Global

System for Atmospheric Modeling

gSAM

Version 1.6.2

User Guide

Marat Khairoutdinov

Stony Brook University

marat.khairoutdinov@stonybrook.edu

Updated: April, 2024

The gSAM project has been made possible through the support provided by the

National Science Foundation

Table of Contents

1 Introduction

The Global System for Atmospheric Modeling (gSAM, Khairoutdinov et al., 2022a) is a versatile non-hydrostatic anelastic model designed for simulating cloudy atmospheres across various scales. Its capabilities encompass the modeling of phenomena from small-scale turbulence and individual cloud formations to convective systems, hurricanes, and even the global general circulation. Depending on your needs, gSAM can be configured in several ways:

1. As a **Direct Numerical Simulation (DNS)** model, offering a resolution of a few millimeters.
2. As a **Large-Eddy Simulation (LES)** model, with resolution ranging from meters to tens of meters.
3. As a **Cloud-Resolving Model (CRM)**, which provides resolutions between a hundred meters up to several kilometers.
4. As a **Global Storm-Resolving Model (GSRM)** that captures details at a resolution of a few kilometers.

Details of the model's formulation and the results of various global model tests are given by Khairoutdinov et al. (2022a).

While SAM (Khairoutdinov and Randall, 2003) served as a reliable LES and CRM tool for a wide research community, gSAM marks a leap forward, particularly with its ability to model large-scale cloud organizations more accurately. One of the pivotal enhancements is the transition from a Cartesian coordinate system to a latitude-longitude-based grid. This shift not only minimizes code changes but also makes it convenient to revert to Cartesian coordinates when needed.

The model comes integrated with a newly developed, highly efficient hybrid FFT-multigrid pressure solver, revolutionizing the way anelastic flow is handled. Moreover, gSAM introduces an innovative approach to simulating topography, employing the body-force or mask method to produce stagnated flow around structures—a feature absent in SAM. This capability opens the doors for advanced simulations in diverse applications, including urban planning and architectural design.

gSAM also introduces a plethora of other advanced features, such as:

- The ability to initialize and set boundary conditions using 2D and 3D input datasets.
- Options to run weather-nudged global and regional simulations.
- A considerably updated land surface model, featuring interactive snow cover.
- Numerous new namelist variables for expanded functionality.

This User Guide aims to offer comprehensive instructions on compiling, configuring, and running gSAM, along with managing its output. Whether you're new to atmospheric modeling or an expert accustomed to the standard SAM, this guide is designed to make your transition to gSAM as smooth as possible.

2 Files and Directories

When you navigate to the gSAM root-directory, you'll find a file-structure similar to the following:

```
…gSAM1.6> ls

Build*    CASES/       CaseName      DOC/    Changes_Log/    GLOBAL_DATA/   GRIDS/     Makefile
SCRIPTS/    SRC/    UTIL/
```

SRC/: This directory contains all the source files of the model.

CASES/: Here, you'll find individual case-directories. Each directory corresponds to a specific simulation case, containing all the necessary files for that simulation. As a user, you can create new case-directories with ease.

CaseName: This file determines the name of the current case-directory that's in use when model is run.

Build: This is a C-shell script responsible for constructing the gSAM executable. Not only does it generate the main executable but also sets up additional output directories and symbolic links (more on that below). The script manages environmental variables, creates a file-dependency tree, and then utilizes an appropriate version of the make-utility for code compilation. **It's important to note: you shouldn't use the make utility directly to compile gSAM,** even though there's a Makefile present.

UTIL/: Within this directory, you'll find source files essential for building utilities. These utilities primarily convert model output files from a customized internal format into netcdf, among other functions. **Contrary to gSAM's main build process, you should employ the make utility** to compile the conversion utilities located in the UTIL/SRC sub-directory. Notably, the UTIL directory maintains its distinct Makefile, which you should edit for your system and library paths.

SCRIPTS/: This directory is a repository of beneficial C-shell scripts as well as NCL (NCAR Command Language) scripts for many of the cases in the CASES/ directory that can be used for analysis and visualization of results. These have been graciously provided by the author, but it's crucial to understand that these scripts currently do not come with specific documentation. The exception is the detailed description of the process of simultaneous or parallel conversion of multiple output files using the scripts in SCRIPTS/FILES/ directory.

GRIDS/: This directory contains a utility for generation of a global grid in latitudinal direction as well as it contains many pre-made global grid files for various resolutions.

DOC/: Directory to contain model documentation.

Changes_Log/: This will contain notes on major model changes from one version to the next.

GLOBAL_DATA/: directory that contain scripts to generate initial and boundary conditions as well as symbolic links to various datasets.

3 Building the executable

Those familiar with building SAM will find the process for gSAM quite similar. While the core of the procedure remains unchanged, there are noteworthy additions and new variables to configure. Thus, even if you're well-versed with SAM, it's essential to continue reading and familiarize yourself with these updates.

To compile the gSAM executable, follow 4 steps outlined below.

3.1 Step 1: Modifying the Build script.

If you're compiling gSAM for the first time on a given machine, you'll need to tweak the **Build** script to set several environmental variables crucial for the compilation.

3.1.1 Defining the SAM_SCR Variable:

The gSAM executable always runs from the gSAM root-directory, typically found in a user's home directory. This placement ensures that root directory is protected from unintentional deletions by scratch-disk cleaning tools. However, due to limited space in user home directories, especially on supercomputer systems, users often use a larger 'scratch' disk space to store the output.

The **SAM_SCR** variable directs where to save model's output files and compilation files. For instance, if you want to set a directory with the same name as the gSAM root-directory (recommended) in the scratch-disk directory /scratch/username, use the following line in Build replacing /scratch/username/ with actual path:

```
setenv SAM_SCR /scratch/username/gSAM1.6
```

Once you run the **Build** script, it will create the directory defined by **SAM_SCR** (if it doesn't exist already) and several subdirectories: OBJ, OUT_STAT, OUT_2D, OUT_3D, OUT_2DL, OUT_MISC, OUT_INV, OUT_MOMENTS, OUT_MOVIES, RESTART and RESTART1. If the **SAM_SCR** variable points to a directory other than the SAM root-directory, symbolic links to these subdirectories will be created in the root directory. The use of symbolic links makes use of output directories as if they are present in the root directory itself. Remember that deleting the symbolic links does not mean that the actual directories they point to are deleted as well.

3.1.2 Output Directories

The model writes its output files to various directories that start with prefix OUT. See the section of this document dealing with the output for more details. Also, there are directories used to store the restart files – RESTART and RESTART1 – where files crucial for restarting/continuing a previous run or branching off an earlier run are stored. Finally, there is a directory for compilation – OBJ. It is a designated space for all object files and dependency files created during model compilation.

3.1.3 Model Variables in Build script

The **Build** script contains several variables that allow users to configure the model:

ADV_DIR: This variable points to the directory within SRC/ that contains the advection code for all scalars. Currently, there are two available options: the original SAM's advection scheme (MPDATA; Smolarkiewicz 2006) and the flux-corrected transport scheme (UM5; Yamaguch et al. 2011). Both schemes have roughly similar computational costs. It's important to note that only the MPDATA scheme is compatible with the lat-lon grid, making it the sole option for the global model.

RAD_DIR: This variable indicates the directory containing the desired radiation package. At present, the choices are CAM radiation and RRTMG. They originate from the NCAR Community Atmosphere Model (RAD_CAM) and Atmospheric and Environmental Research, Inc. (RAD_RRTM).

MICRO_DIR: This points to the directory with the cloud microphysics package. The available choices include:

- **MICRO_SAM1MOM:** SAM's original single-moment microphysics.
- **MICRO_M2005:** Morrison's double-moment microphysics from the WRF model.
- **MICRO_THOM:** Thompson's microphysics, also from the WRF model.
- **MICRO_P3:** P3 scheme from the WRF model.
- **MICRO_DRIZZLE:** A basic drizzle microphysics approach, loosely based on the Khairoutdinov and Kogan 2000 scheme.
- **MICRO_KH3:** This is an enhanced version of the SAM1MOM scheme to better simulate ice. It's currently in development and should be used with care.

To incorporate new microphysics packages, users can utilize the MICRO_TEMPLATE directory as a blueprint.

SGS_DIR: This variable defines the subgrid-scale (unresolved) turbulence model utilized by gSAM. Currently, the only option is **SGS_TKE**, which is grounded in the Smagorinski and 1.5-order prognostic SGS turbulent kinetic energy (TKE) closure. Users have the flexibility to develop or adapt other existing SGS packages using the guidelines in the SGS_TEMPLATE directory.

BUFFERED_OUT: This environment variable enables buffered output on compatible systems. Buffering temporarily stores data before transmission, streamlining the writing process, especially for voluminous files. This is achieved by minimizing disk writes or other slow storage media interactions. However, not all systems/compilers support output buffering. Typically, Intel's ifort compiler on Intel processor-based computers does. If you're using another compiler, this feature might not be accessible. To deactivate buffered output, simply comment out the corresponding line in the **Build** script.

PNETCDF_OUT: This environmental variable pertains to the pNetCDF (Parallel NetCDF) library, an extension of the NetCDF library designed for high-performance parallel I/O. To utilize this, pNetCDF must be installed on your system. The benefit of this option is that gSAM produces NetCDF output, so post-processing of native SAM binary output using the routines in the UTIL/ directory is not required. The potential drawbacks is slowdowns of simulations when writing output, especially for very large files and when a large number of compute cores are used. Therefore, if pNetCDF isn't installed on your system or if you see the output routines consuming a lot of run time in the timing outputs, simply comment out the relevant line in the Build script. However, you may choose this option for convenience, especially for relatively small domains, as you would avoid post-processing output files into NetCDF files.

3.1.4 Best Practices

For organization, always use unique output directories for different gSAM home directories, even when using the same version of gSAM. For example, if you are simulating a dry boundary layer case PBL, first copy the gSAM1.6 directory to a new directory preserving all the time stamps using the following command:

```
…> cp -pr  gSAM1.6   gSAM1.6_PBL
```

Then cd to a new root directory and remove all the symbolic links associated with the old root directory using command:

```
…> rm OUT_*  RESTART*  OBJ
```

Be careful not to write a separate *, that is with space on both sides; you may not like the result. Also be careful to remove your output files by using rm OUT_*/*. Don’t worry that you may remove actual directories containing your output data when removing symbolic links. In any case, rm command itself, without special flags, cannot remove directories.

Next, reset the gSAM1.6_PBL/Build script’s SAM_SCR variable to a new value:

```
setenv SAM_SCR /scratch/username/gSAM1.6_PBL
```

That way you won’t mix your cases and will keep your output data well organized and separate.

On personal computers or Linux-clusters, with ample disk space, which is yours only, you might choose to keep the root directory on the main disk and save model output there. In such cases, direct the SAM_SCR variable to the SAM root directory. You can use a direct path, like ./. For clarity and safety, use the line below to generate a comprehensive path (remembering the distinction between Unix command-execution back-quote ` and string quote ‘):

```
setenv SAM_SCR `pwd`
```

3.2 Step 2: Customizing the Makefile for Compilation

The Makefile, a crucial tool for automating the build process, comes pre-configured for the model to be compatible with multiple platforms. However, it's essential to understand how to tailor it to your specific needs.

3.2.1 Pre-existing Platforms

Platform Check: Even if your platform is listed, make sure you inspect the relevant subsection of the Makefile. Ensure that the compiler names, options, and paths to libraries match your system's configuration.

3.2.2 Adding a New Platform

Platform Absence: If the Makefile doesn't list your platform, you'll need to introduce a new subsection.

Use Templates: Leverage the pre-existing platform subsections as a blueprint for creating your own.

Platform Name Retrieval: For UNIX-like systems, run the **uname** command in your terminal. This helps you know the name to be used in the ifeq ($(PLATFORM),...) statement.

3.2.3 Compilation Flags

All compiler calls should contain a flag to invoke a C-preprocessor (e.g., -fpp in ifort).

To compile, you need to set the following Makefile symbolic variables or flags:

FF77: This variable specifies the compiler for Fortran 77 files (with .f extension). A few of these legacy files might still be in use.

Example: mpif90 -c -fpp -fixed -extend_source

FF90: Designates the compiler for Fortran 90 files (.f90 extension).

Example: mpif90 -c -fpp -free

FF95: This variable is for files with .F extension. They're essentially Fortran 90 but with a specific feature to automatically promote all real numbers to real(8) during compilation.

Example: FF95 = mpif90 -c -fpp **-r8** -free (note the flag -r8 that promotes all variable of type REAL to REAL(8))

FFLAGS: Contains additional compiler flags, like those for optimization or memory models.
Example: -O3 -g -traceback -pad -assume buffered_io -mcmodel=large

Compiler Optimization: Always be cautious with high optimization levels, as compilers might sometimes insert flawed code. If your model crashes unexpectedly or gives odd results while it was running "just fine" before, it could be that a new version of compiler was installed on your computer system with new optimizations that may be buggy. Hence, consider first reducing the optimization level as a preliminary troubleshooting step.

3.2.4 Linking Flags

LD: Specifies the compiler utilized for linking.

Example: LD = mpif90 -mcmodel=large

LDFLAGS: These are the flags used to link libraries.

Example: LDFLAGS = -L${LIB_NETCDF} -L{PNETCDF}/lib -lnetcdff -lpnetcdf -mkl=sequential

3.2.5 Flexibility and Dependencies

The Makefile in gSAM employs a dynamic approach to building that avoids the traditional method of hardcoding source file names and dependencies for the *make* utilities. While some developers still prefer the hardcoded approach—suitable for smaller codes—it becomes impractical and overly cumbersome for larger codes that have a multitude of interdependent files. Taking a cue from the CCSM software, gSAM uses Perl scripts to autonomously generate file dependencies that the *make* utilities can use to build the executable.

Perl Scripting Requirement: To enable this dynamic building process, your system must support Perl scripting. Perl is a staple in nearly all Linux/UNIX-based systems. More often than not, the Perl executable (labeled as "perl") is situated in /usr/bin. If your system's configuration diverges from this, modify the paths at the start of the two Perl scripts housed in the SRC/SCRIPTS directory.

Automatic Inclusion: One of the standout features of this method is its ability to automatically recognize, compile, and link any new source file added to the gSAM code. This is as long as the file bears the .f, .f90, or .F extension (refer to compiler flags for their specific purposes). The dependency tree, sculpted by the Perl scripts, orchestrates the sequential compilation of files. If there's a file you'd like to sidestep—perhaps because it's outdated—simply change its extension so it doesn't end with .f, .f90, or .F. Moreover, if you add or remove any file from the source directories, ensure you delete all files in the OBJ directory prior to recompiling to foster the creation of a new dependency tree. And if you've made modifications to any source file, only that particular file and those dependent on it (via module usage) will undergo recompilation automatically.

If you added a new directory to SRC and put your files in there, you need to make sure you added that directory to the directory search-tree in Build script.

Finally, note that gSAM uses Fortran only. It does not incorporate any C files, rendering a C-compiler unnecessary.

3.3 Step 3: Configuring the Computational Grid

Before initiating the compilation process, one must define the computational grid dimensions. These dimensions are outlined in the **SRC/domain.f90** file. The gSAM executable operates strictly on a predefined grid size (and, hence, large array sizes) set during its compilation. Such a design strategy sidesteps dynamic array allocation (although it is still used in some parts of the code), especially in performance-critical sections. Knowing array sizes at compile time often enables the compiler to optimize the code more aggressively. Also try to make the stack memory on your computer as large as possible as automatic arrays tend to be created on stack memory which can be considerably faster than heap memory. For example, for csh/tcsh the command to make maximum possible stack memory could be

```
..> limit stacksize unlimited
```

In some instances, the stack memory can be still insufficient which may crash the model with little explanation why. If you suspect that, use your compiler's option to use heap memory for stack space. For example, in ifort there is the option **-heap-arrays=n** which forces compiler to generate the heap-based allocation of automatic arrays bigger than **n** kilobytes.

3.3.1 Configuring Domain Sizes

Domain Dimensions: The overall grid dimensions in the horizontal x, y, and vertical z directions are defined by the variables **nx_gl**, **ny_gl**, and **nz_gl**, respectively.

Horizontal Domain Constraints: The horizontal dimensions, **nx_gl** and **ny_gl**, are constrained due to certain Fast-Fourier-Transform routines. These dimensions should only be products of 2, 3, and 5. For instance, a value of **nx_gl**=240 is acceptable as $240=2^4\times3\times5$, while **nx_gl**=168 isn't, because $168=2^3\times3\times7$. The model's performance can vary based on the chosen domain dimensions due to factors like cache memory size. Thus, it's advisable to assess model efficiency using various dimensions within a close range. Typically, dimensions that are powers of two (e.g., 64, 128, 512) are optimal. However, other dimensions utilizing the factors 3 and 5 may also be as efficient. It all depends.

