm support

OpenMPI

You need to tell Spack to build OpenMPI with Slurm support and to build NetCDF-Fortran with the correct OpenMPI version as a dependency:

```
$ spack install openmpi@4.0.4%gcc@9.3.0 +pmi schedulers=slurm
$ spack install netcdf-fortran%gcc@9.3.0  ^netcdf-c^hdf5^openmpi@4.0.4+pmi␣
↪schedulers=slurm
```

You may run into issues building OpenMPI if your cluster has preexisting versions of PMIx that are newer than OpenMPI's internal version. OpenMPI will search for and use the newest version of PMIx installed on your system, which will likely cause a crash during build because OpenMPI requires you to build with the same libevent library as was used to build PMIx. This information may not be readily available to you, in which case you can tweak the build arguments for OpenMPI to always use OpenMPI's internal version of PMIx. Open `$SPACK_ROOT/var/spack/repos/builtin/packages/openmpi/package.py` and navigate to the `configure_args()` function. In the body of this function, place the following line:

```
config_args.append('--with-pmix=internal')
```

Intel MPI

No build-time tweaks need to be made to install Intel MPI with Slurm support.

```
$ spack install intel-mpi%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^intel-mpi
```

Scroll down to find environment variables you need to set when running GCHP with Intel MPI, including when using Slurm.

MVAPICH2

Like OpenMPI, you must specify that you want to build MVAPICH2 with Slurm support and build NetCDF-Fortran with the correct MVAPICH2 version.

```
$ spack install mvapich2%gcc@9.3.0 process_managers=slurm
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^mvapich2
```

Once you've installed all of your dependencies, you can follow the GCHP instructions for downloading, compiling, and setting up a run directory in the User Guide section of this Read The Docs site.

12.4 One-line install of GCHP dependencies with Spack

Rather than using Spack to install individual dependencies, you can use the `spack install --only dependencies gchp` command to install every dependency for GCHP in a single command. The `--only dependencies` option tells Spack to build GCHP's dependencies without building GCHP itself.

Spack is smart about choosing compatible versions for all of GCHP's different dependencies. You can further specify which package versions or MPI implementations (OpenMPI, Intel MPI, or MVAPICH2) you wish to use by appending options to `spack install --only dependencies gchp`, such as `^openmpi@4.0.4` or `^intel-mpi`. If you wish to use Slurm with GCHP and want Spack to install a new version of OpenMPI or MVAPICH2, you need to specify `+pmi schedulers=slurm` (for OpenMPI) or `process_managers=slurm` (for MVAPICH2). A full install line for all of GCHP's dependencies, including Slurm-enabled OpenMPI, would look like `spack install --only dependencies gchp ^openmpi +pmi schedulers=slurm`.

Once you've installed all of your dependencies, you can follow the GCHP instructions for downloading, compiling, and setting up a run directory in the User Guide section of this Read The Docs site.

12.5 One-line install of GCHP and its dependencies with Spack

You can use Spack to install all of GCHP's dependencies and GCHP itself in a single line: `spack install gchp`. Just as when installing only GCHP's dependencies, you can modify this command with further options for GCHP's dependencies (and should do so if you intend to use a job scheduler like Slurm).

Spack is smart about choosing compatible versions for all of GCHP's different dependencies. You can further specify which package versions or MPI implementations (OpenMPI, Intel MPI, or MVAPICH2) you wish to use by appending options to `spack install gchp`, such as `^openmpi@4.0.4` or `^intel-mpi`. If you wish to use Slurm with GCHP and want Spack to install a new version of OpenMPI or MVAPICH2, you need to specify `+pmi schedulers=slurm` (for OpenMPI) or `process_managers=slurm` (for MVAPICH2). A full install line for GCHP and all of its dependencies, including Slurm-enabled OpenMPI, would look like `spack install gchp ^openmpi +pmi schedulers=slurm`.

In addition to specifying options for GCHP's dependencies, GCHP also has its own options you can specify in your `spack install gchp` command. The available options (which you can view for yourself using `spack info gchp`) include:

- `apm` - APM Microphysics (Experimental) (Default: off)
- `build_type` - Choose CMake build type (Default: RelWithDebInfo)
- `ipo` - CMake interprocedural optimization (Default: off)
- `luo` - Luo et al 2019 wet deposition scheme (Default: off)
- `omp` - OpenMP parallelization (Default: off)
- `real8` - REAL*8 precision (Default: on)
- `rrtmg` - RRTMG radiative transfer model (Default: off)
- `tomas` - TOMAS Microphysics (Experimental) (Default: off)

To specify any of these options, place it directly after `gchp` with a + to enable it or a ~ to disable it (e.g. `spack install gchp ~real8 +rrtmg`).

When you run `spack install gchp`, Spack will build all of GCHP's dependencies and then download and build GCHP itself. The overall process may take a very long time if you are installing fresh copies of many dependencies, particularly MPI or ESMF. Once the install is completed, Spack will leave you with a built `gchp` executable and a copy of GCHP's source code at `spack location -i gchp`.

You can use Spack's included copy of the source code to create a run directory. Navigate to the directory returned by `spack location -i gchp`, and then `cd` to `source_code/run`. Run `./createRunDir.sh` to generate a GCHP run directory. Once you've created a run directory, follow the instructions on Running GCHP in the User Guide.

You can find information on loading your environment for running GCHP below.

12.6 Loading Spack libraries for use with GCHP and/or ESMF

After installing the necessary libraries, place the following in a script that you will run before building/running GCHP (such as `$HOME/.bashrc` or a separate environment script) to initialize Spack and load requisite packages for building ESMF and/or building/running GCHP.

OpenMPI

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#===============================================================================
%%%%% Load Spackages %%%%%
#===============================================================================
# List each Spack package that you want to load
# NOTE: Only needed if you did not install GCHP directly through Spack
pkgs=(gcc@9.3.0       \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 openmpi@4.0.4        \
 esmf@8.0.1           )
```

(continues on next page)

(continued from previous page)

```
# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

# If you installed GCHP directly through Spack,comment out the above code after "Load␣
↪Spackages"
# and uncomment the following line
#spack load gchp

export MPI_ROOT=$(spack location -i openmpi)

# These lines only needed for building ESMF outside of Spack
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=openmpi
```

Intel MPI

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#==============================================================================
%%%%% Load Spackages %%%%%
#==============================================================================
# List each Spack package that you want to load
# NOTE: Only needed if you did not install GCHP directly through Spack
pkgs=(gcc@9.3.0       \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 intel-mpi            \
 esmf                 )

# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

# If you installed GCHP directly through Spack,comment out the above code after "Load␣
↪Spackages"
# and uncomment the following line
#spack load gchp

# Environment variables only needed for Intel MPI
export I_MPI_CC=gcc #icc for intel compilers
export I_MPI_CXX=g++ #icpc for intel compilers
export I_MPI_FC=gfortran #ifort for intel compilers
```

(continues on next page)

(continued from previous page)

```
export I_MPI_F77=gfortran #ifort for intel compilers
export I_MPI_F90=gfortran #ifort for intel compilers
export MPI_ROOT=$(spack location -i intel-mpi)

export I_MPI_PMI_LIBRARY=/path/to/slurm/libpmi2.so #when using srun through Slurm
#unset I_MPI_PMI_LIBRARY #when using mpirun

# These lines only needed for building ESMF outside of Spack
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=intelmpi
```

MVAPICH2

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#==============================================================================
%%%%% Load Spackages %%%%%
#==============================================================================
# List each Spack package that you want to load
# NOTE: Only needed if you did not install GCHP directly through Spack
pkgs=(gcc@9.3.0       \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 mvapich2             \
 esmf                 )

# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

# If you installed GCHP directly through Spack,comment out the above code after "Load␣
↪Spackages"
# and uncomment the following line
#spack load gchp

export MPI_ROOT=$(spack location -i mvapich2)

# These lines only needed for building ESMF outside of Spack
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=mvapich2
```

You can also add other packages you've installed with Spack like `emacs` to the `pkgs` lists above.

12.7 ESMF and your environment file

The following gives some information on building ESMF separately from Spack and provides more environment file examples.

You must load your environment file prior to building and running GCHP.

