Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions
Open Access
EGU

`sympl` (v. 0.3.2) and `climt` (v. 0.11.0) – Towards a flexible framework for building model hierarchies in Python

Joy Merwin Monteiro[1], Jeremy McGibbon[2], and Rodrigo Caballero[1]

[1]Department of Meteorology, Stockholm University, SE-106 91 Stockholm, Sweden
[2] 408 Atmospheric Sciences–Geophysics (ATG) Building Box 351640, Seattle, Washington 98195-1640

Correspondence: Joy Merwin Monteiro (joy.merwin@gmail.com)

Abstract. `sympl` (System for Modelling Planets) and `climt` (Climate Modelling and diagnostics Toolkit) represent an attempt to rethink climate modelling frameworks from the ground up. The aim is to use expressive data structures available in the scientific Python ecosystem along with best practices in software design to build models that are self-documenting, highly inter-operable and that provide fine grained control over model components and behaviour. We believe that such an approach towards building models is essential to allow scientists to easily and reliably combine model components to represent the climate system at a desired level of complexity, and to enable users to fully understand what the model is doing.

`sympl` is a framework which formulates the model in terms of a "state" which gets evolved forward in time by `TimeStepper` and Implicit components, and which can be modified by `Diagnostic` components. `TimeStepper` components in turn rely on `Prognostic` components to compute tendencies. Components contain all the information about the kinds of inputs they expect and outputs that they provide. Components can be used interchangeably, even when they rely on different units or array configurations. `sympl` provides basic functions and objects which could be used by any type of Earth system model.

`climt` is an Earth system modelling toolkit that contains scientific components built over the `sympl` base objects. Components can be written in any language accessible from Python, and Fortran/C libraries are accessed via Cython. `climt` aims to provide different user APIs which trade-off simplicity of use against flexibility of model building, thus appealing to a wide audience.

Model building, configuration and execution is through a Python script (or Jupyter Notebook), enabling researchers to build an end-to-end Python based pipeline along with popular Python based data analysis tools. Because of the modularity of the individual components, using online data analysis, visualisation or assimilation algorithms and tools with `sympl`/`climt` components is extremely simple.

1 Introduction

The climate is a complex system constituted of a heterogeneous set of mutually interacting subsystems (atmosphere, ocean, cryosphere, biosphere, chemosphere) each encompassing a broad range of physical and chemical processes with space and time scales spanning many orders of magnitude. Modelling and understanding the climate system in its entirety is truly a grand scientific and technological challenge. An increasingly recognised strategy for tackling this challenge is to build a hierarchy

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

of models of varying complexity. Simpler models are more amenable to in-depth analysis and understanding; the insight gained from these simpler models can then be used to understand slightly more complicated models, and so on (Held, 2005). Specifying which particular models should occupy each rung in such a hierarchy is necessarily a matter of subjective choice, and the questions of how to create a hierarchy and what models are desirable in a canonical hierarchy has generated extensive discussions (Jeevanjee et al., 2017). Our purpose here is to present a modelling framework which enables climate scientists to easily and transparently traverse the modelling hierarchy.

Designing and building a framework that facilitates traversing this hierarchy is non-trivial and continues to remain a challenge. The lack of flexibility of existing climate models forces scientists to spend a lot of time reading and modifying code to construct alternative model versions that should in principle be straightforward to build. Most modelling frameworks simply provide code to exchange information between different physical domains such as atmosphere and ocean (See Theurich et al., 2015; Valcke et al., 2012, for a survey of modelling frameworks), with each physical domain being represented by a monolithic block of code. It was only with the advent a new generation of frameworks like the Earth System Modelling Framework (ESMF) (DeLuca et al., 2012; Theurich et al., 2015) that fine-grained control over the components that constitute a climate model was made possible. However, the norm continues to be that climate models are configured by namelist variables and boolean flags in the code rather than framework based approaches (like the component trees that ESMF allows). `sympl` and `climt`, on the other hand, allow finer grained control over the composition of the model, with individual components representing physical processes (such as radiation, convection, etc.,) rather than physical domains. Attempting to model the climate system at this fine-grained level has its own challenges, which we attempt to solve in these packages. Initiatives to build frameworks to traverse the hierarchy between highly idealised models to full scale GCMs (general circulation models) do exist (Fraedrich et al., 2005; Vallis et al., 2017), but we believe the fine-grained configuration allowed by our design to be unique.

Another emerging concern in the scientific community is that of reproducibility of research (Peng, 2011). While publicly available climate models do provide validated configurations that are in principle completely reproducible, climate scientists routinely need to make changes to the code that are hard to document (or understand), as mentioned above. Short of sharing a copy of the entire code base, such modifications makes it difficult to reproduce simulations. While some level of code manipulation is inescapable, we note from our own experience and from reading about such modifications in the literature that most of them follow set patterns which could easily be provided by modelling frameworks themselves.

`sympl` and `climt` were written to address some of these issues, and to create a modelling framework that is easy to use and facilitates reproducibility of simulations. Both these packages are subject to ongoing development, but have reached a level of maturity that makes it worthwhile to document them here. In the following sections, we describe some design issues that modelling frameworks have to solve, followed by the design of `sympl` and `climt`, some example scripts and benchmark calculations, and conclude with a discussion of developments planned in the future.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

2 Climate modelling frameworks – general considerations

For the purposes of this paper, we define a modelling framework as a library or libraries that allow the creation of climate models by providing abstractions of "infrastructure" code. This infrastructure could include model-independent parts: data, time and units management, component description and interoperability as well as model dependent parts: component configuration, model state and grid initialisation and model creation from components.