Note: The end section of the SRC/domain.f90 file lists all numbers below 10,000 that adhere to the above constraints.

3D vs. 2D: In SAM, the variable **YES3D** determines the model dimensionality. A value of 0 makes SAM run as a 2D model, whereas a value of 1 configures it as a 3D model.

Single-Column Model: gSAM has introduced a **COL1** flag for its single-column model (SCM) version. It's managed similarly to **YES3D**. For a standard run, it will automatically be set to 1. However, for the SCM, when both **nx_gl** and **ny_gl** are set to 1, also set **COL1** to zero.

3.3.2 Parallelization with MPI

gSAM employs the Message-Passing Interface (MPI) protocol for parallel computing. Here's how it works.

Parallel Domains: For parallel execution, the computational domain is divided into uniformly sized subdomains, with each one allocated to a distinct processor core (MPI task). The variables **nsubdomains_x** and **nsubdomains_y**, both located in **SRC/domain.f90**, define the number of subdomains in the x and y directions, respectively. Consequently, the total number of MPI processes amounts to the product of **nsubdomains_x** and **nsubdomains_y**. It's advisable to structure the subdomains, each with nx_gl/nsubdomains_x by ny_gl/nsubdomains_y grid points in the horizontal, to be as close to a square as feasible. This minimizes the boundary perimeter and subsequently reduces the size of MPI messages exchanged among subdomains. In situations where a more rectangular subdomain shape is inevitable, it's generally preferable to have the x dimension of subdomains longer than the y dimension.

3.3.3 Rules for MPI Configuration

The global domain dimensions **nx_gl** and **ny_gl** should be divisible without remainder by **nsubdomains_x** and **nsubdomains_y**, respectively. The rules for choosing the number of subdomains and, hence, the total number of MPI processes or compute cores are fairly simple:

- The maximum number of cores should not exceed **nx_gl**.
- **nx_gl** should be divisible by **ny_gl**.
- **nz_gl** should be divisible by **m = nx_gl/ny_gl**.

If either of the above conditions is not met or if domain is 2D, then the number of pressure levels **nz_gl** should be divisible by the total number of processors. This last case is usually used for 2D runs as they require less MPI tasks. For simulations in rectangular domains with high aspect ratios (e.g., nx_gl >> ny_gl), the choices of grid points and subdomains have some additional constraints. We provide a google sheet that allows you to design grids and domain decompositions that satisfy these constraints here.

Generally, the number of grid points in a horizontal direction is larger than the number of the vertical levels. Therefore, the maximum number of processors that can be used for a grid of given size depends mostly on horizontal dimensions. For example, for 512 x 512 x 64 grid in x, y, and z, the maximum number of processors that can be used is 512 (e.g., **nsubdomains_x**=16, **nsubdomains_y**=32), while for 256 x 512 x 64 grid it is 256. For the 2-D version of SAM (**YES3D** = will be set to 0 in that case), the number of levels should always be divisible by the total number of processors; for example, for 64 levels, you can use, 4, 8, 32, or 64 processors, but not 20.

Serial Execution: If both **nsubdomains_x** and **nsubdomains_y** are set to 1, the code runs on a single processor, foregoing parallel processing. Even in this case, MPI libraries are linked since the model doesn't omit MPI-related code. If MPI libraries aren't available for a single-processor setup, you can replace **task_util_MPI.f90** with **task_util_NOMPI.f90**. For that, rename the **task_util_MPI.f90** to be **task_util_MPI.f9000** and **task_util_NOMPI.f9000** to **task_util_NOMPI.f90 .** Then, delete **all** the files in the OBJ directory and recompile.

Additional Settings in domain.f90: Lastly, in **domain.f90**, there's a provision to set the number of tracer arrays with the variable **ntracers**. If you don't wish to transport any tracers, set **ntracers** to 0.

3.4 Step 4: Compiling gSAM

After the first three steps are completed, it is time to compile the code. Just run the edited **Build** script by executing

```
> Build
```

or, if failed (you may want to change permissions for Build script to executable file and make sure that your shell path variable contains the “dot” . , that is the current directory) try:

```
> csh Build
```

Upon a successful build, an executable file named gSAM will be generated in the root directory. Additionally, you may find various output directories and symbolic links appearing if they haven't been created already.

3.4.1 Checking the executable

To find out the grid size for which this executable has been configured, as well as the physics packages it utilizes, you can run it from the command line using the -info flag. Executing this command will display relevant information, as demonstrated in the following example:

```
..> gSAM -info
 gSAM version: 1.6
 microphysics: sam1mom
 radiation: CAM
 scalar advection: MPDATA
 DOMAIN SIZE:
 nx_gl =       2304
 ny_gl =       1152
 nzm =         74
 nsubdomains_x=        48
 nsubdomains_y=        24
 number of MPI tasks:       1152
```

3.4.2 Checking namelists and input datasets

You can use an interactive command from your terminal prompt to check correctness of your setup before submitting an actual MPI job.

To check the namelists parameters that gSAM executable would read, basic information on the chosen grid, and the ability of model to read supplied initial and boundary datasets, just use the following interactive command:

```
..> gSAM -namelists
```

3.5 Arithmetic Precision in gSAM

Default Precision: By default, gSAM utilizes single-precision calculations. This means most variables and arrays use a 4-byte real representation, even for global runs.

Double Precision: For those desiring more accuracy, gSAM can operate in double precision mode. This implies an 8-byte real representation. To enable this, you'd specify a compiler option to set the default size of the real number to 8 bytes (e.g., the **-r8** flag for the Intel compiler).

Output Precision: Regardless of your choice between single and double precision, all output files generated by gSAM will retain the 4-byte real number format.

Caution: Please be aware that operating in double precision comes with the cost. Your task may run significantly slower, potentially over 50% slower in certain scenarios, and it might consume substantially more memory. This could even result in exceeding available memory capacity of your system.

3.6 Core Utilization

SAM's Core Allocation: SAM adheres strictly to the number of cores or MPI tasks specified in the SRC/domain.f90 file. For instance, if you set nsubdomains_x = 2 and nsubdomains_y = 4, SAM will require exactly 8 cores that should be available to it by the system, not more, not less.

gSAM's Flexibility: Contrary to SAM, gSAM exhibits flexibility in core allocation. gSAM can request more cores than specified in SRC/domain.f90. Any cores exceeding the specified number will remain idle. This feature is useful in high-performance computing environments where core allocation is node-based.

Case in Point - Cheyenne Supercomputer: Consider the Cheyenne supercomputer at NCAR, equipped with 36 cores per compute node. Let's say you want to run gSAM on 1024 cores. The catch is, 1024 isn't divisible by 36. The running script on Cheyenne, however, allocates cores in multiples of 36. If you were using SAM, it would raise an error due to a mismatch between allocated MPI jobs and the subdomains set by SRC/domain.f90. gSAM, on the other hand, adapts by utilizing the required 1024 cores and leaving the surplus idle. Hence, to deploy gSAM on 1024 cores on Cheyenne, you'd have to request a minimum of 29 nodes (calculated as ceiling(1024/36 + 1)). Then, the system will allocate 29x36 = 1044 cores, 20 of which will be idle.

Caution: If you intend to use, for instance, only 8 cores and inadvertently allocate 100 cores, gSAM will happily proceed to run on all 100 cores. This could leave you unaware that 92 of those cores are lying idle while keeping eating your allocation account on a supercomputer!

There are other ways of launching parallel jobs, for example batch systems. However, these issues are system dependent and, therefore, are beyond the scope of this document.

Usually, in interactive mode, you may be able to use a simple command like this (to run on 8 cores as an example):

```
> mpirun -np 8 gSAM
```

To run in the background rather than interactively and using a file for the printout (e.g., with the name out), you could use a slightly expanded command:

```
> mpirun -np 8 gSAM > out 2>&1 &
```

The "out" file will capture then both standard output and error messages (that's why you need *2 > &1*). When running in the background, it's essential to monitor the "out" file to check the model's progress and ensure there aren't any errors during execution, or the model may have 'hung up'. You can do this by 'tailing' the output:

```
> tail -100 out
```

-100 means print last 100 lines of file *out*.

Advice: When using batch jobs on a supercomputer, especially when many cores are allocated, it's crucial to regularly monitor the output printout file. Don't assume that if you check and your job is still shown in the queue marked as running, it is. At times, gSAM might crash, but the system may not necessarily terminate the job for some reason that hard to understand (happens sometimes). As the result, the job might linger for the entire allocated duration, squandering your valuable compute time. Always be vigilant to such occurrences to ensure efficient utilization of resources. So, use a way to constantly "peek" at the standard output file generated by your job to make sure it is being constantly updated as gSAM prints something (like timestep number) every time step. For example, 'watch -n 300 tail -10 log_file' would print out the last ten lines of the log file every five minutes.

4 Setting up a Case

4.1 Files in Case-Directory

To set-up a case, first modify the **CaseName** file. This file contains the name corresponding to a case-directory located in the **CASES/** directory. It could be already created, or you may create a new one. The case directories may contain the following ASCII-formatted files:

prm: This is the file where the Fortran namelists for the model parameters are stored. This file should be present in any case directory. This document will discuss the namelist variables that may be set within the prm file in later sections.

snd: This file holds sounding profiles at different times of height or pressure, potential temperature, water vapor mixing ratio, and wind components (zonal and meridional). The model utilizes linear interpolation, both temporally and vertically, for initializing the atmosphere. It can also be used as an external large-scale forcing data via nudging. Therefore, it's essential to have at least two time-sampled soundings and that the model simulations starts and finishes within the range of times represented by these soundings. If you have only one available sounding, make sure to duplicate it in the snd file with different timestamps to ensure proper initialization and interpolation. However, you can use only one sounding profile (without duplication) if you set the **dosimplesnd** to .**true**.

Example: snd file with two time-levels:

```
 z[m] p[mb] tp[K] q[g/kg] u[m/s] v[m/s]  This line is required
0.     2  1000.   !  time(day)      “number of height levels to read for given time”      “surface pressure (mb)”
0.       -999.  300.  2.  5.  5.
1100.   -999.  300.  2.  5.  5.
100.   2  1000.          ! next time level
0.       -999.  300.  2.  5.  5.
1100.   -999.  300.  2.  5.  5.
```

z[m]: Height in meters specifying the atmospheric level. If pressure levels are used instead, height should be denoted as -999.
p[mb]: Atmospheric pressure in millibars. If undefined, it should be set to -999, and the model will calculate it based on the sounding and surface pressure.
tp[K]: Potential temperature in Kelvin. Negative values indicate that absolute temperature is being used instead.
q[g/kg]: Water vapor mixing ratio in grams per kilogram. Negative values indicate the use of relative humidity instead.
u[m/s], v[m/s]: Zonal and meridional wind components in meters per second.

grd: This file outlines the vertical grid structure in accordance with the Arakawa–C staggered grid system. There are two methods to define grid level heights:

Method 1: Mid-Level Points
In this legacy method, which is still supported for backward compatibility with standard SAM’s cases, you specify the mid-level points of the grid layers in meters. For example:

```
5.
10.
20.
30.
```

Method 2: Layer Interfaces
In this preferred method, you list the heights of the layer interfaces, starting from the lowest layer and proceeding upward. The code identifies which method you are using based on the value of the first level. If the first level is zero, the heights in the file are assumed to represent interfaces between layers. For example:

```
0.
```

```
10.
25.
40.
```

It's not mandatory to list all levels up to the total number of levels **nz_gl**. The algorithm will automatically extrapolate grid levels above the last one specified in the **grd** file. It assumes that the spacing between the last two specified levels will be consistently applied to all subsequent levels.

Note that the **grd** file is not used when the **dz_const** parameter in the **PARAMETERS** namelist is set to **.true**. In such cases, a uniform vertical grid step, denoted by **dz**, should be defined directly in the **PARAMETERS** namelist.

lsf: The large-scale forcing file supplies vertical profiles for various atmospheric tendencies and large-scale winds. This file becomes active when the **dolargescale** parameter is set to **.true**. in the **PARAMETERS** namelist. The format for the **lsf** file is akin to the **snd** file (in terms of setting time levels, number of layers to read and surface pressure), but contains different variables:

z[m]: Height in meters specifying the atmospheric level. If pressure levels are used instead, height should be denoted as -999.
p[mb]: Atmospheric pressure in millibars. If undefined, it should be set to -999, and the model will calculate it based on the sounding and surface pressure.
tpls[K/s]: Tendency of temperature due to large-scale advection.
q[g/kg]: Tendency of water vapor due to large-scale advection.
uls[m/s]: Zonal wind component.
vls[m/s]: Meridional wind components.
wls[m/s]: Large-scale vertical velocity.

Important: If the wls variable is provided (non-zero at least at some levels), it indicates that the temperature and water vapor tendencies are due solely to horizontal advection. The tendency due to the vertical advection due to large-scale vertical velocity (such as large-scale subsidence in PBL cases) will then be computed by the model). If wls is set to zero, the tendencies are understood to be full 3D large-scale tendencies, incorporating both vertical and horizontal components.

sfc: This houses the chronological progression of the SST and surface fluxes. Activate it by setting the **dosfcforcing** parameter to **.true**. within the **PARAMETERS** namelist. In that case the format of data is given by this example:

```
 day sst(K) H(W/m2) LE(W/m2) TAUx(m2/s2) TAUy(m2/s2)
274.00000    301.71799    8.7919998    102.32800    0.0000000
274.12500    301.71799    8.7919998    102.32800    0.0000000
274.25000    302.09299    7.9180002    100.66600    0.0000000
274.37500    302.07101    8.8079996    110.93400    0.0000000
```

….

sst – sea-surface temperature (SST); H – sensible-heat flux; LW – latent heat flux; TAU– total stress due to surface wind.

Special Considerations:

1. **dosfcforcing** only controls the reading of surface fluxes from the **sfc** file.

2. In general, the surface fluxes, whether computed by the model or prescribed, will be used if the **dosurface** parameter is also set to .**true**..

3. To use the prescribed scalar fluxes read from the **sfc** file, the **SFC_FLX_FXD** parameter should be set to .**true**.. To prescribe momentum fluxes, the **SFC_TAU_FXD** should be also set to **.true. .**

4. If the surface fluxes and SST don't change with time, they can be prescribed directly using namelist parameters, so you don't need the **sfc** file in that case, but you still need to set **SFC_FLX_FXD** parameter to true and **SFC_TAU_FXD** to true if you want to prescribe the surface momentum fluxes as well.
   For Latent Heat Flux (LHF) use **fluxq0** (W/m^2)
   For Sensible Heat Flux (SHF) use **fluxt0** (W/m^2)
   For momentum flux use **tau0** (m^2/s^2)

rad: Here, you'll find prescribed radiation heating rates. It's activated when **doradforcing** is toggled to .true. in the **PARAMETERS** namelist. The file format is given by this example:

```
p[mb] or z(m)    (dt/dt)rad [K/s]
0., 33    ! time    Number of levels
  42.500   -0.16042E-04
  200.92   -0.22303E-04
  456.28   -0.24178E-04
  743.00   -0.22789E-04
  1061.1   -0.20891E-04
```
…..

First column represents either pressure in millibars (mb) or height in meters (m). The choice of the coordinate is automatically determined by the model based on the order of values. For height, the order should be increasing; for pressure, the order should be decreasing. Second column is the heating (positive values) or cooling (negative values) rates in K/s. Unlike snd and lsf files, the data in rad file are not required to be set for at least two time-levels.

soil: A file that describes the vertical grid in the soil parameterization among other parameters. As it is a part of the Simple Land Model (SLM), it will be discussed a corresponding section dedicated to setting-up the SLM.

4.2 Run Controls

Most model parameters, as well as logical switches or flags, are configured using the **PARAMETERS** namelist. However, there are specialized namelists for other aspects of the model, such as microphysics, subgrid-scale models, and SLM (Simplified Land Model). Unless otherwise stated, the parameters discussed herein are sourced from the **PARAMETERS** namelist. Parameters originating from other namelists will be explicitly identified.

caseid: This serves as the unique identifier for each run. Together with the name of the case directory, all output file produced during the run will contains this identifier. It's advisable to incorporate detailed information into this string to simplify the process of locating and identifying output files later. For instance, if caseid ='128x128x112_80m_40m_2s_LES' used for PBL case, all associated output files will commence with PBL_128x128x112_80m_40m_2s_LES.

dt - Time step interval (in seconds): In gSAM, the value of **dt** serves as the maximum time step that advances the model from one step to the next. This is a departure from the strategy employed in SAM, where the time step would be halved each time the advection stability criterion was violated. In contrast, gSAM adjusts the time step dynamically and gradually to ensure stability without violating the advection criterion. As a result, the actual runtime of the model may slightly deviate from what would be predicted by simply multiplying **dt** with the current time step number. However, this deviation is typically minimal. Some sub-cycling, similar to what is done in SAM, can still occur. When calculating averages over specific time periods, this minor discrepancy in time steps is accounted for. All time intervals for specific tasks, such as collecting statistics or producing regular model output, should be configured as though the model operates with a constant time step defined by **dt**. Note that the diffusion processes do not affect the change of time step at a given timestep; rather, the SGS eddy coefficients themselves are modified to maintain stability.

nstop: This specifies the maximum number of time steps to execute the case. It's a mandatory parameter. Note that nstop isn't saved in the restart file, so it can be altered prior to each restart. Consequently, this parameter should always be present in the namelist.

nelapse: An optional parameter to designate the elapsed time (in time steps) before stopping. It's useful for resubmitting jobs on systems with CPU time constraints, the so-called *continuation runs*. After completing **nelapse** time steps, the model gracefully concludes after saving restart files.

nrestart: This parameter determines the nature of the run.