```
$ source /home/envs/gchpctm_ifort18.0.5_openmpi4.0.1.env
```

If you don't already have ESMF 8.0.0+, you will need to download and build it. You only need to build ESMF once per compiler and MPI configuration (this includes for ALL users on a cluster!). It is therefore worth downloading and building somewhere stable and permanent, as almost no users of GCHP would be expected to need to modify or rebuild ESMF except when adding a new compiler or MPI. ESMF is available through Spack, and will already be installed if you chose the `spack install gchp --only dependencies` or `spack install gchp` routes above. Instructions for manually downloading and building ESMF are available at the GCHP wiki.

It is good practice to store your environment setup in a text file for reuse. Below are a couple examples that load libraries and export the necessary environment variables for building and running GCHP. Note that library version information is included in the filename for easy reference. Be sure to use the same libraries that were used to create the ESMF build install directory stored in environment variable `ESMF_ROOT`.

Environment file example 1

```
# file: gchpctm_ifort18.0.5_openmpi4.0.1.env

# Start fresh
module --force purge

# Load modules (some include loading other libraries such as netcdf-C and hdf5)
module load intel/18.0.5
module load openmpi/4.0.1
module load netcdf-fortran/4.5.2
module load cmake/3.16.1

# Set environment variables
export CC=gcc
export CXX=g++
export FC=ifort

# Set location of ESMF
export ESMF_ROOT=/n/lab_shared/libraries/ESMF/ESMF_8_0_1/INSTALL_ifort18_openmpi4
```

Environment file example 2 (Spack libraries built with a pre-installed compiler)

```
# file: gchpctm_gcc7.4_openmpi.rc

# Start fresh
module --force purge

# Load modules
module load gcc-7.4.0
spack load cmake
spack load openmpi%gcc@7.4.0
spack load hdf5%gcc@7.4.0
spack load netcdf%gcc@7.4.0
```

(continues on next page)

(continued from previous page)

```
spack load netcdf-fortran%gcc@7.4.0

# Set environment variables
export CC=gcc
export CXX=g++
export FC=gfortran

# Set location of ESMF
export ESMF_ROOT=/n/home/ESMFv8/DEFAULTINSTALLDIR
```

CHAPTER

THIRTEEN

USING GCHP CONTAINERS

Containers are an effective method of packaging and delivering GCHP's source code and requisite libraries. We offer up-to-date Docker images for GCHP through Docker Hub. These images contain pre-built GCHP source code and the tools for creating a GCHP run directory. The instructions below show how to create a run directory and run GCHP using Singularity , which can be installed using instructions at the previous link or through Spack. Singularity is a container software that is preferred over Docker for many HPC applications due to security issues. Singularity can automatically convert and use Docker images.

13.1 Software requirements

There are only two software requirements for running GCHP using a Singularity container:

- Singularity itself
- An MPI implementation that matches the type and major/minor version of the MPI implementation inside of the container

The current images use OpenMPI 4.0.1 internally, which has been confirmed to work with external installations of OpenMPI 4.0.2-4.0.5.

13.2 Performance

Because we do not include optimized infiniband libraries within the provided Docker images, container-based GCHP is currently not as fast as other setups. Container-based benchmarks on Harvard's Cannon cluster up to 360 cores and c90 (~1x1.25) resolution averaged 15% slower than equivalent non-container runs, and may perform worse at a higher core count and resolution. If this performance hit is not a concern, these containers are the quickest way to setup and run GCHP.

13.3 Setting up and running GCHP using Singularity

Available GCHP images are listed on Docker Hub. The following command pulls the image of GCHP 13.0.2 and converts it to a Singularity image named *gchp.sif* in your current directory.

```
$ singularity pull gchp.sif docker://geoschem/gchp:13.0.2
```

If you do not already have GCHP data directories, create a directory where you will later store data files. We will call this directory *DATA_DIR* and your run directory destination *WORK_DIR* in these instructions. Make sure to replace these names with your actual directory paths when executing commands from these instructions

The following command executes GCHP's run directory creation script. Within the container, your *DATA_DIR* and *WORK_DIR* directories are visible as */ExtData* and */workdir*. Use */ExtData* and */workdir* when asked to specify your ExtData location and run directory target folder, respectively, in the run directory creation prompts.

```
$ singularity exec -B DATA_DIR:/ExtData -B WORK_DIR:/workdir gchp.sif /opt/geos-chem/bin/
↪createRunDir.sh
```

Once the run directory is created, it will be available at *WORK_DIR* on your host machine. `cd` to *WORK_DIR*.

To avoid having to specify the locations of your data and run directories (RUN_DIR) each time you execute a command in the singularity container, we will add these to an environment file called *~/.container_run.rc* and point the *gchp.env* symlink to this environment file. We will also load MPI in this environment file (edit the first line below as appropriate to your system).

```
$ echo "module load openmpi/4.0.3" > ~/.container_run.rc
$ echo "export SINGULARITY_BINDPATH=\"DATA_DIR:/ExtData, RUN_DIR:/rundir\"" >> ~/.
↪container_run.rc
$ ./setEnvironment.sh ~/.container_run.rc
$ source gchp.env
```

We will now move the pre-built *gchp* executable and example run scripts to the run directory.

```
$ rm runScriptSamples #remove broken link
$ singularity exec ../gchp.sif cp /opt/geos-chem/bin/gchp /rundir
$ singularity exec ../gchp.sif cp -rf /gc-src/run/runScriptSamples/ /rundir
```

Before running GCHP in the container, we need to create an execution script to tell the container to load its internal environment before running GCHP. We'll call this script *internal_exec*.

```
$ echo ". /init.rc" > ./internal_exec
$ echo "cd /rundir" >> ./internal_exec
$ echo "./gchp" >> ./internal_exec
$ chmod +x ./internal_exec
```

The last change you need to make to run GCHP in a container is to edit your run script (whether from *runScriptSamples/* or otherwise). Replace the typical execution line in the script (where `mpirun` or `srun` is called) with the following:

```
$ time mpirun singularity exec ../gchp.sif /rundir/internal_exec >> ${log}
```

You can now setup your run configuration as normal using *runConfig.sh* and tweak Slurm parameters in your run script.

If you already have GCHP data directories, congratulations! You've completed all the steps you need to run GCHP in a container. If you still need to download data directories, read on.

13.4 Downloading data directories using GEOS-Chem Classic's dry-run option

GCHP does not currently support automated download of requisite data directories, unlike GEOS-Chem Classic. Luckily we can use a GC Classic container to execute a dry-run that matches the parameters of our GCHP run to download data files.