- **nrestart = 0**: Represents a new, initial run. For a new run, all output files with the same **caseid** in OUT_* directories should not exist. If they do, the model will terminate when attempting to overwrite those files. This safeguard prevents accidental overwriting of older output files with the same **caseid**, by mistake.
- **nrestart = 1**: Signifies a continuation or restart of an earlier run. During a restart, most of the namelist parameters are ignored and instead sourced from the restart file, with the exceptions being nstop, **nprint**, **nstat**, **nstatfrq**, and the parameters controlling the output of 2D and 3D files.
- **nrestart = 2**: Indicates a branch restart. Here, a new run is initiated using restart file from form some other run (with the same grid). While the run employs the initialization of prognostic variables from another run's restart file, the namelist parameters and forcing fields are considered as if it's a brand-new run. This setting requires the specification of two additional namelist variables:

  - **case_restart**: The branch-root identifier case (the case directory for previous run that is used to branch out a new run)) for a branch restart.

  - **caseid_restart**: This is the branch-root identifier caseid.

day0: Represents the run's starting day-of-year, and it can be fractional number. For instance, to initiate from 6 am on the third day of year (Jan 3rd), you would set **day0** = 3.25. This timestamp is instrumental in the time-based interpolation of data from the **snd**, **lsf**, **sfc**, and **rad** files. It's worth noting that the count starts from January 1st as day 1, not day 0. As the model runs, the current time is calculated from the **day0** reference point. The computation of solar radiation is based on the current day. However, if you're prescribing the solar radiation (with specifics provided in the section detailing namelist parameters for managing radiation computations), the value of **day0** can be zeroed out, rendering the absolute value of the calendar day immaterial. Time is always assumed being UTC.

date0: This is an alternative method to the day0 parameter for defining the start date of a run. It's presented as a 14-digit integer in the format YYYYMMDDHHMMSS, where:
- Y stands for year
- M represents month
- D indicates day
- H signifies hour
- M marks minutes
- S symbolizes seconds.

For instance, using 20220617060130 would pinpoint the time to 6:01:30 on June 17, 2022. Based on this, the model will then calculate the fractional day-of-year (**day0**) and the year (**year0**). The parameters day0 and date0 should not be specified simultaneously.

year0: Represents the starting year of the run. When paired with **day0**, it specifies the precise initial time. It's important to note that there isn't a **year0**=0; the count of years (AD) starts from year 1 (default value). If you're using the **date0** parameter in place of **day0**, the **year0** will be automatically taken from that date.

restart_sep: When set to .true., each MPI task creates a distinct restart file, writing information only for a particular subdomain. **This approach is recommended for simulations involving very large domains and/or thousands of cores and will typically results in much faster writing/reading of restart files.** By default (with restart_sep=.false.), a unified 'serial' restart file is generated, with each MPI task writing sequentially into the same file. Such serial restart files can become exceptionally large for large-scale and global simulations, so users are encouraged to look at the timing output included in the log file at the end of each simulation and setting restart_sep = .true. if more than one or two percent of the runtime is spent in restart_out.

nrestart_steps: This parameter determines the frequency at which restart files are written, based on the number of model time steps. If not specified in the namelist file, the model defaults to using the largest step interval from the set of syeps defined for saving various types of model outputs. It is also required that **nrestart_steps** is divisible by these intervals without a remainder. This approach in gSAM differs from the standard SAM, where the default restart interval is the same as the interval for writing the mean-statistics output (*.stat files).

nrestart_skip: This parameter enables the skipping of restart file writing a specified number of times. By default, with **nrestart_skip** set to 0, restart files are written at each interval determined by **nrestart_steps**. This setting allows for the omission of certain restart file updates; for example, a value of **nrestart_skip** set to 11 means a restart file is written every 12th time it would normally occur. It should be noted that at the conclusion of a run, as determined by **nstop** or **nelapse**, the saving of restart files will always take place, regardless of the **nrestart_skip** setting.

dosaferestart: By activating (setting to .**true**.), the model is forced to safeguard against run loss by maintaining two sets of restart files: one from the prior continuation run and another generated during the current run. These sets alternate between the RESTART and RESTART1 directories. For example, if a run initiates from the RESTART directory, the current run will save new restart files to RESTART1. The subsequent run will then begin from RESTART1 and save files back to RESTART. This cyclical saving process ensures that if writing new restart files fails, the previous set (from previous continuation run) remains available as a fallback. Default is .**false**.

The model logs two crucial parameters in the RESTART/case_caseid_restart.txt file every time a new restart file is created: the timestep of the save point and an indicator of the restart directory (1 for RESTART, 2 for RESTART1). This record aids in tracking the save points and corresponding directories used for restarts.

In the event of a run failure and does not restart because of restart files corruption, it is recommended to copy the backup file RESTART/case_caseid_restart.txt_save, which contains the starting timestep and the directory of the failed run, to RESTART/case_caseid_restart.txt before resubmitting. This procedure guarantees that a simulation can always be restarted from a prior, successfully written restart point, thus providing a reliable fallback mechanism.

4.3 Types of Runs

gSAM can be operated in three core modes: Cartesian rectangular grid mode, akin to the standard SAM, Latitude-Longitude (Lat-Lon) grid mode featuring spherical coordinates, and the so-called Regional or Semi-Open Boundary Conditions Mode. Each mode provides unique configuration options, detailed below.

4.3.1 Cartesian Mode

This is the default mode and employs double-periodic boundary conditions. The grid spacing is uniform in both the x and y directions. In the model, the x-axis points to the East, and y-axis to the North. To set the grid dimensions based on the domain size (configured in **SRC/domain.f90**), two parameters in the **PARAMETERS** namelist are used:

dx: grid spacing (in meters) in the x direction (West-East).
dy: grid spacing (in meters) in the y direction (South-North).

In specialized cases, such as running gSAM in a Direct Numerical Simulation (DNS) mode inside a box with solid walls, domain boundaries can be modified from periodic boundaries using the following namelist parameters:

dowallx: If set to .true., solid walls will serve as domain boundaries in the x direction, replacing the default periodic boundaries.

dowally: If set to .true., solid walls will serve as domain boundaries in the y direction, replacing the default periodic boundaries.

Note that **dowally** is also used in other modes; in particular in global and near-global runs, it is set automatically.

More on running gSAM in DNS mode see the **SGS_TKE** namelist parameters.

4.3.2 Latitude-Longitude Mode

This mode differentiates gSAM from the standard SAM model. In this configuration, a periodic domain is not possible in the y-direction, making **dowally** automatically set to .**true**.. Grid cells near the poles degenerate into triangles, effectively making the pole a wall, one grid-point wide, in lat-lon space. While the grid spacing (in degrees) is constant in the longitudinal (x) direction, it can vary in the latitudinal (y) direction.

Important: Keep in mind, that with Lat-Lon grid, the pressure solver is not based on bi-directional FFT solver in x and y, which is non-iterative procedure, and, in a way, produces an exact solution (down to the truncation error). On the Lat-Lon grid, a hybrid FFT-Geometric Multigrid (GMG) solver, which uses the FFT only in zonal direction, and GMG in meridional. The GMG is an iterative solver and is not "exact". As noted below in section 4.3.3., the precision (or error tolerance) can be adjusted by the **gmg_precision** namelist variable, but high precision (or low error tolerance) may come at relatively large computational cost.

To read the grid coordinates in latitudinal direction you need to set the following namelist parameters:

readlat – this variable instructs the model to read the file with the name specified by namelist variable **latlonfile**. This file can be either ASCII (**latlonfilebin = .false.)** or binary file (**latlonfilebin = .true.**). The file contains the latitude of cell centers (where scalar quantities are defined) that can be read by a single Fortran read statement like:

for ASCII files:

```
read(1,*) lat(1:ny_gl)
```

or for binary files (all recorded values should have type REAL8):

```
read(1) lat(1:ny_gl)   for binary files
```

Latitudes are increasing from south to north. They are negative for Southern Hemisphere and positive for Northern Hemisphere.

Also, in some cases (for example, for regional simulations) it is convenient to read longitudes sd well. In that case, instead of **readlat**, you set namelist variable **readlatlon** to .**true**.. The file pointer is also **latlonfile**, which can also be either ASCII or binary. The difference is that the file content adds data for longitudes, so the Fortran read statements would read it using:

for ASCII files:

```
read(1,*) lon(1:nx_gl)
read(1,*) lat(1:ny_gl)
```

or for binary files (all recorded values should have type REAL8):

```
read(1) lon(1:nx_gl)
read(1) lat(1:ny_gl)
```

4.3.3 Considerations for the Lat-Lon Grid Pressure Solver

When working with the Lat-Lon grid, it's crucial to recognize the differences in the pressure solver's foundation. Unlike the bi-directional FFT solver in Cartesian mode, which offers a non-iterative approach and, to an extent, delivers an exact solution (subject to truncation error), the Lat-Lon grid employs a hybrid FFT-Geometric Multigrid (GMG) solver. This hybrid model utilizes FFT in the zonal direction and GMG in the meridional direction.

The GMG method is iterative by nature, meaning it doesn't guarantee an "exact" solution in the traditional sense. However, you can adjust its precision using the **gmg_precision** namelist variable. Be cautious, as seeking higher precision might significantly increase computational demands. Always strive for a balance between precision and efficiency.

4.3.4 Global run

To initiate a global run, set the **doglobal** parameter to **.true.**. This will automatically configure the x and y grid resolutions to be uniform (using domain sizes to compute the resolution in degrees) unless specified otherwise through the **readlat** parameter.

The parameter **doglobal** =.**true**. will also set **dolatlon** = .**true**. automatically, so you don’t have to set it yourself.

doglobalpresets: if set to **.true**., some namelists parameters and microphysics parameters are set to values to make ‘nice’ simulations in terms of top-of-atmosphere fluxes, etc. These parameters values are based on the previous experience of running gSAM in global configuration. Works only with SAM1MOM microphysics.

Special considerations for global grids

Due to the use of a latitude-longitude grid, the equations exhibit singularities at the poles. To avoid significantly reducing the time step for advection stability, it's advisable to position the centers of the grid cells near the poles at a safe distance away. For instance, the center of the last cell can be at least 1 degree away from the pole. In typical gSAM simulations, the grid spacing varies with latitude. The local resolution in the y-direction (measured in meters, not degrees) is kept consistent (the same) with the x-direction. This results in a grid that is locally isotropic horizontally, offering finer grid spacing in the tropics and especially in mid-latitudes compared to the polar regions. As one approaches the poles, the grid should start to coarsen gradually. This grid arrangement provides better resolution in the tropics and mid-latitudes when compared to a uniformly spaced y-grid with the same number of latitudinal circles. To assist with generating such grids, the **GRIDS** directory includes a Fortran utility called **grid.f90**, which can automatically create an ASCII file with such latitudinal grids, which can be directly read by the model, given the number of grid cells in both longitudinal and latitudinal direction, and desired resolution at the pole.

Even with a grid cell center positioned at least 1 degree away from the pole, the grid step in the longitudinal direction can still be relatively small. As a result, strong zonal winds near the poles could compromise stability and significantly reduce the model's running time step, and therefore, the running time of the whole simulation. To address this issue, it is highly recommended to activate artificial wind damping near the poles. The procedure does not eliminate the winds there but tries to reduce them just enough to maintain the local stability CFL criterion from being violated. This can be achieved by setting the namelist parameter **dodamping_poles** to **.true**. It is highly recommended, for efficiency, to use this option in high-resolution global runs.

4.3.5 Near-Global run

Activated by setting the **donearglobal** to **.true**., solid walls will be placed at certain latitudes away from the poles while spanning 360 degrees in the longitudinal direction. The x-resolution will be computed automatically. To set a uniform grid in the y-direction, use the **dlat** namelist parameter to specify the grid spacing in latitudinal degrees. Do not use the **dy** namelist parameter, as it is reserved for Cartesian grid settings. In this setup, the model will automatically generate a grid that is symmetrical around the equator. Alternatively, you can read the y-grid from a file, as per the general procedures described earlier. This option allows you to create a Near-Global Run that is asymmetrical in the latitudinal direction around the equator, or to implement higher resolution at specific latitudes.

4.3.6 Latitudinal Segment run

This run is analogous to Near-global or global runs, but does not have to span all 360 degrees in longitudinal direction. To make domain smaller in that direction, you need to set grid spacing **dlon** namelist parameter (in degrees) and do not set **donearglobal** or **doglobal** parameter. You would still have to set **dolatlon** to **.true**. though. For y-direction, use uniform grid setting **dlat** parameter or read y-grid from a file. Do not use the **dx** and **dy** namelist parameters, as they are reserved for Cartesian grid settings.

4.4 Regional or Semi-Open Boundary Conditions run

This experimental mode is designed to approximate the Open-Boundary conditions. However, the true open-boundary conditions have not been implemented in gSAM yet. Instead, this mode is designed to simulate a region with lateral boundary conditions nudged to the prescribed fields; for example, taken from reanalysis data or from other model runs for dynamic downscaling studies.

This model works the best for CRM resolution (1 km and coarser), but not so well for LES resolutions.

There are two types of runs in the regional mode: when the boundary conditions depend on the height only, so called 1D-forced regional run and when they are 3D data, or so-called 3D-forced regional run.

4.4.1 1D-Forced Regional run

This subtype is available only when using a Cartesian grid. The model runs with the standard periodic domain but nudges the solution towards 1D profiles within narrow buffer zones in both the x and y directions. This effectively works to negate the periodic nature of the domain, resulting in a non-periodic solution. These profiles are interpolated in both time and space, using data in the **snd** file.

An example of utilizing this mode might involve simulating an island situated within a body of water or a mountain in the middle of the domain, which experience a prescribed incoming flow. In a conventional periodic domain, the flow altered by the island or mountain would re-emerge at the inflow boundary due to the domain's periodic nature. By employing the buffer zones, the model ensures that the incoming flow adheres to the undisturbed, prescribed profiles, thus mitigating the impact of the island on the inflow conditions.

To set up the buffer zones, use the following namelist parameters:

dobufferzonex: Enables a buffer zone along the eastern and western domain boundaries.

dobufferzoney: Enables a buffer zone along the northern and southern domain boundaries.

bufferzonex: Specifies the fraction of the domain width in the x-direction taken up by the buffer zone. This value can range from 0 to 1 but is typically set between 0.1 and 0.2, effectively utilizing 10-20% of the domain width for nudging. Note that buffer zones will be positioned on both sides of the domain, each with a width equal to half of **bufferzonex**.

bufferzoney: Functions similarly to **bufferzonex**, but applies to the northern and southern boundaries.

4.4.2 3D-Forced Regional run

This mode can be used for both Cartesian and Lat-Lon grids.

It can be used for simulations covering substantial areas—spanning hundreds or even thousands of kilometers. It was initially developed for simulating specific large regions on Earth using data from reanalysis or other large-scale model outputs. Given that the actual flow across such expansive regions cannot generally be described by a single profile, 3D fields are employed to guide the simulation within the buffer zones. This method employs wall-like boundary conditions. However, unlike traditional solid wall conditions where the normal velocity component is set to zero, the flow is nudged to align with observed flows, and horizontal pressure gradients are set to zero at the boundaries.

The same buffer zone namelist parameters used in the 1D-Forced method apply here as well. This 3D-Forced method is activated by setting the **doregion** parameter to .**true**.. Furthermore, the model continuously reads the 3D 'observed' fields from pre-prepared files in order to nudge the prognostic fields at the boundaries. Also, it is beneficial to weakly nudge (over relatively long time scale like 6-12 hours) the velocity fields and perhaps temperature to the observations everywhere in the domain too. Detailed instructions for preparing these nudging 'observation' files and how to force specific nudging of the simulations to observations can be found in the "Nudging Runs" section of this document.

4.5 Runs with Cumulus and Cloud parameterizations

For simulations that use very coarse horizontal grids—specifically, those that fall outside the so-called "gray-zone" with grid spacings larger than ten kilometers—the representation of convection and clouds can be significantly deficient. In such cases, the model can employ cumulus and large-scale cloud parameterizations to better represent these phenomena. In the context of gSAM, Kerry Emanuel's convective parameterization (Emanuel and Zivkovic-Rothman, 1999) and Sandrine Bony's large-scale cloud scheme (Bony and Emanuel, 2001) are available for use. This mode is still quite experimental and very little testing has been done, so use it with caution.

Namelist Parameters:

docup: If set to true, the cumulus and large-scale cloud parameterizations are activated. When this is the case, all cloud processes are handled by these parameterizations, and no microphysics parameterization is utilized. You should set the microphysics package to SAM1MOM anyway.

n_cup: This sets the frequency at which the cumulus and large-scale cloud parameterizations are called, measured in model time steps. Normally, the time step of gSAM should not exceed 20 seconds, even when using very coarse horizontal grids, to ensure the stability of gravity waves. Invoking the cumulus parameterization at such short intervals would be both computationally expensive and could introduce excessive noise, given that these parameterizations operate on a statistically defined ‘average’ state. As a result, the tendencies attributable to cumulus and large-scale parameterizations should be updated at intervals longer than a single modeling time step. A general recommendation is to set this parameter for at least 3-minute intervals or longer. For example, for time step **dt** = 10s, **n_cup** can be set to 18 to update tendencies every 3 minutes.

4.6 Runs with RAVE

The model can be run with Reduced Acceleration in the VErtical (RAVE) hypohydrostatic rescaling (Kuang et al 2005). To use it, just specify the so-called RAVE *gamma-factor* setting the namelist variable **gamma_RAVE**. Default value is 1 (no rescaling).

5 Initial and Boundary Conditions

gSAM inherits its initial condition-setting mechanism from the standard SAM, which is both straightforward and simple. Initial profiles, read from a **snd** file, are horizontally uniform. To initiate turbulent flow and break the initial symmetry, you can introduce some initial noise, configurable via the **perturb_type** namelist variable. The perturbations corresponding to these numbers can be found in **SRC/setperturb.f90** file.

In addition to these standard features, gSAM introduces enhanced methods for establishing initial and boundary conditions through pre-prepared binary input files. These files vary depending on the surface type under study, which can be one of three primary categories: oceanic (water), terrestrial (land), or a combination of both (island).

5.1 Surface Types

To specify the surface type for a given run, one and only one of the following logical flags must be set:

OCEAN: If set to .**true**., this indicates that the surface is an ocean.

LAND: If set to .**true**., this indicates that the surface is land.

ISLAND: If set to .**true**., this indicates a mixed environment featuring both water and land surfaces. This parameter is particularly relevant for real-Earth global simulations.

5.2 Specification of SST

The simplest way to specify the sea-surface temperature (SST), assuming that it is horizontally uniform and constant with time, is to use the namelist parameter **tabs_s,** in degrees Kelvin**.** When temperature is changing with time but is the same everywhere in the domain (no horizontal variation), then the surface forcing file **sfc** can be used (see the discussion on **sfc** file above).

For global and near-global simulations, you have the option to specify SST patterns based on pre-defined formulas rather than using a constant SST value. These pattern options can be found in the file **SRC/simple_ocean.f90**, specifically within the **set_sst()** subroutine. You can select a pattern by assigning a specific integer number to the namelist parameter **ocean_type**, which corresponds to your chosen pattern. Two namelist parameters are used to define these patterns: the previously mentioned **tabs_s**, and an additional parameter, **delta_sst**, which relates to the amplitude of SST perturbations for the given pattern. It's important to note that these SST patterns are also applicable when part of the domain includes land, as long as **ISLAND** = .**true**. In such cases, the land temperature is determined through a separate SLM initialization process. Keep in mind that the

SST patterns are only possible when **dosfcforcing** is set to false. Otherwise, the model expects the **sfc** file to be used instead to specify uniform SST.

Note: If your case involves an ocean surface, be sure to set the parameter **doseawater** to .**true**. This adjustment accounts for the lower saturation vapor pressure over saltwater.

5.2.1 Running with the Slab-Ocean Model

Similar to SAM, gSAM also offers the ability to simulate the evolution of SST using a Slab Ocean Model (SOM) instead of prescribing it. To initialize SST in this mode, follow the regular procedure as described earlier. To enable the SOM feature, set the namelist parameter **doslabocean** to .**true**. The depth of the slab ocean is specified through the namelist parameter **depth_slab_ocean**, in meters.

Additionally, two parameters can be set to simulate the ocean's heat transport characteristics:

Szero: Represents the mean heat transport away from the ocean surface for the Slab Ocean Model, measured in W/m².

deltas: Specifies the amplitude of linear variation in ocean heat transport along the x-axis for the Slab Ocean Model, also measured in W/m². The total transport is calculated using the formula **Szero + deltas * |2x/L – 1|**, where L is the domain width.

5.2.2 Running with the Skin-Ocean Model

The Skin-Ocean Model is an alternative to the Slab-Ocean Model, designed for rapid equilibrium between the ocean's Sea Surface Temperature (SST) and all surface enthalpy and radiative fluxes. In this model, the ocean is assumed to have no heat capacity, meaning its SST responds instantaneously to surface forcing. To activate this model, set the namelist parameter **doequilocean** to .**true**.

One limitation of the Skin-Ocean Model is that SST becomes highly sensitive to diurnal cycles, similar to land surfaces, which is not realistically reflective of ocean behavior. To address this, it is recommended to also set the logical flag **dossthomozonal** to .**true**. This parameter ensures that SST is continuously homogenized along latitudinal circles. Of course, it makes sense only is solar radiation depends on longitude and time.

Additionally, the model offers an option for sea-ice formation in polar regions by setting the parameter **doseaice** to .**true**. The sea-ice thickness is either prescribed through the parameter **seaicethickness**, in meters, or can be dynamically calculated by setting **doseaiceevol** to .**true**.

5.2.3 SST homogenization and nudging

Regardless of which interactive SST model is employed—be it the Slab-Ocean or Skin-Ocean Model—you have the option to globally homogenize the predicted SST field. To do this, set the **dossthomo** parameter to .**true**.

Additionally, you can nudge the interactive SST (whether it is homogenized or not) towards a prescribed 'climatological' value over a specific time scale. Activate this feature by setting the **dosstclimo** parameter to .**true**. The 'climatological' SST is specified through the **sst_climo** parameter, in Kelvin. The nudging time scale is set using the **tau_ocean** parameter, in seconds.

5.3 Input files

gSAM allows users to specify initial and boundary conditions via input files containing both 2D or 3D fields. For 2D fields, there is a standard file format (see below) except for clay/sand 2D initial datasets that have their own format (see Simplified Land Model section). The 3D files are used for initialization of dynamics and 3D ozone field in global runs, as well as nudging. The 3D files do not generally have a standard unified format. Therefore, to prepare those binary datasets, see examples in the corresponding section in this document.

5.3.1 gSAM's calendar-day convention

The model currently uses a specific calendar-day convention that is crucial for preparing the binary input datasets. Although not ideal and likely to be updated to a more robust and flexible system in the future, this convention serves its purpose well for most gSAM applications.

For relatively short runs that do not extend beyond a given year, a conventional calendar day is used, which can be fractional. For instance, 6 AM on July 20th, 2022 would be represented as 201.25. You can easily calculate this using a tool like Google with a query such as "calendar day 2022". The input datasets should utilize this calendar-day convention for the time variable. Note that January 1st has a calendar day of 1, not 0. So, 12 PM on January 1st corresponds to a calendar day of 1.5.

If both your input datasets and the model's run period are confined to a specific year, a simple calendar day will suffice, which will always be less than 365 (or 366 in a leap year).

Complications arise when your simulation spans multiple years. In such cases, the calendar day must be computed from January 1st of the year 0001 using the following formula:

$calday\_new=(year-1)\times 365+calday\_of\_year$

Here, *year* represents the actual year in question.

Example:

Let's say your run starts on July 20th, 2022, and ends on February 15th, 2023. You need to prepare forcing datasets (e.g., SST) that cover this time range. Assume these are daily datasets starting on July 18th, 2022, and ending on February 16th, 2023. Dates should be chosen to allow the model to perform linear interpolation between time samples.

For September 1st, 2022, the calendar day would be (2022-1)×365+244 = 737909. Similarly, for February 1st, 2023, the calendar day would be (2023-1)×365+32=738062.

5.3.2 Standard 2D file format

It's crucial that all 2D fields—like those designating surface properties or terrain height—correspond with the model's grid, as no real-time interpolation is performed. This is especially relevant when the simulation domain includes both land and water surfaces (**ISLAND=.true**.). For example, if SST data is provided on a different grid from the model's and includes land areas, determining how to interpolate this data onto the model's water surface grid points becomes problematic. Similar issues arise with land cells. Therefore, the preparation of specialized input datasets conforming to a specific model grid is left for the user and should be conducted in a preprocessing stage before the model run.

To enable reading from an input file, a corresponding logical flag in the appropriate namelist must be set to .true. (further details will be provided later). Each field requires a separate file and all are written using the same standard 2D format as described below. The exception are fields related to initialization of soil variables that use different format (described in the section on SLM). The standard 2D file format is best explained through a sample Fortran-like program below that writes a compliant binary file for a given field. Note that compliant binary files can also be generated using other programming languages, like NCAR Command Language (NCL). In fact, most binary files for gSAM have been generated in the past by the author using NCL.

Standard 2D file format example:

```
integer ntime                       ! number of time samples in a file
real(4) time(ntime)                 ! an array of times (gSAM's calendar-day), also 4-byte real
real(4) field(nx_gl, ny_gl, ntime) ! all fields are 4-bite real

open(1, file=filename, form='unformatted')
write(1) ntimes
write(1) nx_gl
write(1) ny_gl
write(1) time(1:ntimes)
do i=1,ntimes
  write(1) field(: , : , i)
end do
```

The time() array should be monotonically increasing. If the dataset is not cyclic (more on that below), start time (in fractional gSAM's calendar days) should be equal to or smaller than the initial time for a run, set by the parameter **day0** (or **date0**). The last time sample—time(ntimes)—should be at or beyond the time when the run concludes. These conditions ensure the proper linear time interpolation of boundary data. If only one sample is present in the file (ntimes=1), it can either be used for field initialization (in which case the timestamp time() is irrelevant), or the field will be applied continuously throughout the run without any interpolation.

5.3.3 Cyclic datasets

You can make all input datasets that have several time samples cyclic or periodic, that is repetitive from the beginning when the end time is reached. For that you need to set the namelist parameter **docyclic** to true. Also, in that case, you need to set the period of the cycle , set by namelist integer variable **cycle_period**, in days. The period cannot be a fractional day, only whole days. By default, it is 365, or annual cycle. In cyclic dataset, the first time point, time(1) should be greater than or equal to 0, and the last one, time(ntime) should be smaller than **cycle_period**. Note that the **docyclic** applies to all time-dependent input datasets.

5.3.4 2D boundary files controlled by namelist PARAMETERS

Note: Fields that control the initialization and forcing of the Simple Land Model (SLM) are specified in the **SLM** namelist. Detailed information on setting up this model will be provided in the respective section of this document.

Land mask

The landmask field designates which surface grid points are covered by land (landmask = 1) or water (landmask = 0). This is relevant when the parameter **ISLAND** is set to **.true**..

readlandmask – when set to **.true**, the model reads the land mask from a file indicated by the parameter **landmaskfile**.

File format: Standard 2D

Sea-Surface Temperature

The SST file can be used to specify either a static single field or time-evolving SST. In the latter case, the model's runtime should fall within the time range of the SST dataset, as the model will linearly interpolate the SST for each time step.

readsst – when set to **.true**, the model reads the SSTs from a file indicated by the parameter **sst0file**.

File format: Standard 2D

Terrain

Detailed information on setting up the terrain will be provided in the respective section of this document.

readterrain – when set to .**true**, the model reads the terrain heights from a file indicated by the parameter **terrainfile**.

File format: Standard 2D

5.3.5 Making invariant datasets in GLOBAL_DATA

The **GLOBAL_DATA** directory contains several handy utilities designed to help to create the landtype and terrain boundary conditions for both global and regional simulations.

Start by downloading the **gSAM_Global_Data.tar.gz** file from the same source as the model. Once you've decompressed this file, it will reveal a directory containing multiple universal datasets. It's essential to be aware that this directory can be relatively extensive, exceeding 1.5 GB. Ensure you allocate it to a storage space that can handle large files. Then, in the **GLOBAL_DATA** directory, create a symbolic link in the gSAM directory named DATA. This link should point directly to your dataset. Use the following command:

```
ln -s <path_to_data_directory> DATA
```

The directory includes a landtype file extracted from a 300-m global land cover classification dataset, which can be found at Climate Copernicus. I've adapted this dataset to fit the IGDP land types. Additionally, the directory has a terrain file that also contains the ocean floor depth in addition to the land terrain. This file is derived from the ETOP01 Arc-Minute Global Relief Model courtesy of NOAA.

The file **GLOBAL_DATA/MAKING_IC_BC** contains detailed instructions on how to produce the lanftype and terrain netcdf files tailored to the gSAM grid. The **GLOBAL_DATA** directory also contains the NCL scripts that I used in the past to generate initial and boundary conditions in binary format for gSAM that you can use as templates.

5.3.6 3D file conventions

Unlike 2D fields, 3D fields can exist on grids different from the model's grid. These fields will be interpolated both spatially and temporally, except when used for initial conditions. By default, all 3D datasets are assumed to be periodic in the longitudinal (x) direction but not in the latitudinal (y) direction. Even if a Cartesian grid is used for your run, by default, the input data should be on

Lat-Lon grid. To change that and instruct the model that the horizontal distances in 3D input files are in meters, not degrees, set the **read_meters** namalist parameter to .**true**.. For Cartesian grid, gSAM will calculate the corresponding latitudes and longitudes for all grid points based on the **longitude0** and **latitude0** parameters (in degrees), which are set in the **PARAMETERS** namelist. Note that **longitude0** represents the longitude of the western boundary's scalar grid point, and **latitude0** is the latitude at the center of the domain.

gSAM uses longitudes in degrees-east only, ranging from 0 to 360 degrees. If your dataset uses a degrees-west convention, which ranges from -180 to 180 degrees, set the flag **dofliplon** to .**true** to read the dataset correctly.

For global simulations, the input data should cover latitudes from -90 to 90 degrees, adhering to the gSAM convention where negative latitudes correspond to the Southern Hemisphere and positive ones to the Northern Hemisphere. The latitudes should start at the South Pole and progress to the North Pole. Be careful as some reanalysis products like ERA5 write data from North Pole to the South Pole order.

For near-global simulations, the input data's minimum and maximum latitudes should fall outside the model grid's corresponding ranges. In the longitudinal direction, the data should remain periodic.

For regional simulations, you don't have to adhere to global extents in either longitude or latitude. However, the minimum and maximum latitudes and longitudes of the input data should encompass the respective ranges of the model grid you're using.

5.3.7 3D initial-conditions dataset

To understand the file format for 3D initial-conditions datasets, consider the following example of a Fortran code that writes such a file:

```
real(4) u(nx, ny, nz)      ! zonal wind (m/s)
real(4) v(nx, ny, nz)      ! meridional wind (m/s)
real(4) w(nx, ny, nz)      ! vertical velocity (Pa/s) – units are m/s if w3D_pressure=.false.
real(4) t(nx, ny, nz)      ! absolute temperature (K)
real(4) q(nx, ny, nz)      ! water vapor mixing ratio (kg/m3)
real(4) qc(nx, ny, nz)     ! cloud water mixing ratio (kg/m3)
real(4) qi(nx, ny, nz)     ! cloud ice mixing ratio (kg/m3)
real(4) qr(nx, ny, nz)     ! rain water mixing ratio (kg/m3)
real(4) qs(nx, ny, nz)     ! snow/graupel mixing ratio (kg/m3)
real(4) z(nz)   ! heights (m)
real(4) p(nz)  ! pressure (mb)
real(4) lon(nx)  ! longitudes (degrees)
real(4) lat(ny)  ! latitudes (degrees)
```

```
open(1,file=filename,form='unformatted')
write(1) nx, ny, nz
write(1) z(1:nz)
write(1) p(1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) u(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) v(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) w(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) t(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) q(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) qc(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) qi(1:nx,1:ny,1:nz)
write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) qr(1:nx,1:ny,1:nz)

write(1) lon(1:nx)
write(1) lat(1:ny)
write(1) z(1:nz)
write(1) p(1:nz)
write(1) qs(1:nx,1:ny,1:nz)
```

Note: If some microphysics fields like qr or qs are missing, replace them with arrays filled with zeros.
Important: By default, it is assumed that the w(:,:,:) array in initial-conditions file is pressure vertical velocity (in Ps/s), NOT ordinary vertical velocity (in m/s). If you created the initial dataset with actual vertical velocity, not pressure velocity, then you would have to instruct the model that w() in initial dataset is in fact vertical velocity by setting **w3D_pressure** namelist variable to **.false**. (the default is **w3D_pressure = .true.**)

To force gSAM to initialize the run with the data in your initial 3D dataset, set the namelist flag **readinit** to **.true**., and provide the path to the 3D file using the namelist variable **initfile**; for example:

readinit = .true.
initfile = './GLOBAL_DATA/BIN_D/init_era5_2020012000_1440x721x37.bin'

5.3.8 Important Considerations for Real-Earth Global Runs

For accurate real-Earth global simulations with gSAM using initialization from reanalysis datasets like ERA5, it's crucial to use data on pressure coordinates rather than height coordinates. The paper by Khairoutdinov et al. (2022) provides details on why this is important. In effect, gSAM global runs operate on pressure coordinates. Don't try to interpolate the data on pressure coordinates to height coordinates for creating the initial dataset! Also, 3D output from gSAM should also be compared to observations or reanalysis data in pressure coordinates, not height coordinates. Just use the pressure profile in 3D output files as pressure coordinate.