```
$ #get GC Classic image from https://hub.docker.com/r/geoschem/gcclassic
$ singularity pull gcc.sif docker://geoschem/gcclassic:13.0.0-alpha.13-7-ge472b62
```

(continues on next page)

(continued from previous page)

```
$ #create a GC Classic run directory (GC_CLASSIC_RUNDIR) in WORK_DIR that matches
$ #your GCHP rundir (72-level, standard vs. benchmark vs. transport tracers, etc.)
$ singularity exec -B WORK_DIR:/workdir gcc.sif /opt/geos-chem/bin/createRunDir.sh
$ cd GC_CLASSIC_RUNDIR
$ #get pre-compiled GC Classic executable
$ singularity exec -B .:/classic_rundir ../gcc.sif cp /opt/geos-chem/bin/gcclassic /
↪classic_rundir
```

Make sure to tweak dates of run in input.geos as needed, following info here.

```
$ #create an internal execute script for your container
$ echo ". /init.rc" > ./internal_exec
$ echo "cd /classic_rundir" >> ./internal_exec
$ echo "./gcclassic --dryrun" >> ./internal_exec
$ chmod +x ./internal_exec
$ #run the model, outputting requisite file info to log.dryrun
$ singularity exec -B .:/classic_rundir ../gcc.sif /classic_rundir/internal_exec > log.
↪dryrun
```

Follow instructions here for downloading your relevant data. Note that you will still need a restart file for your GCHP run which will not be automatically retrieved by this download script.

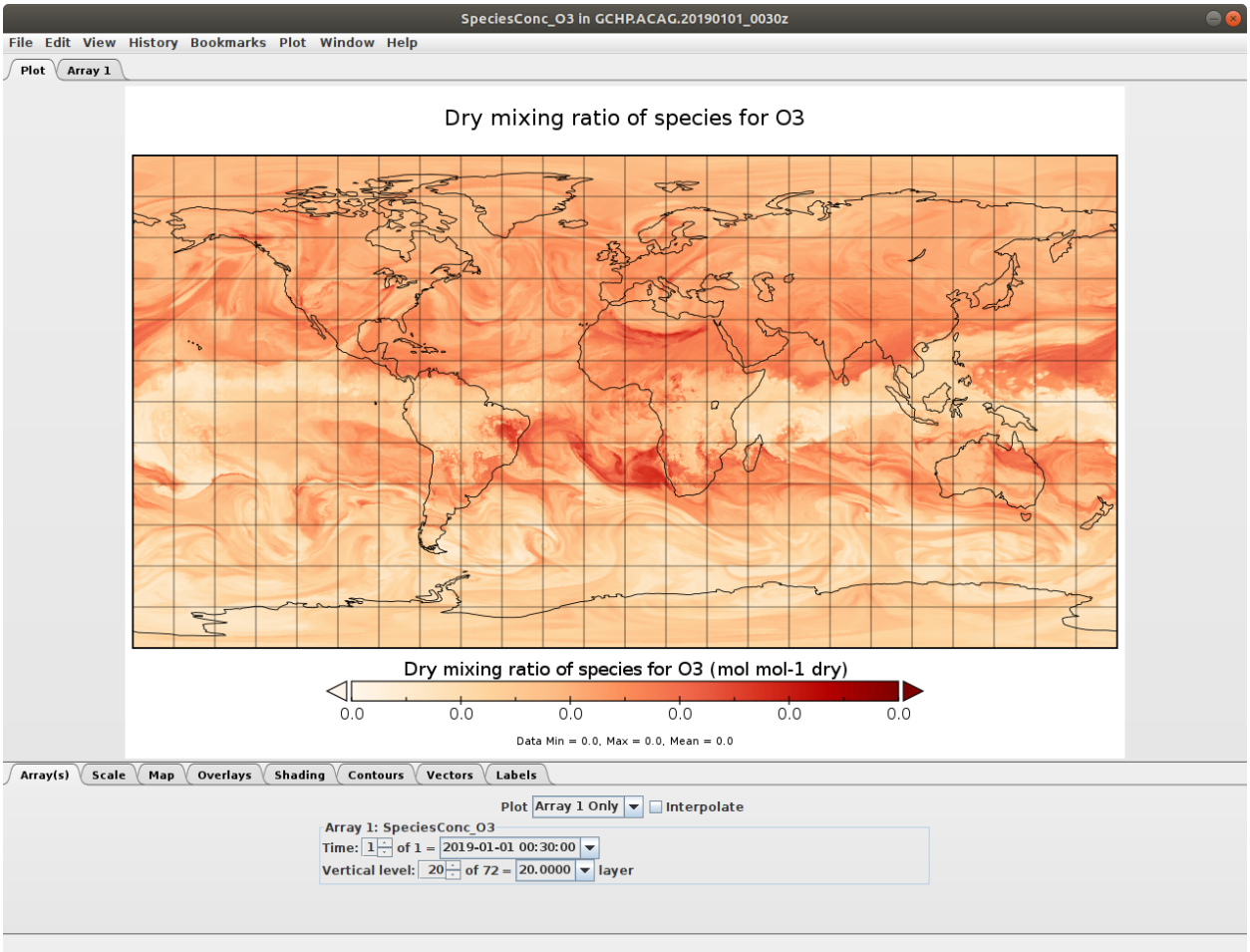
CHAPTER

FOURTEEN

PLOTTING GCHP OUTPUT

14.1 Panoply

Panoply is useful for quick and easy viewing of GCHP output. Panoply is a grahpical program for plotting geo-referenced data like GCHP's output. It is an intuitive program and it is easy to set up.



You can read more about Panoply, including how to install it, here.

Some suggestions

- If you can mount your cluster's filesystem as a Network File System (NFS) on your local machine, you can install Panoply on your local machine and view your GCHP data through the NFS.

- If your cluster supports a graphical interface, you could install Panoply (administrative priviledges not necessary, provided Java is installed) yourself.
- Alternatively, you could install Panoply on your local machine and use **scp** or similar to transfer files back and forth when you want to view them.

Note: To get rid of the missing value bands along face edges, turn interpolation off in the *Array(s)* tab of your plot settings.

14.2 Python

Todo: see the *stretched-grid tutorial's plotting section* in the meantime.

Some notes:

- xarray and cartopy are the fundamental tools
- cartopy > 0.18 fixes the "streaking" of grid-boxes crossing the antimeridian with `pcolormesh()`. As of writing, cartopy 0.19 is not yet released. In the meatime you can install it from GitHub with