The best would be to write global arrays from reanalysis to initial-data file, without any interpolation, horizontal or vertical. Let gSAM interpolate the data to its grid. For the p() array in the example above, use pressure coordinate values from the reanalysis dataset and the global-mean geopotential height profile for the z() array. Don't warry about terrain cutting onto your initial fields; the model will take care of that.

While this guideline is indispensable for accurate global simulations, its importance diminishes for regional runs and is virtually negligible for small-domain simulations. Ignoring this guideline could result in your global simulations quickly diverging from the reanalysis data, rendering any forecasts meaningless within just one simulated day.

5.3.9 Initialization with CAMs Single-Column Model’s input file

There is an alternative way of setting the initial and forcing profiles that does not need **snd**, **lsf** and **sfc** files, that is using a standard NCAR SCAM’s (CAM single-column model) input file in netcdf format. This capability has been inherited from SAM.

doscamiopdata – if .true., then the case setup is done using a SCAM file

dozero_out_day0 – if .true., forces the initial calendar day be 0

iopfile – specify the path and name of the SCAM input file

6 Nudging to Data

Nudging is a technique used in modeling to gently guide the model's generated variables toward observed or pre-determined values using so-called Newtonian relaxation. The goal is to ensure that the model reproduces large-scale features that are consistent with actual observations, or other reliable data sources, without entirely overriding the model's internal dynamics. There are two types of nudging in gSAM —1D and 3D—each suited for different modeling scenarios.

6.1 1D Nudging

This type of nudging is typically employed when the model is run using a Cartesian grid and a periodic domain.

In 1D nudging, the focus is on the horizontal mean profiles of the 3D variables (such as temperature, pressure, or wind speed), rather than their local or point-specific values. The model's average conditions over a horizontal layer are adjusted or "nudged" toward the observed or prescribed average conditions for that layer, so that the specific value of the nudged field at each individual point within that layer would not necessarily be adjusted to match specific observed values.

1D nudging affects the following 3D variables: the horizontal wind components, temperature and water vapor mixing ratio. The reference profiles are generally taken from the **snd** file and, therefore, can change in time. For horizontal velocity components, the reference profiles can also be taken from the **lsf** file when **dolargescale** parameter is set to .**true**.

To activate the nudging for specific fields, the following logical flags should be set to .**true**.:

donudging_uv – for nudging horizontal wind components

donudging_tq - for nudging both temperature and total nonpreciptating water (vapor+cloud water)

donudging_t - for nudging temperature

donudging_q - for nudging total nonprecipitating water

By default, nudging would be done at all model levels. However, a user can specify a range of heights between which the nudging will be applied using the following parameters:

nudging_uv_z1, nudging_uv_z2 – nudging height boundaries (z2>z1) for horizontal wind components, in meters.

nudging_t_z1, nudging_t_z2 – nudging height boundaries (z2>z1) for temprtature, in meters.

nudging_q_z1, nudging_q_z2 – nudging height boundaries (z2>z1) for total nonprecipitating water, in meters.

The nudging exponential time-scale is set by these parameters:

tauls – nudging time-scale for all variables if **tautqls** is not set

tautqls – nudging time-scale for temperature and total nonpreciptating water

6.2 3D Nudging

This technique is particularly useful for complex scenarios such as global or regional model runs. Unlike 1D nudging, which relies on a single average profile, 3D nudging adjusts the model's three-dimensional prognostic fields (e.g., temperature, wind, water vapor) towards a prescribed 3D dataset, such as reanalysis, or an output of some large-scale model for dynamic-downscaling runs.

The timescale for nudging is important. If this timescale is too short, the simulation could become overly constrained, essentially replicating the target dataset without any meaningful internal dynamics. Conversely, if the timescale is too lengthy, the model might diverge significantly from observed conditions. Generally, for global simulations, the nudging timescale is recommended to be longer than 6 hours to maintain a balance between model freedom and adherence to observed data.

6.2.1 3D Nudging Data Format

Data File Structure

The data files used for 3D nudging closely resemble the binary format used for the model's initial 3D conditions, as described earlier. This means that these nudging files might use a different grid from the one used by the model. If this is the case, the model will automatically interpolate the data upon reading.

Required Fields

File formatting conventions and the binary data format align with those of the initial conditions file, with one exception: the 3D nudging files omit the qc, qi, qr, and qs fields.

Temporal Variability

If only a single file is used, the nudging data is assumed to be time-invariant. For time-varying prescribed fields, multiple files corresponding to different time instances are used. The model assumes that the run time falls within the period covered by the nudging data files. It then reads and linearly interpolates the data to align with the current time step of the run.

Look-Up File

To guide the model in locating and time-matching the nudging files, a look-up file must be created. This ASCII text file generally resides in the same directory as the nudging files and lists the following information:

- The total number of time samples
- The specific times corresponding to each sample
- The filenames for each time sample

Example of a Look-Up File

```
5                        ! Total number of time samples
70.                      ! Time (in gSAM's calendar days) of the first sample
70.5                     ! Time of the second sample
80.
80.5
90.
Tropics_nudge_era5_70.0.bin  ! Filename of the first sample
Tropics_nudge_era5_70.5.bin  ! Filename of the second sample
Tropics_nudge_era5_80.0.bin
Tropics_nudge_era5_80.5.bin
Tropics_nudge_era5_90.0.bin
```

Add the following variables to the **PARAMETERS** namelist in your **prm** file to specify the file locations:

nudge3D_dir: The directory path containing the nudging files.
nudge3D_file: The full path to the ASCII look-up file.

Example

```
nudge3D_dir = './GLOBAL_DATA/NUDGE'
nudge3D_file = './GLOBAL_DATA/NUDGE/nudge_info.ascii'
```

6.2.2 Enabling 3D Nudging

To activate 3D nudging, set the **donudge3D** parameter to **.true.**. You can also specify the time-scale for nudging using the **nudge3D_tau** parameter, which is in seconds. Field-specific nudging can be controlled by flags **donudging_***, which are the same as used for 1D nudging. For vertical velocity nudging, the flag **donudging_w** is also available. Note that height-range parameters for nudging are disregarded in 3D nudging, as all model grid levels are nudged.

6.2.3 Cyclic nudging

You can make nudging datasets cyclic or periodic, that is repetitive from the beginning when the end time is reached. For that you need to set the namelist parameter **docyclic** to true. Also, in that case, you need to set the period of the cycle , set by namelist variable **cycle_period**, in days. By default, it is 365, or annual cycle. In cyclic dataset, the first time point, time(1) should be greater then or equal to 0, and the last one, time(ntime) should be smaller than **cycle_period**. Note that the **docyclic** applies to all time-dependent input datasets.

7 Dynamical Core Controls

7.1 Main parameters

nadv_mom – integer parameter that sets the order of advection scheme used for momentum. The choices are 2 (default), 3, or 23, which is the hybrid (mixture) of 2 and 3 (see Khairoutdinov et al 2022 for details).

alpha_hybrid – the real parameter that controls the fraction of tendencies produced by 2nd-order scheme in the hybrid scheme (**nadv_mom**=23), that is =1 means that the scheme is pure 2nd-order, and =0 is 3rd order. Note that the 3rd-order scheme is very diffusive (strong damping of small scales), therefore, for noise control, it is recommended to set it to 0.8 or so. Also, the 3rd -order scheme is much less stable; therefore, the model may need smaller timestep, and hence, would take longer time. (see Khairoutdinov et al 2022 for details).

gmg_precision – maximum error of iterations in GMG solver. Used only for Lat-Lon grid. Usually set to 1.e-5 or 1.e-6 for single-precision runs and should never be larger than 1.e-4. For double precision runs, will be automatically set to a values smaller by three orders of magnitude than set for single-precision runs. Default 1.e-6.

Note: the 2nd order scheme for momentum sometime produces small-scale noise in simulations with topography, especially steep topography or buildings. If you noticed that issue, try to use **nadv_mom=23** with **alpha_hybrid=0.8**. It also tends to mitigate noise issues in global high-resolution simulations.

docoriolis – if .true., the Coriolis force is applied (default = .false.)

docorioulisz – if .true, the vertical Coriolis patameter is also applied (default = .false.)

dofplane – if .true., the Coriolis parameter is constant everywhere (f-plane approximation); otherwise, it will vary in y direction. The center of the domain is set by **latitude0**. In the case when **dofplane**=.false. and **docoriolis**=.true., the **dowally** should be set to .true., as periodic boundary conditions in y direction won't make sense. Don't forget to set **docoriolis** flag to .true. to have the Coriolis force on. (default = .true.)

fcor - Coriolis parameter (1/s) in the case when **dofplane**=.true.. Don't forget to set **docoriolis** flag to .true. to have the Coriolis force on.

dometric = if .false., calulate metric curviture terms (with tangent) will be switched off for lat-lon grid (default = .true.)

doflat - if .true., use Cartesian coordinates even when **doglobal** or **dolatlon**=.true. (default = .false.)

donodynamics – if .true., wind is not evolving (frozen). (default = .false.)

dofixdynamics – if .true., prescribed 3D wind field is set by **setfixdynamics.f90** using the integer case-switch **fixdynamics_type.** Be careful with setting the wind as the flow mass-flux should still be non-divergent. (default = .false.)

donobuoyancy - if .true., buoyancy is not computed (default = .false.)

perturb_type – integer parameter that is used to choose custom initialization of 3D fields, including addition of random noise, as listed by the **setperturb.f90**. Note that default is 0. To avoid adding any noise set it to -1.

7.2 Weather-forecasting controls

In weather forecasting scenarios, future SSTs and sea ice conditions are typically unknown. Unlike climate simulations where these factors can be prescribed from datasets that evolve over time, forecasting requires a different approach. In the context of weather forecasting, you generally initialize the model with a specific dataset for SST and sea ice, that may have many time snapshots, that is fixed at the values at initialization time.

dofcast: When set to .true., this parameter ensures that the model will not update the SST values during the simulation, effectively freezing them at the initial conditions.

dofcast_ice: Setting this to .true. will fix the sea ice conditions throughout the forecast period, based on the initial data used for model initialization.

For generation of ensembles, the following parameters can be used:

nensemble: This is an integer ranging between 1 and 100. It signals the model to read perturbation profiles for temperature and water vapor from a file located at **RUNDATA/tqpert**. This approach is commonly used with Cartesian models where the initial conditions are supplied via a **snd** file. The specified perturbations are added to the profiles read from the **snd** file.

nens: This integer parameter is particularly useful when you initialize the model by reading a 3D initial-conditions file. It's an integer that serves to seed the random-number generator differently for each ensemble member, creating initial noise variations. The type of noise is set by **perturb_type** integer parameter (see setperturb.f90 for choices).

7.3 Damping Controls

In gSAM, damping controls serve two primary functions:

1. To minimize the reflection of vertical gravity waves in the upper part of the model domain.

2. To maintain stability based on the CFL advection stability condition, thereby avoiding unnecessary reduction of the time step, which can prolong the simulation. This is especially important for managing local updrafts and dealing with the "pole problem," where zonal grid spacing can become exceedingly small.

Here are the relevant namelist parameters for controlling damping:

dodamping: If set to .true., this activates a sponge layer in the upper part of the model domain where damping of gravity waves occurs. Note that this setting only dampens the vertical velocity component.

nub: A real number between 0 and 1 that specifies the relative height at which the sponge layer begins. Beyond this height, damping strength will incrementally increase until it reaches its full force near the top of the domain. The default is 0.6.

dodamping_w: If set to .true., damping of vertical velocity throughout the domain is activated to ensure that the simulation adheres to the CFL stability criterion. Damping will be initiated if the local CFL number, based on local vertical velocity, exceeds the **damping_w_cu** parameter.

damping_w_cu – sets the critical CFL based on vertical velocity that initiates its damping when dodamping_w is .true.

dodamping_u: Functions similarly to **dodamping_w** but applies only to horizontal velocities and is only activated above a pressure level of 70 mb unless it is near the poles. It is further conditional on the **dodamping_poles** parameter being set to .true., indicating that the simulation is a global run.

damping_u_cu – This parameter sets the critical CFL value based on horizontal velocity. Damping will be initiated when either **dodamping_u** or **dodamping_poles** (or both) are set to .true..

dodamping_poles – If set to .true., this enables the damping of horizontal velocities near the poles to enhance stability. Damping is activated at latitudes higher than 80°S and 80°N. Initially, the damping effect is weak, but it intensifies as it approaches the poles. This setting is highly recommended for global simulations.

Note on CFL Stability: The maximum CFL number varies depending on the advection scheme. For a second-order scheme (**nadv_mom**=2), the maximum CFL is 0.72. For a third-order scheme (**nadv_mom**=3), it's 0.55. For a hybrid scheme (**nadv_mom**=23), stability ranges linearly between 0.35 and 0.72 based on the value of **alpha_hybrid**, which ranges from 0 to 1. Keep in mind that stability also depends on horizontal velocity, so it's advisable to set **dodamping_w** at least 25% lower than the maximum CFL.

Monitoring CFL: The model's standard printout includes the maximum CFL values (CFL – all velocities, CFLH for horizontal velocities, and CFLZ for vertical velocity) at each time step. Ideally, the model should not have to subcycle (i.e., NCYCLE > 1) frequently. If it does, consider reducing the time step or lowering the **dodamping_w_cu** or **dodamping_u_cu** value depending on which, CFLZ or CFLH, is too high. Use caution when adjusting this parameter; setting it too low (below around 0.35 for **nadv_mom**=2) could adversely impact the simulation.

7.4 Initialization of Fluid Motion in Simulations

The initialization of fluid motion is a crucial step in initiating the flow especially for horizontally uniform initial conditions. Two primary methods are available for this purpose in a new run: the introduction of initial random noise in the boundary layer or the specification of a 'warm bubble'.

Here are the key namelist variables that control the initialization:

perturb_type: Specifies the type of perturbation introduced. It can be set to various integer values, with the default being 0, which initializes white noise in the temperature field near the surface. The **SRC/setperturb.f90** file outlines the specific types of initialization corresponding to different values of this parameter. For instance, setting this value to 2 would create a warm bubble. No initial perturbation corresponds to **perturb_type** set to -1.

Parameters for Warm Bubble

If you choose to initialize a warm bubble (when **perturb_type** is set to 2), the following parameters control its characteristics:

bubble_x0, bubble_y0, bubble_z0: These parameters set the coordinates of the bubble center, in meters. Note that the horizontal coordinates **bubble_x0, bubble_y0** in x and y directions, respectively, are coordinates relative to the center of the domain, so **bubble_x0 = 0., bubble_y0 = 0.** will place the bubble in the horizontal domain's center.

bubble_radius_hor: Sets the horizontal radius of the bubble, also in meters.

bubble_radius_ver: Sets the vertical radius of the bubble, in meters.

bubble_dtemp: Specifies the temperature perturbation within the bubble in Kelvin, relative to the ambient environment. The temperature perturbation follows a cosine-squared function, with its maximum at the center of the bubble and zero at the edges.

bubble_dq: Sets the water vapor perturbation in kg/kg, also relative to the environment. Similar to temperature, the water vapor perturbation changes as a cosine-squared function and is zero at the bubble edges.

8 Subgrid-Scale (SGS) Parameterization

The SGS parameterization in gSAM has several unique features compared to the standard SAM model, offering increased flexibility and improved simulation performance. The primary objective of the SGS model is to calculate the subgrid-scale (SGS) eddy-diffusivity and eddy-viscosity coefficients, which are then applied as second-order diffusion to all prognostic variables.

8.1 Key Differences from Standard SAM

gSAM departs from the standard SAM's third-order Adams-Bashforth scheme for momentum equations and instead employs a forward-in-time Euler scheme. This modification increases the stable CFL number for momentum diffusion from approximately 0.1 to 0.5, that is similar to diffusion of scalars.

The computation of the mixing length in the equations controlling the eddy-coefficients has been revised. Unlike the standard SAM that relies solely on vertical grid spacing, gSAM also takes into account horizontal grid spacing. This is particularly helpful for simulations with anisotropic grids and tends to improve the PBL profiles in that case.

In gSAM, the diffusion does not impact the time step needed for stability. The coefficients are auto-adjusted to not exceed the local CFL stability criterion.

gSAM allows you to set minimum eddy-diffusivity and eddy-viscosity coefficients. This is primarily to minimize small-scale noise due to gravity waves, particularly in areas with high local static stability and low shear.

8.2 Direct Numerical Simulation (DNS)

gSAM offers the capability to operate in DNS (Direct Numerical Simulation) mode. In this setting, diffusion coefficients are aligned with molecular values. The application scenarios for DNS are confined primarily to simulations between two plates, featuring a doubly periodic domain horizontally, and for flows within enclosed spaces like boxes or chambers. In DNS mode, no-slip boundary conditions are used on all solid boundaries. Boundary heat fluxes are computed using prescribed surface temperatures. For vapor fluxes, the surfaces are assumed to be moist (covered with liquid water).

Central to DNS is the emphasis on molecular diffusion and viscosity. Consequently, the time step when operating in DNS mode is predominantly influenced by the stability of diffusion processes. Contrarily, in other gSAM modes, the eddy-viscosity and eddy-conductivity coefficients are adjusted to meet a stability criterion rooted in the advection processes' stability. To found out the value for maximum permissible time step in the DNS mode, utilize the gSAM -namelists command and take a look at the SGS namelist printout.

Notably, the DNS mode demands an exceptionally high resolution, often necessitating measurements in mere millimeters for air. Given the precision required, it's recommended to utilize double precision when compiling the model. This can be achieved by promoting standard floating numbers (real) to real(8) during compilation (refer to the section discussing model compilation for further details).

8.3 Namelist SGS_TKE

SGS_TKE Namelist Parameters

dosmagor – If set to .true., the Smagorinsky-type SGS closure will be used instead of the prognostic TKE 1.5 order closure.

tk_factor – controls minimum eddy-diffusivity and eddy-viscosity coefficients. Typical (non-dimensional) value to set is 0.01. Default is 0.

dodns – when set to .true., the model will use constant diffusivity and viscosity coefficients, effectively operating as a DNS model. By default, the domain is doubly periodic in horizontal.

dochamber – when set to true, the model will perform a DNS in a box or chamber. **dodns** will be set to true automatically.

DIFF_DNS –sets the kinematic diffusivity coefficient (m^2/s) value when **dodns** is .true. Default is set for diffusion coefficient for air.

PR_DNS – Specifies the Prandtl number (the ratio of viscosity to diffusivity coefficients) when **dodns** is set to .true. Default is for air.

T_top – temperature (K) of the top plate in DNS mode.

T_bot – temperature (K) of the bottom plate (K) in DNS mode.

T_wall – temperature (K) of walls in **dochamber** is set to true.

9 Radiation

gSAM offers two options for radiation models: NCAR's original Community Atmosphere Model (CAM) and the updated RRTMG found in later versions of NCAR's Earth System Community Model. Your choice of radiation model is configured in the Build script. Both models share the following set of namelist parameters:

doshortwave – If set to .true., activates solar radiation transfer.

dolongwave – When set to .true., enables the calculation of longwave or infrared radiation.

doradlon: When .true., accounts for longitude-dependent solar insolation. Useful for large computational domains. Note: this is incompatible with doubly-periodic domains. By default, radiation is calculated using a fixed longitude defined by the **longitude0** parameter.

doradlat: If .true., the model considers latitude-dependent solar insolation. Also meant for large computational domains and incompatible with doubly-periodic domains. Default behavior is to compute radiation for the latitude specified by the **latitude0** parameter.

nrad: Specifies how often the model updates radiation heating rates (in model time steps). For instance, **nrad** = 20 means radiation updates occur every 20 time steps. These calculations are computationally expensive, so frequent updates are not recommended. Generally, for deep convection, 3-5 minutes frequency is adequate. For global runs, 15 minute interval is typically used.

nrad_ems: Defines the frequency for updating emissivity/absorptivity coefficients in RAD_CAM, which depend mostly on water vapor. As these coefficients are very expensive to compute (much longer than the radiation itself), they are updated every **nrad** * **nrad_ems** steps, generally at a lower frequency than the nrad updates (i.e., nrad_ems >> 1). Used only by RAD_CAM.

doperpetual: When .true., the model simulates perpetual sunlight with an intensity matching that of the normal sun for the day specified by the **day0** parameter.

dosolarconstant used together with the **doperpetual** flag set to .true..

solar_constant – value of the sun solar input when **doperpetual** = .true. Generally, it is not equal to the actual solar constant as for **perpertual** run, there is averaging over the whole day.

zenith_angle – average value of the zenith angle when **doperpetual** = .true.

doseasons – If set to .true., incoming solar radiation will change according to the current calendar day, allowing for both seasonal and diurnal cycles.

doradforcing – if .true., prescribed radiation from the rad file will be applied

doradhomo – Activates horizontal averaging of radiation heating rates when set to .true.. This option should only be used with Cartesian grids. Note that 2D radiation diagnostic is not affected and shows the horizontal variability of radiative fluxes before horizontal homogenization.

doradhomozonal – Enables zonal (east-west direction) averaging of radiative heating rates. This should generally not be used in real-Earth simulations and is mostly intended for aquaplanet scenarios. Note that 2D radiation diagnostic is not affected. Generally this option should not be used with real-Earth simulations, only for aquaplanet.

doradsimple – When .true., the model uses an analytical expression for radiation rates, often used for Large Eddy Simulation (LES) of stratocumulus clouds following Stevens et al. (2005). By default the rad_simple() function is called; however, when dosmoke is set to .true., a different function rad_simple_smoke(), based on Bretherton et al. (1999), is called instead.

NOTE: If doradsimple=.false. and dolongwave and/or doshortwave are .true., the domain top should be in stratosphere, preferably above 30 km. Otherwise, the radiative heating rates computed with RAD_CAM or RAD_RRTM will be extremely inaccurate.

doradforcing - if .true. , apply prescribed radiation heating rate profiles from file **rad.**

nxco2 – Sets a multiplier for current CO2 levels, based on radiation data files. For example, nxco2=2 signifies a double-CO2 scenario. The default CO2 concentrations are 367 ppmv for RAD_CAM and 355 ppmv for RAD_RRTM.

n2ox, ch4x, cfc11x, cfc12x – if set to nonzero numbers, can overwrite the corresponding gases concentrations (units g/g) in radiative calculations.

notracegases - if .true., no trace gases are used except for CO2

doequinox – If .true., the model will use equinox solar conditions, setting the calendar day for radiation to a perpetual day 80.

reado3 – instructs the radiation to read the 3D ozone dataset instead of the default ozone profile from **trc** file stored in RUNDATA directory. The file path is set by **o3file** parameter. The file format for ozone is similar to the one used for 3D model initialization but can be for multiple time steps, and only one field is written - ozone (units g/g). File format is illustrated by the following example:
real(4) o3(nx, ny, nz, ntimes)
integer ntimes ! number of time samples
real(4) times(ntimes) ! calendar days

```
open(1,file=filename,form='unformatted')
write(1) nx,ny,nz
write(1) ntimes
write(1) times(1:ntimes) ! gSAM's calendar day
do i=1,ntimes
  write(1) lon(1:nx)
  write(1) lat(1:ny)
  write(1) z(1:nz)
  write(1) p(1:nz)
  write(1) o3(1:nx,1:ny,1:nz, i)
end do
```

10 Terrain

One of the standout features of gSAM (as compared to the standard SAM) is its ability to incorporate terrain effects. This feature is implemented in a straightforward manner, making it easy to use while maintaining the core numerical capabilities of the model.

The method (see Khairoutdinov et all 2022a, 2022b) aims to halt the flow's velocity over a single time step in the grid cells marked as terrain or buildings. Users only need to prepare a 2D dataset representing terrain height, which the model will then read and use to prepare internal datasets automatically.

In regional simulations, the terrain height dataset should represent relative heights, so that at least one grid cell should have a zero height.

For global runs, the ocean height is considered to be zero.

Types of Terrain

- Predefined patterns (e.g., bell-shaped mountains).
- Custom terrain read from a file.

Namelist PARAMETERS settings

doterrain - When set to .true, the model expects terrain specifications.

terrain_type - Controls the type of prescribed terrain pattern to use. The options for this are defined in the terrain.f90 file.

doterrepair - Instructs the model to "repair" the terrain by smoothing out one-point sharp peaks and filling a-point-wide pits (a single point hole) that could introduce noise or instability. It's highly recommended to set this to .true (which is also the default setting).

readterr - When set to .true, terrain heights will be read from an external file.

terrainfile - Specifies the file name (including path) from which the terrain data will be read when readterr is set to .true. The terrain height file should be in a standard 2D file format and express heights in meters.

npressure_iter – The method used for stagnating the flow over a single time step may leave some small residual flow in the simulation. To address this issue, this parameter specifies the number of iterations to be used for minimizing the residual flow. The default is no iterations. From the experience, the iterations can reduce the residual flow by approximately a factor of two per iteration. For more details, consult the study by Khairoutdinov et al. (2022b). However, each iteration involves solving a 3D elliptic pressure equation, which can be computationally expensive. For global-scale simulations, it's generally recommended to avoid using iterations due to the computational burden and potentially minor impact on the results. Note that the residual flow will not affect the conservation of mass, such as water vapor, as the fluxes of scalars are artificially set to zero at the terrain surface.

11 Simplified Land Model (SLM)

The theoretical formulation of SLM follows the paper by Lee and Khairoutdinov (2015). However, gSAM adds several additional features that were not in the original formulation in standard SAM. The main feature is the inclusion of interactive snow over land. The snow is represented as a single layer with prognostic depth and temperature. In addition, the surface imperviousness was added as the prescribed variable that controls the blocking of rain penetration into the soil or evaporation from the soil. Also, sea ice was added. Currently, sea ice is not represented by prognostic model, but rather is a prescribed field with prescribed thickness. However, the sea-ice temperature is predicted. Also, some changes to the numerical representation and processes were implemented that will be described elsewhere in the future.

The SLM uses one layer of vegetation over interactive soil. It supports 17 classes of land cover or landtypes as defined by the International Geosphere–Biosphere Programme (IGBP) classification. The landtypes are

0 – water and sea ice
1 - evergreen needleaf forest
2 - evergreen broadleaf forest
3 - deciduous needleaf forest
4 - deciduous broadleaf forest
5 - mixed forest
6 - closed shrublands
7 - open shrublands
8 - woody savannas
9 – savannas
10 – grasslands
11 - permanent wetlands
12 – croplands
13 – urban
14 - croplands/natural mozaics
15 – permanent ice (glaciers)
16 - baresoil

Here is the link to details of landtypes: http://www.eomf.ou.edu/static/IGBP.pdf

Landtype data should be consistent with landmask data. They should be on the same grid, and non-zero landtype values should correspond to landmask=1, whereas landtype=0 should correspond to landmask=0.

A landtype index in each surface grid point automatically defines many parameters of vegetation and soil, such as visible and near-infrared albedos, canopy roughness length, root parameters, stomata resistance parameters, basal area index, IR emissivity, displacement height, etc. None of these parameters need to be specified by hand. The only vegetation parameter that needs to be specified in addition to the landtype is the Leaf Area Index (LAI). Note that some landtypes do not require LAI, such as landtypes 0, 11, 13, 15 and 16. Other landtypes do require explicit specification of LAI to work correctly. The LAI cannot be generally defined by the choice of landtype as it generally depends on geographic location and season.

The SLM is driven by the incoming radiation; therefore, the SLM needs the interactive radiation activated by dolongwave and doshortwave set to true in namelist **PARAMETERS**. Also, no perpetual sun (**doperpetual**=.true.) is allowed in that case.

The SLM also has an interactive soil model. The soil is composed of nsoil layers. The nsoil is hardcoded in the file slm_vars.f90 (currently 9). However, the vertical soil layer thicknesses are not hardcoded, but need to be specified by the file **soil**, which should be present in any case-directory, which uses SLM. Besides the soil layer thicknesses (in the order from soil surface to bottom), the file **soil** specifies initial profiles of soil temperature, soil wetness (from 0 to 1, with 1 corresponding to the maximum moisture holding capacity, or field capacity of soil), sand and clay content in percent, and also the soil relaxation factor, which controls the nudging of the soil prognostic temperature and wetness to initial profile if one wants to minimize the 'climate drift' from desired state. The clay and sand contents are needed to compute the soil hydrological properties. Note that the clay and sand contents in soil file can be overwritten by SLM namelist parameters clay0 and sand0, or can be read from a 2D data file.

There is also a way to overwrite the initial soil temperature and wetness profiles set in file **soil** by using **soilt0** and **soilw0** namelist parameters. The profiles will be uniform with height.

While the soil layer thicknesses are defined only by the file **soil**, and cannot be overwritten by any other method, the other parameters (except for forcing factors) in that file can be overwritten using the variables in namelist SLM. Note though that in that case, the soil parameters will be constant with depth. See the namelist SLM for further details.

In order to activate the SLM, the following parameters in **PARAMETERS** namelist should be set:

dosurface = .true.

LAND = .true. or **ISLAND** = .true. (not both)

SFC_FLX_FXD = .false.

Namelist SLM:

landtype0 – Sets the landtype for all grid points in the model domain to a uniform type.

LAI0 – Sets the Leaf Area Index for all grid points that require LAI specification to a constant value.

clay0 – Specifies the clay content in the soil (in %) across the entire model domain as a percentage. Overrides values from file **soil**.

sand0 – Specifies the sand content in the soil (in %) across the entire model domain as a percentage. Overrides values from file **soil**.

dosoiltnudging – nudge the soil temperature profile to initial profile using the forcing factor profile, defined also in file **soil** (last column). The nudging term at each soil level is multiplied by the forcing factor, so 1 at some soil level means full nudging, and 0 would mean no nudging. (default false)

dosoilwnudging – nudge the soil wetness profile to initial profile using the forcing factor profile, defined also in file **soil** (last column). The nudging term at each soil level is multiplied by the forcing factor, so 1 at some soil level means full nudging, 0 would mean no nudging, and 0.5 would mean nudging time-scale twice as long as set by **tausoil**. (defalt false)

tausoil – the time-scale (s) of nudging the soil profiles to the initial profiles (defined in **soil** file). (default 86400)

soit0 – Specifies the initial uniform temperature for all soil layers in Kelvin. Overrides values from file **soil**.

soilw0 – initial uniform soil wetness (all levels), from 0 (bone dry) to 1 (saturated). Overwrite the initial profiles set by file **soil**.

snow0 – Sets the initial snow thickness in meters.

snowt0 – Sets the initial snow temperature, in K. If snow0 is set but snowt0 is not, the model will make a guess on snow temperature.

imperv0 – Sets a uniform imperviousness value for the soil, ranging from 0 to 1. A value of 1 means the soil is completely impervious to water. By default, all datatypes have imperviousness 0, except for urban landtype 13 which, by default has imperviousness of 0.75.

readlandtype, readinitsoil, readsoil, readLAI, readseaice, readsnow, readsnowt, readimperv: Logical flags to read corresponding map data from files.

landtypefile – a string containing a path to the file with landtype map read when **readlandtype** is set to true. Uses a standard 2D binary format.