```
$ pip install git+https://github.com/SciTools/cartopy.git
```

- The cubed-sphere grid is a curvilinear grid, so you need grid-box corners to plot cubed-sphere data with `pcolormesh()`. See the stretched-grid tutorial for an example.

CHAPTER

FIFTEEN

STRETCHED-GRID SIMULATIONS

Note: Stretched-grid simulations are described in [Bindle et al., 2020]. The paper also discusses things you should consider, and offers guidance for choosing appropriate stretching parameters.

A stretched-grid is a cubed-sphere grid that is "stretched" to enhance its resolution in a region. To set up a stretched-grid simulation you need to do two things:

1. Create a restart file for your simulation.
2. Update `runConfig.sh` to specify the grid and restart file.

Before setting up your stretched-grid simulation, you will need to choose stretching parameters.

15.1 Choose stretching parameters

The *target face* is face of a stretched-grid that shrinks so that the grid resolution is finer. The target face is centered on a target point, and the degree of stretching is controlled by a parameter called the stretch-factor. Relative to a normal cubed-sphere, the resolution of the target face is refined by approximately the stretch-factor. For example, a C60 stretched-grid with a stretch-factor of 3.0 has approximately C180 (~50 km) resolution in the target face. The enhancement-factor is approximate because (1) the stretching gradually changes with distance from the target point, and (2) gnominic cubed-sphere grids are quasi-uniform with grid-boxes at face edges being ~1.5x shorter than at face centers.

You can choose a stretch-factor and target point using the interactive figure below. You can reposition the target face by changing the target longitude and target latitude. The domain of refinement can be increased or decreased by changing the stretch-factor. Choose parameters so that the target face roughly covers the refion that you want to refine.

Note: The interactive figure above can be a bit fiddly. Refresh the page if the view gets messed up. If the figure above is not showing up properly, please *open an issue*.

Next you need to choose a cubed-sphere size. The cubed-sphere size must be an even integer (e.g., C90, C92, C94, etc.). Remeber that the resolution of the target face is enhanced by approximately the stretch-factor.

15.2 Create a restart file

A simulation restart file must have the same grid as the simulation. For example, a C180 simulation requires a restart file with a C180 grid. Likewise, a stretched-grid simulation needs a restart file with the same stretched-grid (i.e., an identical cubed-sphere size, stretch-factor, target longitude, and target latitude).

You can regrid an existing restart file to a stretched-grid with GCPy's **gcpy.file_regrid** program. Below is an example of regridding a C90 cubed-sphere restart file to a C48 stretched-grid with a stretch factor of 3, a target longitude of 260.0, and a target latitude of 40.0. See the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.file_regrid                                \
               -i initial_GEOSChem_rst.c90_standard.nc      \
               --dim_format_in checkpoint                   \
               -o sg_restart_c48_3_260_40.nc                \
               --cs_res_out 48                              \
               --sg_params_out 3.0 260.0 40.0               \
               --dim_format_out checkpoint
```

Description of arguments:

-i `initial_GEOSChem_rst.c90_standard.nc`
: Specifies the input restart file is `initial_GEOSChem_rst.c90_standard.nc` (in the current working directory).

--dim_format_in `checkpoint`
: Specifies that the input file is in the "checkpoint" format. GCHP restart files use the "checkpoint" format.

-o `sg_restart_c48_3_260_40.nc`
: Specifies that the output file should be named `sg_restart_c48_3_260_40.nc`.

--cs_res_out `48`
: Specifies that the output grid has a cubed-sphere size 48 (C48).

--sg_params_out `3.0 260.0 40.0`
: Specifies that the output grid's stretched-grid parameters in the order stretch factor (3.0), target longitude (260.0), target latitude (40.0).

--dim_format_out `checkpoint`
: Specifies that the output file should be in the "checkpoint" format. GCHP restart files must be in the "checkpoint" format.

Once you have created a restart file for your simulation, you can move on to updating your simulation's configuration files.

15.3 Update your configuration files

Modify the section of `runConfig.sh` that controls the simulation grid. Turn `STRETCH_GRID` to `ON` and update `CS_RES`, `STRETCH_FACTOR`, `TARGET_LAT`, and `TARGET_LON` for your specific grid.

```
#------------------------------------------------
#   Internal Cubed Sphere Resolution
#------------------------------------------------

# Primary resolution is an integer value. Set stretched grid to ON or OFF.
```

(continues on next page)

(continued from previous page)

```
#   24 ~ 4x5, 48 ~ 2x2.25, 90 ~ 1x1.25, 180 ~ 1/2 deg, 360 ~ 1/4 deg
CS_RES=24
STRETCH_GRID=ON

# Stretched grid parameters
# Rules and notes:
#    (1) Minimum STRETCH_FACTOR is 1.0001
#    (2) Target lat and lon must be floats (contain decimal)
#    (3) Target lon must be in range [0,360)
STRETCH_FACTOR=3.0
TARGET_LAT=40.0
TARGET_LON=260.0
```

Next, modify the section of `runConfig.sh` that specifies the simulation restart file. Set `INITIAL_RESTART` to the restart file we created in the *previous step*.

```
#-----------------------------------------------
#    Initial Restart File
#-----------------------------------------------
# By default the linked restart files in the run directories will be
# used. Please note that HEMCO restart variables are stored in the same
# restart file as species concentrations. Initial restart files available
# on gcgrid do not contain HEMCO variables which will have the same effect
# as turning the HEMCO restart file option off in GC classic. However, all
# output restart files will contain HEMCO restart variables for your next run.
# INITIAL_RESTART=initial_GEOSChem_rst.c${CS_RES}_TransportTracers.nc

# You can specify a custom initial restart file here to overwrite:
INITIAL_RESTART=sg_restart_c48_3_260_40.nc
```

Lastly, execute **`./runConfig.sh`** to update to update your run directory's configuration files.

```
$ ./runConfig.sh
```

CHAPTER

SIXTEEN

OUTPUT ALONG A TRACK

HISTORY collections can define a `track_file` that specifies a 1D timeseries of coordinates that the model is sampled at. The collection output has the same coordinates as the track file. This feature can be used to sample GCHP along a satellite track or a flight path. A track file is a NetCDF file with the following format

```
$ ncdump -h example_track.nc
netcdf example_track.nc {
dimensions:
   time = 1234 ;
variables:
   float time(time) ;
      time:_FillValue = NaNf ;
      time:long_name = "time" ;
      time:units = "hours since 2020-06-01 00:00:00" ;
   float longitude(time) ;
      longitude:_FillValue = NaNf ;
      longitude:long_name = "longitude" ;
      longitude:units = "degrees_east" ;
   float latitude(time) ;
      latitude:_FillValue = NaNf ;
      latitude:long_name = "latitude" ;
      latitude:units = "degrees_north" ;
}
```

Important: Longitudes must be between 0 and 360.

Important: When using `recycle_track`, the time offsets must be between 0 and 24 hours.

To configure 1D output, you can add the following attributes to any collection in `HISTORY.rc`.

track_file Path to a track file. The associated collection will be sampled from the model along this track. A track file is a 1-dimensional timeseries of latitudes and longitudes that the model is be sampled at (nearest neighbor).

recycle_track Either `.false.` (default) or `.true.`. When enabled, HISTORY replaces the date of the `time` coordinate in the track file with the simulation's current day. This lets you use the same track file for every day of your simulation.

Note: 1D output only works for instantaneous sampling.

The `frequency` attribute is ignored when `track_file` is used.

16.1 Creating a satellite track file

GCPy includes a command line tool, **`gcpy.raveller_1D`**, for generating track files for polar orbiting satellites. These track files will sample model grid-boxes at the times that correspond to the satellite's overpass time. You can also use this tool to "unravel" the resulting 1D output back to a cubed-sphere grid. Below is an example of using **`gcpy.raveller_1D`** to create a track file for a C180 simulation for TROPOMI, which is in ascending sun-synchronous orbit with 14 orbits per day and an overpass time of 13:30. Please see the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.raveller_1D create_track --cs_res 24 --overpass_time 13:30 --direction␣
↪ascending --orbits_per_day 14 -o tropomi_overpass_c24.nc
```

The resulting track file, `tropomi_overpass_c24.nc`, looks like so

```
$ ncdump -h tropomi_overpass_c24.nc
netcdf tropomi_overpass_c24 {
dimensions:
   time = 3456 ;
variables:
   float time(time) ;
      time:_FillValue = NaNf ;
      time:long_name = "time" ;
      time:units = "hours since 1900-01-01 00:00:00" ;
   float longitude(time) ;
      longitude:_FillValue = NaNf ;
      longitude:long_name = "longitude" ;
      longitude:units = "degrees_east" ;
   float latitude(time) ;
      latitude:_FillValue = NaNf ;
      latitude:long_name = "latitude" ;
      latitude:units = "degrees_north" ;
   float nf(time) ;
      nf:_FillValue = NaNf ;
   float Ydim(time) ;
      Ydim:_FillValue = NaNf ;
   float Xdim(time) ;
      Xdim:_FillValue = NaNf ;
}
```

Note: Track files do not require the `nf`, `Ydim`, `Xdim` variables. The are used for post-process "ravelling" with **`gcpy.raveller_1D`** (changing the 1D output's coordinates to a cubed-sphere grid).

Note: With `recycle_track`, HISTORY replaces the reference date (e.g., 1900-01-01) with the simulation's current date, so you can use any reference date.

16.2 Updating HISTORY

Open `HISTORY.rc` and add the `track_file` and `recycle_track` attributes to your desired colleciton. For example, the following is a custom collection that samples NO2 along the `tropomi_overpass_c24.nc`.

```
  TROPOMI_NO2.template:       '%y4%m2%d2_%h2%n2z.nc4',
  TROPOMI_NO2.format:         'CFIO',
  TROPOMI_NO2.duration:       240000
  TROPOMI_NO2.track_file:     tropomi_overpass_c24.nc
  TROPOMI_NO2.recycle_track:  .true.
  TROPOMI_NO2.mode:           'instantaneous'
  TROPOMI_NO2.fields:         'SpeciesConc_NO2            ', 'GCHPchem',
::
```

16.3 Unravelling 1D overpass timeseries

To covert the 1D timeseries back to a cubed-sphere grid, you can use **`gcpy.raveller_1D`**. Below is an example of changing the 1D output back to model grid. Again, see the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.raveller_1D unravel --track tropomi_overpass_c24.nc -i OutputDir/GCHP.
↪TROPOMI_NO2.20180101_1330z.nc4 -o OutputDir/GCHP.TROPOMI_NO2.20180101_1330z.OVERPASS.
↪nc4
```

The resulting dataset, `GCHP.TROPOMI_NO2.20180101_1330z.OVERPASS.nc4`, are simulated concentration on the model grid, sampled at the times that correspond to TROPOMI's overpass.

CHAPTER

SEVENTEEN

STRETCHED-GRID SIMULATION: EASTERN US

This tutorial walks you through setting up and running a stretched-grid simulation for ozone in the eastern US. The grid parameters for this tutorial are

Parameter	Value
Stretch-factor	3.6
Cubed-sphere size	C60
Target latitude	37° N
Target longitude	275° E

These parameters were choosen so that the target face covered the eastern US. Some back-of-the-envelope resolution calculations are

$$\text{average resolution of target face} = \mathrm{R_{tf}} \approx \frac{10000\ \mathrm{km}}{\mathrm{N} \times \mathrm{S}} = 46\ \mathrm{km}$$

and

$$\text{coarsest resolution in target face (at the center)} \approx \mathrm{R_{tf}} \times 1.2 = 56\ \mathrm{km}$$

and

$$\text{finest resolution in target face (at the edges)} \approx \mathrm{R_{tf}} \div 1.2 = 39\ \mathrm{km}$$

and

$$\text{coarsest resolution globally (at target antipode)} \approx \mathrm{R_{tf}} \times \mathrm{S}^2 \times 1.2 = 720\ \mathrm{km}$$

where N is the cubed-sphere size and S is the stretch-factor. The actual value of these, calculated from the grid-box areas, are 46 km, 51 km, 42 km, and 664 km respectively.

Note: This tutorial uses a relatively large stretch-factor. A smaller stretch-factor, like 2.0, would have a refinement that more broad, and the range resolutions would be smaller.

17.1 Tutorial prerequisites

Before continuing with the tutorial:

- You need to be able to run GCHP simulations
- You need to install gcpy >= 1.0.0, and cartopy >= 0.19
- You need emissions data and MERRA2 data for July 2019

Create a new run directory. This run directory should be use full chemistry with standard simulation options, and use MERRA2 meteorology. Make the following modifications to `runConfig.sh`:

- Change the simulation's start time to `"20190701 000000"`
- Change the simulation's end time to `"20190708 000000"`
- Change the simulation's duration to `"00000007 000000"`
- Change `timeAvg_freq` to `"240000"` (daily diagnostics)
- Change `timeAvg_dur` to `"240000"` (daily diagnostics)
- Update the compute resources as you like. This simulation's computational demands are about $1.5\times$ that of a C48 or 2°x2.5° simulation.

Note: I chose to use 30 cores on 1 node, and the simulation took 7 hours to run. For comparison, I also ran the simulation on 180 cores across 6 nodes, and that took about 2 hours.

Update `gchp.local.run` so `nCores` matches your setting in `runConfig.sh`. Now you are ready to continue with the tutorial. The rest of the tutorial assume that your current working directory is your run directory.

17.2 Create your restart file

First, create a restart file for the simulation. GCHP ingests the restart file directly (no online regridding), so the first thing you need to do is regrid a restart file to your stretched-grid. You can regrid `initial_GEOSChem_rst.c48_fullchem.nc` with GCPy like so:

```
$ python -m gcpy.file_regrid                          \
     -i initial_GEOSChem_rst.c48_fullchem.nc          \
     --dim_format_in checkpoint                       \
     --dim_format_out checkpoint                      \
     --cs_res_out 60                                  \
     --sg_params_out 3.6 275 37                       \
     -o initial_GEOSChem_rst.EasternUS_SG_fullchem.nc
```

This creates `initial_GEOSChem_rst.EasternUS_SG_fullchem.nc`, which is the new restart file for your simulation.

Note: This command takes about a minute to run. If you regridding a large restart file (e.g., C180) it may take significantly longer.