LAIfile – a string containing a path to a file with LAI map read when **readLAI** is set to true. Uses a standard binary 2D format.

seaicefile – a string containing a path to the file with sea-ice mask (0 – no ice, 1 – sea-ice) read when **readseaicetype** is set to .true. Uses a standard binary 2D format.

snowfile – a string containing a path to the file with snow-depth (in meter) map read when **readsnow** is set to true. Uses a standard binary 2D format.

snowtfile – a string containing a path to the file with snow-depth (in meter) map read when **readsnowt** is set to true. Uses a standard binary 2D format.

impervfile – a string containing a path to the file with imperviousness read when **readimperv** is set to true. Uses a standard binary 2D format.

soilfile - a string containing a path to the file with clay and soil content read when **readsoil** is set to true. The content is the same for all soil grid levels, that is only horizontal variation is allowed. The file has its own binary format illustrated by the following example:

```
real(4) clay(nx_gl, ny_gl), sand(nx_gl, ny_gl)  ! unites are in %
open(1, file=filename, form='unformatted')
write(1) nx_gl
write(1) ny_gl
write(1) clay(1:nx_gl,1:ny_gl)
write(1) sand(1:nx_gl,1:ny_gl)
```

soilinitfile - a string containing a path to the file with initial 3D soil temperature and wetness data when **readinitsoil** is set to true. The file has its own binary format illustrated by the following example:

```
integer nsoil ! number of soil layers
real(4) soilt(nx_gl, ny_gl,nsoil). ! temperature (K)
real(4) soilw(nx_gl, ny_gl,nsoil) ! wetness (0 to 1)
real(4) zsoil(nsoil) ! depth of soil layer midlevel, in meters
open(1, file=filename, form='unformatted')
write(1) nsoil
write(1) nx_gl
write(1) ny_gl
do i=1,nsoil
 write(1) soilt(1:nx_gl,1:ny_gl,i)
end do
do i=1,nsoil
 write(1) soilw(1:nx_gl,1:ny_gl,i)
end do
```

The model will interpolate local soil profiles to fit its own grid if the soil depths specified in **soilinitfile** differ from the model's soil layer depths.

12 Microphysics

12.1 Common parameters

There are five microphysical packages in gSAM that should be chosen in the csh-script Build before the compilation of the code. There is also an experimental MICRO_KH3 microphysics, which is an extension of SAM_1MOM microphysics by adding a prognostic variable for pristine ice to include the Bergeron process in mixed-phase clouds. It is not formally documented yet, and still under development and testing. The main namelist parameters that control formation of clouds and precipitation apply to all microphysics schemes are set by the namelist **PARAMETERS**. They are:

docloud – if .true., allow cloud formation. If false, the transport of water vapor still occurs, and the surface latent heat fluxes still computed. Default is .false.

doprecip – if .true., precipitation formation is allowed. Default is .false.

12.2 SAM_1MOM

This is the original SAM's single-moment microphysics package. With only two water-related prognostic variables, it's the fastest option and is recommended for expensive large-domain or global simulations. Despite its simplicity, it produces results that are very reasonable compared to observations. A key update in gSAM is the extension of 1D vertical arrays of various microphysics coefficients to 2D fields, taking into account height and latitude variations of temperature.

Namelist MICRO_SAM1MOM

qcw0 – the threshold (in kg/kg) for original SAM’s Kessler-type autoconversion of liquid water to rain. Default 1.x10^{-3} kg/kg

doKKauto - if .true., the original SAM’s Kessler type autoconversion and accretion of cloud water by rain will be replaced with the Khairoutdinov and Kogan (2000) formulation (so-called KK-scheme). In that scheme, the autoconversion depends not only on cloud water, but also on concentration of droplets. The latter is prescribed by parameter **Nc_ocn** (default 50 cm^{-3}) over ocean (landmask=0), and **Nc_land** (default 300 cm^{-3}) over land (landmask =1). Default is .false.

doKKaccr - if .true., the accretion of cloud water by rain from original SAM will be replaced with the Khairoutdinov and Kogan (2000). Default .false.

do_scale_dependence_of_autoconv – If true, autoconversion rate will depend on the horizontal grid resolution. Specifically, the rate will scale as (100/dx)^2, where dx is the local horizontal grid spacing in meters. This only works when **doKKauto** is set to .true.. Default is false.

qci0 – the threshold (in kg/kg) for conversion of pristine ice into snow/graupel. Can be used as a "tuning knob" to affect TOA radiative fluxes and hence precipitable water. Default 1.x10^{-4} kg/kg.

icefall_fudge – a "fudge factor" to multiply the pristine ice sedimentation velocity as used by SRC/MICRO_SAM1MOM/ice_fall.f90 to increase of decrease it. Generally smaller than 1. It is also a nice "tuning" knob to increase/decrease anvil sizes and hence to affect the TOA radiation fluxes and precipitable water as well, especially in tropical regions. For instance, reducing it will decrease ice sedimentation rate, thus increasing anvil size, reducing OLR, and increasing precipitable water. The default is 1.0.

donomicro – if.true., suppress microphysics, but carry vapor around. Default .false.

dowarmcloud – if .true., suppress ice microphysics. Default .false.

docloudfall – compute sedimentation of cloud water assuming lognormal distribution with the relative standard deviation of the normal distribution specified by **sigmag** (default 1.5) parameter. Default .false.

doeiscld – parameterize warm cloud fraction to be used by the radiation module based on Estimated Inversion Strength (EIS). Applied only for low clouds over the oceans. May substantially improve TOA fluxes simulations, especially for coarse grids (more than 10 km spacing). Default .false.

Note on global model tuning: The **qci0** and **icefall_fudge** parameters can be effectively used for "tuning" of the top-of-atmosphere radiative fluxes and precipitable water. For example, to make the model "wetter", one can decrease **icefall_fudge** from default value of 1.0. Their value also depend strongly on grid spacing. For example, for 4.25 km at the equator, the "best" value for **icefall_fudge** is 0.8, while for 17 km grid spacing at the equator, **icefall_fudge** = 0.3 can make the model global mean precipitable water close to IRA5. However, for radiation fields, especially for coarse resolution, even better results can be obtained setting doeiscld to .true..

12.3 M2005

This the a two-moment microphysics scheme by Morrison et al (2005). It has been imported from the WRF model. It offers more complex and realistic simulations involving various water and ice species, as well as aerosols. The default values for namelist paramneters are found in SRC/MICRO_M2005/microphysics.f90 file.

Namelist MICRO_M2005

doicemicro – if .true., use ice species (snow/cloud ice/graupel)

dograupel – if .true., graupel is used for falling ice rather than hail

dohail – if .true., graupel species has qualities of hail

osb_warm_rain – if .true., use Seifert & Beheng (2001) warm rain parameterization in place of Khairoutdinov and Kogan (2000)

dopredictNc – if .true., predict cloud drop number based on CCN concentration

dospecifyaerosol – if .true., specify two modes of (sulfate) aerosol (see aer_rm1, aer_rm2)

dosubgridw – if .true., use estimate of subgrid w in microphysics

doarcticicenucl – if .true., use arctic parameter values for ice nucleation

docloudedgeactivation – if .true., activate droplets at cloud edges as well as base

Nc0 – prescribed droplet number concentration (#/cm3)

ccnconst, **ccnexpnt** – CCN power activation spectrum (N=C S^k) parameters, #/cm3

aer_rm1, **aer_rm2** - two modes of aerosol spectrum used when **dospecifyaerosol**=T

aer_sig1, **aer_sig2** - geom standard deviation of aerosol size distribution

dofix_pgam – if .true., specify the gamma in gamma distribution for cloud droplets

pgam_fixed – gamma-parameter

douse_reffc – if .true., compute cloud water effective radius for radiation

douse_reffi – if .true., compute ice-crystal effective size for radiation

dorrtm_cloud_optics_from_effrad_LegacyOption, - - self-explanatory

dosnow_radiatively_active – - self-explanatory

do_scale_dependence_of_autoconv – allow heuristic scaling based on dx

do_scale_dependence_of_activation – allow heuristic scaling based on dx

do_output_micro_process_rates - self-explanatory

doeiscld – parameterize warm cloud fraction to be used by the radiation module based on Estimated Inversion Strength (EIS). Applied only for low clouds over the oceans. May substantially improve TOA fluxes simulations, especially for coarse grids (more than 10 km spacing). Default .false.

12.4 THOM

This is the Thompson et al (2008) microphysics package has been imported from the WRF model.

Namelist MICRO_THOM

Defaults are set in SRC/MICRO_THOM/micro_params.f90

doicemicro – if .true., use ice species (snow/cloud ice/graupel)

doaerosols – if .true., use Thompson-Eidhammer scheme with water- and ice-friendly aerosols. (NOT YET AVAILABLE).

doisotopes – option to enable computation of water isotopologues. (Coming soon.)

Nc0 – prescribed droplet number concentration (#/cm3)

fixed_mu_r, fixed_mu_i, fixed_mu_g – gamma exponents for rain, cloud ice, graupel.

dofix_mu_c, fixed_mu_c - option to specify pgam (exponent of cloud water's gamma distn)

do_output_process_rates - self-explanatory

dosnow_radiatively_active – self-explanatory

dorrtm_cloud_optics_from_effrad_legacyoption – self-explanatory

douse_reffc – if .true., compute cloud water effective radius for radiation

douse_reffi – if .true., compute ice-crystal effective size for radiation

12.5 P3

This is so-called P3 microphysics by Morrison and Milbrandt (2015), imported from the WRF model. The namelist are described in SRC/MICRO_P3/microphysics.f90 file. Defaults are set in SRC/MICRO_P3/micro_params.f90.

Namelist MICRO_P3

iWarmRainScheme – choice of warm rain microphysics

log_predictNc – If true, predict cloud-drop concentration

aerosol_mode1_radius, aerosol_mode1_sigmag, aerosol_mode1_number, aerosol_mode2_radius, aerosol_mode2_sigmag, aerosol_mode2_number - if **log_predictNc**==true, the cloud droplets will be activated from one aerosol mode whose properties can be input here. If the activation code is extended, a second mode will be used as well. radius in m, **sigmag** dimensionless, number in kg^{-1}

Nc0 - initial/specified cloud droplet number conc (#/cm^3)

dofix_pgam - fix value of gamma exponent for cloud DSD

pgam_fixed - value of gamma exponent for cloud DSD

nCat - number of free ice categories

MaxTotalIceNumber - maximum number concentration for all ice categories combined

typeDiags_ON - logic variable, for diagnostic hydrometeor/precip

model - type of model that is used in P3_MAIN subroutine

n_diag_2d - the number of 2-D diagnositic variables output in P3_MAIN subroutine. This allows the user to create additional 2D diagnostics within the P3_MAIN subroutine and have them passed back to the gSAM microphysics() subroutine. (default = 1).

n_diag_3d – Similar to **n_diag_2d**, but for 3D diagnostics.

douse_reffc – if .true., compute cloud water effective radius for radiation (be careful with changing this; better just leave it .true. except for testing)

douse_reffi – if .true., compute ice-crystal effective size for radiation (be careful with changing this; better just leave it .true. except for testing)

do_output_micro_process_rates – logical parameter

12.6 DRIZZLE

The MICRO_DRIZZLE microphysics used the drizzle water parameterization based on Khairoutdinov and Kogan (2000) microphysics. It is a warm-rain microphysics with no ice; therefore, it is only suitable for the shallow clouds, preferably stratocumulus-topped boundary layers.

Namelist MICRO_DRIZZLE

Nc0 – prescribed droplet number concentration (#/cm3)

13 Tracers

13.1 Radiatively active smoke

dosmoke – if .true., the water vapor variable will be replaced with radiatively active smoke variable following Bretherton et al. (1999). There is no surface flux of smoke. The profile of smoke is initialized by **snd** file in place of water vapor variable. The **docloud** and **doprecip** are automatically set to .false. The interactive radiation is also switched off. The only way to make smoke radiatively active is to set **doradsimple** to true. In that case the radiation heating rates will be computed using the subroutine in **rad_simple_smoke.f90**. If **doradsimple** is false (default), then the smoke will be just a passive tracer.

13.2 Passive Tracers

You can add arbitrary number of tracers to be transported by the model. The current implementation assumes passive tracer, but the tracer physics can also be easily implemented by supplying the subroutine **tracers_physics()** in module tracers, which is found in **SRC/tracers.f90**. All tracers, when activated, will be automatically advected and diffused by the model.

To activate the tracers, one need to set the parameter **dotracers** to true in namelist **PARAMETERS**. The number of tracers **ntracers** should be specified before the compilation of the model in **SRC/domain.f90**.

There are two ways to initialize tracers:

1. Initialization subroutine **tracers_init()** located in the module tracers. This subroutine currently initializes all tracers to zero.

2. Point sources using subroutine **tracers_source()**, also in module tracers. This subroutine allows you to define point sources for tracers. Numerous example point sources can be

found in the module. They are self-explanatory. A particular source is chosen by using the case-switch namelist variable **tracer_source_type**.

The point source option is activated by setting **dopointsource** to .true.. The namelist paramemeter **pointsource_start_step** controls the time-step at which point-source tracers will start release (every time step), which is useful for delaying release, perhaps to allow turbulence to establish first. By default, the release will start at the very first time-step.

To allow tracer settlement at the surface by turbulent flux to the surface using Monin-Obukhov theory, set **dotrsfcflux** namelist parameter to .true. The code will then compute the turbulent flux of tracers into the surface assuming that all tracers that are brought to the surface “stick” to it. By default, tracers have no surface flux.

The model outputs horizontally averaged statistics for each tracer, labeled as TR01, TR02, etc. Three-dimensional fields for tracers are also part of the standard 3D output. Additionally, certain diagnostic fields such as the tracers' vertical integral, near-surface concentrations, and accumulated amount of tracers at the surface, will be included in the standard 2D output in directory OUT_2D.

14 Output

The output from gSAM generally falls into three categories: domain average statistics, 2D horizontal fields, and 3D snapshots. These statistics are time-averaged over regular intervals. The 2D fields may or may not be time-averaged, while the 3D fields usually are not. Beyond these basic outputs, the model can generate specialized outputs, such as 3D variables averaged over a certain horizontal scale, and highly compressed quantities useful for quick visualization.

The output is written in the following directories:

OUT_STAT: Time-height horizontally averaged statistics.

OUT_2D: 2D x-y fields (refer to SRC/write_fields2D.f90).

OUT_2DL: 2D x-y fields from the Simplified Land Model (see SRC/SLM/slm_write2D.f90).

OUT_MISC: Miscellaneous 2D x-y fields (refer to SRC/write_fields2DM.f90).

OUT_INV: Contains time-invariant 2D and 3D fields, including terrain masks, SLM's invariant physical fields, etc.

OUT_3D: Captures 3D snapshots of various fields.

OUT_MOMENTS: Contains 3D fields of diverse statistics on a degraded grid.

OUT_MOVIES: Files specifically tailored for visualization and film creation.

These directories are directly accessible in the gSAM root directory. However, you can also set them as symbolic links to actual directories located on external storage spaces, which is the recommended setup for managing large data volumes. See the section of this document on editing the **Build** file.

To optimize storage space and expedite output—since some files can be quite large—the 2D and 3D data are, by default, written in compressed form using 2-byte integers. However, there is also an option to use 4-byte floating-point format or binary format. Regardless of the chosen format, you would need to convert the datasets into NetCDF files in a post-processing stage, using the conversion utilities found in the UTIL directory.

If you install the Parallel NetCDF library (pNetCDF), the 2D and 3D outputs (not statistics output) can be directly written as NetCDF files while model is running. However, my experience indicates that direct parallel output to NetCDF can be considerably slower—up to several times—than binary and compressed outputs for large runs involving thousands of compute cores (that output is highly optimized). But you can experiment yourself to see which is faster on your system. The post-processing the output files using a less resource-intensive analysis cluster (other than production computer) can be more efficient in terms of resources as the production time can be too valuable to waste on inefficient output. For smaller domains and fewer MPI tasks, or for debugging and testing the model, the direct output to pNetCDF can be indeed quite convenient and less time consumimg.

14.1 Standard Printout

When the model runs, it constantly prints some axillary diagnostic information to tell you whether the run is progressing fine (and to help to identify the nature of problems when run is not progressing fine). The printout from the model is written into the standard terminal output, but can be redirected into a file. When running in a batch mode, the standard output is always written to a file.

Initially, the model prints out comprehensive information about the grid structure, resolution, initial datasets, profiles, and so on. Eventually, it enters a main time loop, during which gSAM outputs a single line of information after each time step, which looks like the following:

```
NSTEP =   4682 NCYCLE= 1 dtn= 15.00 CFL=0.52 CFLH=0.51 CFLZ=0.52 CFLG=0.40 niter= 1 err= 0.30E-06
```

NSTEP refers to the current time-step since the start of the run, while NCYCLE represents the number of time adjustments (or subcycling) that the model performed on the current step to maintain CFL stability. The variable dtn indicates the time-step used. CFL, CFLH, and CFLZ are Courant numbers for 3D, horizontal, and vertical velocities, respectively, and CFLG is the Courant number for diffusion. The latter two parameters, *niter* and *err*, are relevant only for Lat-Lon grids when a multigrid iterative pressure solver is used. The *niter* refers to the number of V-cycles executed to achieve the maximum relative error, represented by *err*. Generally, a smaller *niter* value results in more efficient code. The *err* value is largely controlled by the namelist parameter **gmg_precision**, which is usually set to 1.e-5 or 1.e-6 for single-precision runs and should never exceed 1.e-4. As a rule of thumb, choose a **gmg_precision** that keeps *niter* below 2-3. For double-precision runs, **gmg_precision** can be set to a value smaller by four orders of magnitude. Note that for Cartesian-grid runs, both *niter* and *err* are always zero.