17.3 Update runConfig.sh

Make the following updates to runConfig.sh:

- Change INITIAL_RESTART to use initial_GEOSChem_rst.EasternUS_SG_fullchem.nc
- Change CS_RES to 60
- Change STRETCH_GRID to ON
- Change STRETCH_FACTOR to 3.6
- Change TARGET_LAT to 37.0
- Change TARGET_LON to 275.0

Execute runConfig.sh to apply the updates to the various configuration files:

```
$ ./runConfig.sh
```

17.4 Run GCHP

Run GCHP:

```
$ ./gchp.local.run
```

17.5 Plot the output

Append grid-box corners:

```
$ python -m gcpy.append_grid_corners \
     --sg_params 3.6 275 37 \
     OutputDir/GCHP.SpeciesConc.20190707_1200z.nc4
```

Plot ozone at model level 22:

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

# Load 24-hr average concentrations for 2019-07-07
ds = xr.open_dataset('GCHP.SpeciesConc.20190707_1200z.nc4')

# Get Ozone at level 22
ozone_data = ds['SpeciesConc_O3'].isel(time=0, lev=22).squeeze()

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

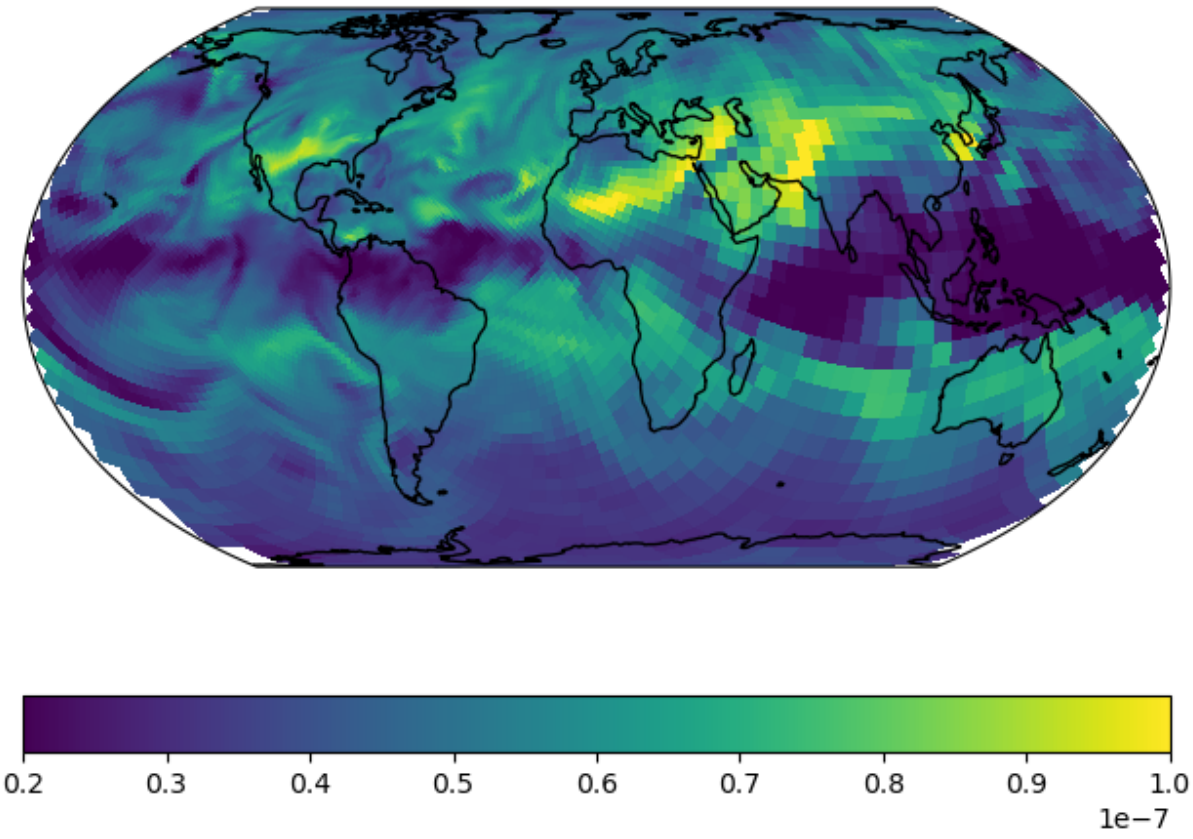
# Plot data on each face
```

(continues on next page)

(continued from previous page)

```
for face_idx in range(6):
    x = ds.corner_lons.isel(nf=face_idx)
    y = ds.corner_lats.isel(nf=face_idx)
    v = ozone_data.isel(nf=face_idx)
    pcm = plt.pcolormesh(
        x, y, v,
        transform=ccrs.PlateCarree(),
        vmin=20e-9, vmax=100e-9
    )

plt.colorbar(pcm, orientation='horizontal')
plt.show()
```



CHAPTER

EIGHTEEN

KNOWN BUGS

This page links to known bugs in GCHP. See the GitHub issues for updates on their status.

18.1 Known bugs in GEOS-Chem

These are known bugs in specific versions of GEOS-Chem. See the associated issue on GitHub for the recommended solution/workaround.

Version 13.0

- Some GEOS-FP and MERRA-2 files between 2020-10 and 2021-05 on https://geoschemdata.computecanada.ca are malformed. See https://github.com/geoschem/geos-chem/issues/748.
- This version of GCHP is not compatible with GCC (GNU) 10 compilers.
- Crash because of incorrect timestamps at start/end of year in OFFLINE_BIOVOC/v2019-10. See https://github.com/geoschem/GCHP/issues/84 for workaround.
- Segmentation fault at runtime with Intel 2021.1 compilers. See https://github.com/geoschem/GCHP/issues/87 for workaround.

Version 12.9

- Crash because of incorrect timestamps at start/end of year in OFFLINE_BIOVOC/v2019-10. See https://github.com/geoschem/GCHP/issues/84 for workaround.

Version 12.8

- Crash because of incorrect timestamps at start/end of year in OFFLINE_BIOVOC/v2019-10. See https://github.com/geoschem/GCHP/issues/84 for workaround.

Version 12.7

- Crash because of incorrect timestamps at start/end of year in OFFLINE_BIOVOC/v2019-10. See https://github.com/geoschem/GCHP/issues/84 for workaround.

18.2 Known bugs in external software affecting GEOS-Chem

These are known bugs in external dependencies that affect GEOS-Chem.

Intel MPI & Compilers

- Intel 2021.1 (or similar) might require CMake 3.19 or newer: https://github.com/geoschem/GCHP/issues/85

CHAPTER

NINETEEN

SUPPORT GUIDELINES

GEOS-Chem support is maintained by the GEOS-Chem Support Team (GCST). The GCST members are based at Harvard University and Washington University in St. Louis.

We track bugs, user questions, and feature requests through GitHub issues. Please help out as you can in response to issues and user questions.

19.1 How to report a bug

We use GitHub to track issues. To report a bug, open a new issue. Please include all the information that might be relevant, including instructions for reproducing the bug.

19.2 Where can I ask for help?

We use GitHub issues to support user questions. To ask a question, open a new issue and select the question template.

19.3 How to submit changes

Please see "Contributing Guidelines".

19.4 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER

TWENTY

CONTRIBUTING GUIDELINES

Thank you for looking into contributing to GEOS-Chem! GEOS-Chem is a grass-roots model that relies on contributions from community members like you. Whether you're new to GEOS-Chem or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

20.1 We use GitHub and ReadTheDocs

We use GitHub to host the GCHP source code, to track issues, user questions, and feature requests, and to accept pull requests: https://github.com/geoschem/GCHP. Please help out as you can in response to issues and user questions.

We use ReadTheDocs to host the GCHP user documentation: https://gchp.readthedocs.io.

20.2 How to submit changes

We use GitHub Flow, so all changes happen through pull requests. This workflow is described here: GitHub Flow. If your change affects multiple submodules, submit a pull request for each submodule with changes, and link to these submodule pull requests in your main pull request.

As the author you are responsible for:

- Testing your changes
- Updating the user documentation (if applicable)
- Supporting issues and questions related to your changes in the near-term

20.3 Coding conventions

The GEOS-Chem codebase dates back several decades and includes contributions from many people and multiple organizations. Therefore, some inconsistent conventions are inevitable, but we ask that you do your best to be consistent with nearby code.

20.4 How to request an enhancement

We accept feature requests through issues on GitHub. To request a new feature, open a new issue and select the feature request template. Please include all the information that migth be relevant, including the motivation for the feature.

20.5 How to report a bug

Please see "Support Guidelines".

20.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

TWENTYONE

EDITING THIS USER GUIDE

This user guide is generated with Sphinx. Sphinx is an open-source Python project designed to make writing software documentation easier. The documentation is written in a reStructuredText (it's similar to markdown), which Sphinx extends for software documentation. The source for the documentation is the `docs/source` directory in top-level of the source code.

21.1 Quick start

To build this user guide on your local machine, you need to install Sphinx. Sphinx is a Python 3 package and it is available via **`pip`**. This user guide uses the Read The Docs theme, so you will also need to install `sphinx-rtd-theme`. It also uses the sphinxcontrib-bibtex and recommonmark extensions, which you'll need to install.

```
$ pip install sphinx sphinx-rtd-theme sphinxcontrib-bibtex recommonmark
```

To build this user guide locally, navigate to the `docs/` directory and make the `html` target.

```
gcuser:~$ cd gcpy/docs
gcuser:~/gcpy/docs$ make html
```

This will build the user guide in `docs/build/html`, and you can open `index.html` in your web-browser. The source files for the user guide are found in `docs/source`.

Note: You can clean the documentation with `make clean`.

21.2 Learning reST

Writing reST can be tricky at first. Whitespace matters, and some directives can be easily miswritten. Two important things you should know right away are:

- Indents are 3-spaces
- "Things" are separated by 1 blank line. For example, a list or code-block following a paragraph should be separated from the paragraph by 1 blank line.

You should keep these in mind when you're first getting started. Dedicating an hour to learning reST will save you time in the long-run. Below are some good resources for learning reST.

- reStructuredText primer: (single best resource; however, it's better read than skimmed)
- Official reStructuredText reference (there is *a lot* of information here)

- Presentation by Eric Holscher (co-founder of Read The Docs) at DjangoCon US 2015 (the entire presentation is good, but reST is described from 9:03 to 21:04)
- YouTube tutorial by Audrey Tavares's

A good starting point would be Eric Holscher's presentations followed by the reStructuredText primer.

21.3 Style guidelines

Important: This user guide is written in semantic markup. This is important so that the user guide remains maintainable. Before contributing to this documentation, please review our style guidelines (below). When editing the source, please refrain from using elements with the wrong semantic meaning for aesthetic reasons. Aesthetic issues can be addressed by changes to the theme.

For **titles and headers**:

- Section headers should be underlined by # characters
- Subsection headers should be underlined by - characters
- Subsubsection headers should be underlined by ^ characters
- Subsubsubsection headers should be avoided, but if necessary, they should be underlined by " characters

File paths (including directories) occuring in the text should use the `:file:` role.

Program names (e.g. **`cmake`**) occuring in the text should use the `:program:` role.

OS-level commands (e.g. **`rm`**) occuring in the text should use the `:command:` role.

Environment variables occuring in the text should use the `:envvar:` role.

Inline code or code variables occuring in the text should use the `:code:` role.

Code snippets should use `.. code-block::` <language> directive like so

```
.. code-block:: python

   import gcpy
   print("hello world")
```

The language can be "none" to omit syntax highlighting.

For command line instructions, the "console" language should be used. The $ should be used to denote the console's prompt. If the current working directory is relevant to the instructions, a prompt like `gcuser:~/path1/path2$` should be used.

Inline literals (e.g. the $ above) should use the `:literal:` role.

CHAPTER

TWENTYTWO

GIT SUBMODULES

22.1 Forking submodules

This sections describes updating git submodules to use your own forks. You can update submodule so that they use your forks at any time. It is recommended you only update the submodules that you need to, and that you leave submodules that you don't need to modify pointing to the GEOS-Chem repositories.

The rest of this section assumes you are in the top-level of GCHPctm, i.e.,

```
$ cd GCHPctm   # navigate to top-level of GCHPctm
```

First, identify the submodules that you need to modify. The `.gitmodules` file has the paths and URLs to the submodules. You can see it with the following command

```
$ cat .gitmodules
[submodule "src/MAPL"]
   path = src/MAPL
   url = https://github.com/sdeastham/MAPL
[submodule "src/GMAO_Shared"]
   path = src/GMAO_Shared
   url = https://github.com/geoschem/GMAO_Shared
[submodule "ESMA_cmake"]
   path = ESMA_cmake
   url = https://github.com/geoschem/ESMA_cmake
[submodule "src/gFTL-shared"]
   path = src/gFTL-shared
   url = https://github.com/geoschem/gFTL-shared.git
[submodule "src/FMS"]
   path = src/FMS
   url = https://github.com/geoschem/FMS.git
[submodule "src/GCHP_GridComp/FVdycoreCubed_GridComp"]
   path = src/GCHP_GridComp/FVdycoreCubed_GridComp
   url = https://github.com/sdeastham/FVdycoreCubed_GridComp.git
[submodule "src/GCHP_GridComp/GEOSChem_GridComp/geos-chem"]
   path = src/GCHP_GridComp/GEOSChem_GridComp/geos-chem
   url = https://github.com/sdeastham/geos-chem.git
[submodule "src/GCHP_GridComp/HEMCO_GridComp/HEMCO"]
   path = src/GCHP_GridComp/HEMCO_GridComp/HEMCO
   url = https://github.com/geoschem/HEMCO.git
```

Once you know which submodules you need to update, fork each of them on GitHub.

Once you have your own forks for the submodules that you are going to modify, update the submodule URLs in `.gitmodules`

```
$ git config -f .gitmodules -e    # opens editor, update URLs for your forks
```

Synchronize your submodules

```
$ git submodule sync
```

Add and commit the update to `.gitmodules`.