Every so often, based on the number of time-steps, the model outputs more detailed diagnostics, including the minimum and maximum values of various variables, moisture conservation, and so on. The frequency of these detailed printouts is controlled by the namelist parameter nprint. For instance, if nprint is set to 30, the extensive printout will occur every 30 time-steps. Keep in mind that generating extended printouts consumes computational resources, so they should not be too frequent; ideally, nprint should be set in the range of 30-100 unless more frequent printouts are required for debugging and testing.

An important metric to monitor in these printouts is the variable div, which holds the maximum and minimum values of the mass-flux divergence. In an anelastic model, this value should theoretically be zero but generally differs from zero because of rounding errors or limited iterations with the FFT-GMG pressure solver. However, as long as the absolute maximum of the divergence does not exceed 1e-7, there is generally no cause for concern. This is especially true in a relatively small domain with several hundred grid points in the horizontal dimension and when using a Cartesian grid with an FFT-pressure solver. However, if this value exceeds, or is significantly greater than 1.e-4 in global runs, it may indicate that the model is not functioning correctly. For Lat-Lon grid configurations using an iterative solver, this divergence can be sensitive to the maximum error defined by the previously mentioned parameter, **gmg_precision**. Generally, the maximum and minimum values of div in global runs should not be larger in absolute terms than 1.e-5.

14.2 Statistics Output

The model generates plenty of statistics, including horizontally and time-averaged fields. These averages are calculated over discrete time periods, resulting in a file with time-varying statistics. Output files with the .stat extension are saved in the OUT_STAT directory and are written in an internal binary format. To convert these statistics files to NetCDF format, use the utility stat2nc found in the UTIL directory. For instance, the command

```
UTIL/stat2nc OUT_STAT/PBL_128x128x96.stat
```

will produce a NetCDF file named OUT_STAT/PBL_128x128x96.nc.

Below is a list of relevant variables in the **PARAMETERS** namelist:

nstat: The number of time steps over which statistics are averaged to create a single time point, which represents the average time over the sampling period.

nstatfrq: The number of samples gathered over the nstat time steps.

Example: If you have a 10-second time step, and you want the statistics to be computed and averaged over one-hour intervals, collecting samples every 2 minutes, then set:
nstat = 360
nstatfrq = 30

doSAMconditionals: If set to .true., both core (suffix: COR) and downdraft core (suffix: CDN) averages will be written to the *.stat file. Be aware that this will significantly increase the size of the stat file due to the large number of fields.

dosatupdnconditionals: If set to .true., averages for cloudy updrafts (suffix: SUP), cloudy downdrafts (suffix: SDN), and cloud-free environments (suffix: ENV) will also be written to the *.stat file. The threshold for defining a cloud is 10^(-5) kg/kg. This means small quantities of cloud water and ice may be present in the environmental air. Again, this could considerably enlarge the stat file due to the many fields involved.

LES_S: the threshold for deciding which grid cell is cloudy or not, for statistics diagnostics. When set to .true. (default), the threshold is 0. Otherwise, it is set to minimum between 1.e-5 and 1% of the local saturation mixing ratio over water.

cwp_threshold: threshold for cloud water/ice path for diagnostics of cloud fraction. Default: 0.01 kg/m2. Also used for diagnostic of 2D cloud-fraction output in *.2D files.

14.3 2D Output

The parameters below govern the creation of horizontal 2D diagnostic output files. While standard SAM produces a single type of 2D output file, gSAM allows for various types, each saved in distinct directories (as previously described). By default, these fields are captured in a compressed internal format using 2-byte integers. Also, separate files are created for each time step by default. Although compressed data saves on storage, it does compromise accuracy when decompressed. Thus, if high-precision analysis is needed, users can opt to write 3D output data using a 4-byte floating-point binary format. Additionally, for smaller domains or shorter simulation runs, users can opt to append the data into a single file.

Every 2D output file, regardless of its type, comes with a .2D extension and can be converted to a NetCDF format via the **2D2nc** utility, housed in the UTIL directory. This utility is programmed in Fortran and must be compiled to generate an executable. To do this, you'll need to adjust the Makefile in the UTIL directory and run the standard make utility.

The command-line syntax for the utility is as follows:

```
2D2nc input.2D [optional: latlon]
```

Here, input.2D refers to the 2D input file, which can be in the model's internal, binary, or compressed format. The optional latlon argument substitutes the Cartesian x-y coordinates in the resulting NetCDF file with latitude-longitude coordinates. This option is redundant for Lat-Lon simulations, as those NetCDF files will already utilize latitude-longitude coordinates. The converted NetCDF files will be saved in the same directories as their corresponding input 2D files.

Important Note: It's practically infeasible to manually convert all the individual 2D output files for a specific simulation run. Therefore, automation scripts are advised for this purpose. The SCRIPTS/FILES directory offers sample csh scripts capable of auto-converting files. These scripts can even be set up for parallel processing over multiple compute cores, assuming your batch system supports such functionality. For an illustrative example, refer to the csh script named 2D2nc_all_parallel.csh.

Namelist variables controlling 2D output

The parameters for controlling 2D output fields are largely consistent across various types of output files, written in different output directories. To simplify the description and avoid repetition for different file types, we'll use the wildcard symbol **$**. In this context, **$** can mean: **2D** for output to OUT_2D, **2DL** for OUT_2DL, **M** for OUT_MISC, and **2DZ** for OUT_2DZ. Note that a couple of exceptions are specific to standard 2D output.

nsave$: Specifies the number of time steps in a time interval over which 2D fields are sampled. If set to .true., the fields will be averaged over this period; otherwise, they will be saved as snapshots at the end of each sampling period.

nsave$start: Sets the initial time step for 2D data sampling. If this time step exceeds the value of the nstop parameter, no 2D output files will be created.

nsave$end: Sets the final time step for 2D data sampling. Sampling will cease for time steps that exceed this value.

save$bin: When set to .true., the output will be saved in a floating-point binary format. Default: F

save$sep: If .true., the 2D output for specific time interval will be saved as a separate file for the specified sampling period. If set to .false., all data will be consolidated into a single file. Default: T

save$avg: If .true., each 2D field will be averaged over the specified sampling period. If .false., only snapshots at the end of the sampling period will be saved. Default: F

dogzip$: When set to true, the gzip utility will be used for additional compression of the output file. Be aware that this could considerably slow down the process for large files. Default: F

save$netcdf: When set to true, the 2D output files will be directly written in NetCDF format. Note that this requires the pNetCDF library to be installed. Default: F

save2Drada: Unique to standard 2D output, this parameter, when set to true, will save time-averaged radiative fluxes, specifically when save2Davg is set to .false. Default: F

save2Dradac: Also unique to standard 2D output, this parameter saves fields of accumulated radiation flux from the beginning of the simulation, along with the total precipitation accumulation field. Default: F

14.4 3D output

The parameters for controlling the snapshots of 3D fields are similar to their 2D counterparts, with a notable exception: 3D data is always saved as a snapshot, that is, without any time averaging. By default, 3D files use an internal 2-byte-integer compressed format, but a more accurate binary output option is also available. While 3D files are typically saved as separate files, users running smaller domains or shorter simulations can choose to append 3D data into a single file.

Due to their large size, generating 3D files can be computationally expensive. This time expense can be reduced by opting to write 3D data across multiple files, a feature controlled by the namelist variable **nfiles3D**. Using multiple files generally speeds up 3D output considerably.

3D files should be converted to NetCDF format during post-processing using the **3D2nc** utility located in the UTIL directory.

The syntax is as follows:

```
3D2nc input.3D [optional: latlon]
```

Here, input.3D refers to the 3D input file in either internal, binary, or compressed format. The optional latlon argument replaces the Cartesian x-y coordinates in the resulting NetCDF file with latitude-longitude coordinates. This is redundant for simulations already in Lat-Lon format. The NetCDF files will be saved in the same directories as their corresponding 3D input files.

When **nfiles3D** is set to a value other than 1 (which is the default), use the **3Dsep2nc** utility with similar syntax. Files in this case will have a **.3D_n** extension, where **n** ranges from 0 to **nfiles3D-1**. The input should still be specified as input.3D without considering the **_n** extension.

For large grids, 3D files can be converted to separate files, one field per file, using the 3D2nc_sepfields utility. Its syntax is similar to the aforementioned utilities.

Namelist Variables Controlling 3D Output

nsave3D: Frequency of saving 3D snapshots in time steps.

nsave3Dstart: Initial time step for capturing 3D snapshots. No 3D files will be generated if this exceeds the **nstop** parameter.

nsave3Dend: Final time step for capturing 3D snapshots. No further data will be written for steps exceeding this value.

save3Dbin: If set to .true., data will be saved in binary format.

save3Dsep: If set to .true., each 3D snapshot is saved as a separate file. For 2D simulations, x-z snapshots will consolidate into a single file unless this parameter is set.

nfiles3D: Specifies the number of files used to save a single snapshot. The number of MPI tasks should be divisible by this number.

dogzip3D: When true, the gzip utility compresses output files further. This may considerably slow down large file processing. Default: F

qnsave: Minimum cloud water (kg/kg) threshold for saving 3D fields. This prevents 3D file generation when no cloud water is present in case if one needs 3D fields saved only when clouds are present. Default is 0.

rad3Dout: If true, effective radii for liquid and ice water are also outputted. Default: false.

save3Dnetcdf: When true, outputs are written directly in NetCDF format, requiring the pNetCDF library. Default: F

14.5 Moments Output

These parameters control the output of specialized 3D moment-statistics fields. These fields are averaged onto a coarser grid as determined by the navgmom_x and navgmom_y variables, which are set in domain.f90.

The data will be in the form of field snapshots, with no time averaging. The filenames will include a time step stamp. The data have extension .3D and can be converted to NetCDF using 3D2nc utility.

nstatmom: Frequency of writing moment-statistics snapshots, in steps.

nstatmomstart: The time step at which to begin sampling moment-statistics fields. If this time step surpasses the nstop parameter, no moment-statistics output files will be generated.

nstatmomend: The time step at which to cease saving moment-statistics data. Data saving will stop for time steps exceeding this value.

savemombin: If set to .true., the data will be saved in binary format, as opposed to compressed format.

savemomsep: Determines whether each 3D moment-statistics snapshot is saved as a separate file. This is always the case for 3D model outputs. For 2D runs, however, you can opt to save x-z snapshots to a single file by using this parameter.

14.6 1-Byte Data for Animation (Movies)

The following parameters control saving 1-byte data used for animations (movies). It has been inherited from standard SAM. All movie files for separately for several 2-D (x-y) fields are written into OUT_MOVIES directory. The code can be found in movies.f90. The compression uses the field limits set by namelist MOVIES. Note that each processor writes its own movie file, which then need to be glued together. There is a utility called glue_movie_raw in UTIL directory that can glue together all these files and produces a single RAW format *.raw file for each 2-D field. To see how to use that utility, just execute it without any parameters.

One can convert the *.raw file into animated gif-movie using Image Magick **convert** command like:

```
convert  -depth 8 -size nx_gl x ny_gl file.raw file.gif
```

Here, nx_gl and ny_gl denote the domain sizes in the x and y dimensions. While it's possible to convert these raw files into other video formats, that subject is outside the scope of this document. Various online resources are available for this.

The movie creation is controlled by the following parameters in namelist **PARAMETERS**:

nmovie: frequency of saving (time steps).

nmoviestart: The time step at which the saving of movie data begins. If this time step is larger than the nstop parameter, no movie will be generated.

nmovieend: The time step at which the saving of movie data ends. No data will be appended after this time step.

The movies *.raw files store several fields as compressed 1-byte data. For that, the maximum and minimum values should be defined. The default values are defined in **movies.f90** file. The limits can be also defined using the namelist **MOVIES**:

Namelist MOVIES

u_min, u_max - surface velocity in x

v_min, v_max - surface velocity in x

cldtop_min, cldtop_max - cloud-top temperature

sst_min, sst_max - surface temperature

tasfc_min, tasfc_max - surface air temperature

qvsfc_min, qvsfc_max - surface vapor mixing ratio

prec_min, prec_max - surface precipitation (will convert to log scale)

cwp_min, cwp_max - cloud water path (will convert to log)

iwp_min, iwp_max - ice water path

The resultant files are saved in the OUT_MOVIES directory separately for each MPI task. They can be first glued together using the UTIL/glue_movie_raw utility and subsequently converted to any desired format.

Note that visualization can also be done by using other standard output files. There is no limit in creativity in this regard.

14.7 Satellite Simulators

gSAM inherited three satellite simulators from SAM. The implementation is rather old, and perhapse should be revitalized in the future version. Here are the namelist parameters that control their activation.

doisccp – if .true., ISCCP satellite simulator will be on and a *.isccp file containing the isccp histograms (if dosimfilesout=.true.) will be written in OUT_STAT directory as well as several bulk statistics into the *.stat file. The file can be converted into netcdf format with isccptonc utility found in UTIL.

domodis – if .true., MODIS satellite simulator will be on and a *.modis file containing the isccp histograms (if dosimfilesout=.true.) will be written in OUT_STAT directory as well as several bulk statistics into the *.stat file

domisr – if .true., MISR satellite simulator will be on and a *.misr file containing the isccp histograms (if dosimfilesout=.true.) will be written in OUT_STAT directory as well as several bulk statistics into the *.stat file

dosimfilesout – if .true., the satellite simulators' histograms will be outputted into separate files written in OUT_STAT directory.

References

Bony, S and K. A. Emanuel, 2001: A parameterization of the cloudiness associated with cumulus convection; Evaluation using TOGA COARE data. *J. Atmos. Sci*., 58, No 21, 3158-3183.

Bretherton, C.S., and co-authors (1999), An intercomparison of radiatively driven entrainment and turbulence in a smoke cloud, as simulated by different numerical models. *Q.J.R. Meteorol. Soc.*, 125: 391-423. https://doi.org/10.1002/qj.49712555402

Emanuel, K. A., & Živković-Rothman, M. (1999). Development and evaluation of a convection scheme for use in climate models. *J. Atmos. Sci.*, *56*(11), 1766-1782.

Khairoutdinov, M.F., P. N. Blossey, and C. S. Bretherton, 2022a: Global System for Atmospheric Modeling: Model Description and Preliminary Results. *J. Adv. Model. Earth Syst*,14, e2021MS002968. https://doi.org/10.1029/2021MS002968

Khairoutdinov, M. F., Vogelmann, Andrew M., and Lamer, Katia. 2022b: Simulating Wind Around Isolated Buildings with the System for Atmospheric Modeling. *BNL Technical Report*. https://doi.org/10.2172/1906172.

Khairoutdinov, M. F., and D.A. Randall, 2003: Cloud-resolving modeling of the ARM summer 1997 IOP: Model formulation, results, uncertainties and sensitivities. *J. Atmos. Sci*., 60, 607-625. https://doi.org/10.1175/1520-0469(2003)060<0607:CRMOTA>2.0.CO;2

Kuang, Z., Blossey, P. N., & Bretherton, C. S. (2005). A new approach for 3D cloud-resolving simulations of large-scale atmospheric circulation. *Geoph. Res. Lett.*, *32*(2).

Lee and Khairoutdinov, 2015: Simplified Land Model (SLM) for use in cloud resolving models: Formulation and evaluation. *J. Adv. Model. Earth Syst*., 07, doi: 10.1002/2014MS000419.

Morrison, H., Curry, J. A., & Khvorostyanov, V. I. (2005). A new double-moment microphysics parameterization for application in cloud and climate models. Part I: Description. *J. Atmos. Sci*., 62(6), 1665–1677.

Morrison, H., & Milbrandt, J. A. (2015). Parameterization of cloud microphysics based on the prediction of bulk ice particle properties. Part I: Scheme description and idealized tests. *J. Atmos. Sci*., 72(1), 287–311.

Smolarkiewicz, P.K., 2006: Multidimensional positive definite advection transport algorithm: an overview, *Int. J. Numer. Methods Fluids*. *50*, 1123–1144.

Stevens, B., and Coauthors, 2005: Evaluation of Large-Eddy Simulations via Observations of Nocturnal Marine Stratocumulus. Mon. Wea. Rev., 133, 1443–1462, https://doi.org/10.1175/MWR2930.1.

Thompson, G., P. R. Field, R. M. Rasmussen, and W. D. Hall, 2008: Explicit Forecasts of winter precipitation using an improved bulk microphysics scheme. Part II: Implementation of a new snow parameterization. *Mon. Wea. Rev*., 136, 5095-5115.

Yamaguchi, T., D. A. Randall, and M. F. Khairoutdinov, 2011: Cloud Modeling Tests of the ULTIMATE-MACHO Scalar Advection Scheme. Monthly Weather Review, 139, pp.3248-3264