```
$ git add .gitmodules
$ git commit -m "Updated submodules to use my own forks"
```

Now, when you push to your GCHPctm fork, you should see the submodules point to your submodule forks.

CHAPTER

TWENTYTHREE

TERMINOLOGY

absolute path The full path to a file, e.g., `/example/foo/bar.txt`. An absolute path should always start with `/`. As opposed to a *relative path*.

build See *compile*.

build directory A directory where build configuration settings are stored, and where intermediate build files like object files, module files, and libraries are stored.

checkpoint file See *restart file*.

compile Generating an executable program from source code (which is in a plain-text format).

gridded component A formal model component. MAPL organizes model components with a tree structure, and facilitates component interconnections.

HISTORY The MAPL *gridded component* that handles model output. All GCHP output diagnostics are facilitated by HISTORY.

relative path The path to a file relative to the current working directory. For example, the relative path to `/example/foo/bar.txt` if your current working directory is `/example` is `foo/bar.txt`. As opposed to an *absolute path*.

restart file A NetCDF file with initial conditions for a simulation. Also called a *checkpoint file* in GCHP.

run directory The working directory for a GEOS-Chem simulation. A run directory houses the simulation's configuration files, the output directory (`OutputDir`), and input files/links such as *restart files* or input data directories.

stretched-grid A cubed-sphere grid that is "stretched" to enhance the grid resolution in a region.

target face The face of a stretched-grid that is refined. The target face is centered on the target point.

CHAPTER

TWENTYFOUR

VERSIONING

todo

CHAPTER

TWENTYFIVE

UPLOADING TO SPACK

This page describes how to upload recipe changes to Spack. Common recipe changes include updating available versions of GCHP and changing version requirements for dependencies.

1. Create a fork of https://github.com/spack/spack.git and clone your fork.
2. Change your `SPACK_ROOT` environment variable to point to the root directory of your fork clone.
3. Create a descriptive branch name in the clone of your fork and checkout that branch.
4. Make any changes to `$SPACK_ROOT/var/spack/repos/builtin/packages/package_name/` as desired.
5. Install Flake8 and mypy using `conda install flake8` and `conda install mypy` if you don't already have these packages.
6. Run Spack's style tests using `spack style`, which will conduct tests in `$SPACK_ROOT` using Flake8 and mypy.
7. (Optional) Run Spack's unit tests using `spack unit-test`. These tests may take a long time to run. The unit tests will always be run when you submit your PR, and the unit tests primarily test core Spack features unrelated to specific packages, so you don't usually need to run these manually.
8. Prefix your commit messages with the package name, e.g. `gchp:  added version 13.1.0`.
9. Push your commits to your fork.
10. Create a PR targetted to the `develop` branch of the original Spack repository, prefixing the PR title with the package name, e.g. `gchp:  added version 13.1.0`.

BIBLIOGRAPHY

[Bey et al., 2001] Bey, I., Jacob, D. J., Yantosca, R. M., Logan, J. A., Field, B. D., Fiore, A. M., Li, Q., Liu, H. Y., Mickley, L. J., and Schultz, M. G. Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation. *Journal of Geophysical Research: Atmospheres*, 106(D19):23073–23095, October 2001. doi:10.1029/2001JD000807.

[Keller et al., 2014] Keller, C. A., Long, M. S., Yantosca, R. M., Da Silva, A. M., Pawson, S., and Jacob, D. J. HEMCO v1.0: a versatile, ESMF-compliant component for calculating emissions in atmospheric models. *Geoscientific Model Development*, 7(4):1409–1417, July 2014. doi:10.5194/gmd-7-1409-2014.

[Long et al., 2015] Long, M. S., Yantosca, R., Nielsen, J. E., Keller, C. A., da Silva, A., Sulprizio, M. P., Pawson, S., and Jacob, D. J. Development of a grid-independent GEOS-Chem chemical transport model (v9-02) as an atmospheric chemistry module for Earth system models. *Geoscientific Model Development*, 8(3):595–602, March 2015. doi:10.5194/gmd-8-595-2015.

[Eastham et al., 2018] Eastham, S. D., Long, M. S., Keller, C. A., Lundgren, E., Yantosca, R. M., Zhuang, J., Li, C., Lee, C. J., Yannetti, M., Auer, B. M., Clune, T. L., Kouatchou, J., Putman, W. M., Thompson, M. A., Trayanov, A. L., Molod, A. M., Martin, R. V., and Jacob, D. J. GEOS-Chem High Performance (GCHP v11-02c): a next-generation implementation of the GEOS-Chem chemical transport model for massively parallel applications. *Geoscientific Model Development*, 11(7):2941–2953, July 2018. doi:10.5194/gmd-11-2941-2018.

[Zhuang et al., 2020] Zhuang, J., Jacob, D. J., Lin, H., Lundgren, E. W., Yantosca, R. M., Gaya, J. F., Sulprizio, M. P., and Eastham, S. D. Enabling High-Performance Cloud Computing for Earth Science Modeling on Over a Thousand Cores: Application to the GEOS-Chem Atmospheric Chemistry Model. *Journal of Advances in Modeling Earth Systems*, May 2020. doi:10.1029/2020MS002064.

[Bindle et al., 2020] Bindle, L., Martin, R. V., Cooper, M. J., Lundgren, E. W., Eastham, S. D., Auer, B. M., Clune, T. L., Weng, H., Lin, J., Murray, L. T., Meng, J., Keller, C. A., Pawson, S., and Jacob, D. J. Grid-Stretching Capability for the GEOS-Chem 13.0.0 Atmospheric Chemistry Model. *Geoscientific Model Development*, December 2020. doi:10.5194/gmd-2020-398.

INDEX

Symbols

--cs_res_out 48
    command line option, 70
--dim_format_in checkpoint
    command line option, 70
--dim_format_out checkpoint
    command line option, 70
--sg_params_out 3.0 260.0 40.0
    command line option, 70
-i initial_GEOSChem_rst.c90_standard.nc
    command line option, 70
-o sg_restart_c48_3_260_40.nc
    command line option, 70

A

absolute path, **91**

B

build, **91**
build directory, **91**

C

CC, 15
checkpoint file, **91**
command line option
    --cs_res_out 48, 70
    --dim_format_in checkpoint, 70
    --dim_format_out checkpoint, 70
    --sg_params_out 3.0 260.0 40.0, 70
    -i initial_GEOSChem_rst.c90_standard.nc, 70
    -o sg_restart_c48_3_260_40.nc, 70
compile, **91**
CS_RES, 70
CXX, 15

E

environment variable
    CC, 15
    CS_RES, 70
    CXX, 15
    ESMF_ROOT, 60
    FC, 15
    GC_DATA_ROOT, 19
    INITIAL_RESTART, 71
    STRETCH_FACTOR, 70
    STRETCH_GRID, 70
    TARGET_LAT, 70
    TARGET_LON, 70
ESMF_ROOT, 60

F

FC, 15

G

GC_DATA_ROOT, 19
gridded component, **91**

H

HISTORY, **91**

I

INITIAL_RESTART, 71

R

relative path, **91**
restart file, **91**
run directory, **91**

S

STRETCH_FACTOR, 70
STRETCH_GRID, 70
stretched-grid, **91**

T

target face, **91**
TARGET_LAT, 70
TARGET_LON, 70