Climate modelling frameworks should make it possible to combine arbitrary model components in an intuitive manner to create a model of complexity appropriate to the scientific question at hand – ideally, the user simply has to specify the components desired, and the framework should be able to automatically build the model. The user should also be able to specify details such as the order in which components are called and the time stepping schemes used. In addition, it is also desirable that the process of creating a model is fairly easy to understand, and that the model code be self-documenting to eliminate the need to write additional documentation whenever possible. To achieve these goals, the following challenges must be addressed by any modelling framework.

2.1 Taxonomy of model components

Typically, the integration of scientific code (or model components) and the modelling framework happens at the physical domain level – atmosphere, ocean, land and so on. The processes the operate within each physical domain (fluid dynamics, radiation, convection etc.,) are not accessible in a systematic manner, and their integration into the model code is harder to unravel. For example, changing the radiation code in an atmosphere model is typically harder than changing the kind of ocean the atmosphere is coupled to (slab, dynamical, etc.,). This tight integration at the process level can make these models highly efficient but less flexible to work with. Furthermore, the fact that certain process level components are written to work only with certain other components demands an "architectural unity" (Randall, 1996) which might also encourage tight integration of model components.

If we allow for configuration at the process level, we are then faced with model components which behave quite differently: Some components (like radiation) return tendencies, while others (like large scale condensation) return a new value of a physical quantity. Any modelling framework must therefore have an ontology of components that is sufficient to capture this diversity, especially if the goal is to automate model construction.

2.2 Behaviour of model components

Model components can display a variety of behaviours which must be exposed in a consistent manner by the modelling framework. For instance, model components could

- Normally return a new value rather than a tendency of some physical quantity, but in certain instances a tendency might be desirable.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

- Provide output that is piece-wise constant in time – the output is updated only once every N iterations, and the same value it output for the next $N-1$ iterations. This behaviour is normally used in radiative transfer codes.
- Scale some of its inputs or outputs by some floating point number. This kind of behaviour is desirable for example in mechanism denial studies.

We can modify certain behaviours of model components, such as the ones above, by interacting with only the inputs and outputs of scientific components. Such modifications can be applied in the same manner to different components.

3 Configuration of models

Models need to be configured at multiple levels, and modelling frameworks have to provide a natural way of configuring the following aspects of a model:

- Physical configuration: the physical constants required by the model components.
- Algorithmic configuration: the "tunable" parameters which modify the behaviour of the algorithm which represent physical processes – like the entrainment coefficient in a convective parameterisation, for example.
- Behavioural configuration: described in the previous section.
- Memory and Computing resource configuration: The layout of arrays used by the model and the distribution of the model components over the available number of processors and co-processors.
- Compositional configuration: The components that compose a model, any dependency between components, the order in which components are executed all need to be quantified. While choosing the order of components, it must be kept in mind that the ordering of components can have an impact on the simulated climate of the model (Donahue and Caldwell, 2018).

3.1 Interacting with model components

It would be desirable to be able to interact with individual components at runtime – Not only would this serve an important educational purpose, but it would also facilitate diagnostic calculations made very often during research. For instance, it would be very helpful if one could calculate radiative tendencies without having to build a full model to access the radiative transfer component.

3.2 Combining model components

Different model components (especially those from different physical domains) can use different discretisations of the domain, or model grids. Combining components which use different grids involves interpolating from one grid to another in the case one of these components requires information provided by other components. This functionality is provided by a separate

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

component known as the coupler (Valcke et al., 2012). It would be desirable that the modelling framework is able to identify that the constituent components of a model are on different grids and provide support for adding couplers when necessary.

3.3 Models and computing infrastructure

Models need to run on a variety of computing facilities, and modelling frameworks need to abstract away the details of the computing infrastructure to enable model users to focus on the scientific aspects of the simulations. Creating such an abstraction is challenging due to the introduction of new architectures and computing models such as Graphics Processing Units (GPUs) and many-core CPUs (See introduction in Balaji et al., 2016). The modelling framework must facilitate storage of information about the computational resources available so that components that can take advantage of certain specialised hardware will be able to do so.

Figure 1 shows the wide variety of configuration information and the places where such configuration lies in climate models. The sheer variety of locations where the configuration resides makes it hard to keep track of what configuration information has changed and makes it daunting for beginners to setup models. In contrast, model configuration using `sympl` and `climt` is highly centralised and easily accessible. Such centralised configuration also reduces errors arising due to misconfiguration of the model.

4 Design Decisions

During the development of the framework described in this paper, it was realised that some aspects of model development were fully model agnostic, whereas some aspects such as model state initialisation, ordering of coordinates axes and labelling conventions were best decided by each model. The model agnostic parts were included in the framework package (`sympl`) whereas the model specific parts were included in the model package (`climt`).

Python was used as the language to write the framework. Python as a language and the Python ecosystem have a number of desirable features, all of which were taken advantage of during the development of `sympl` and `climt`:

- Earlier versions of `climt` were written in Python, which gave the authors an idea of the convenience and flexibility it afforded. In particular, the object orientation capabilities of Python provide a straightforward way to represent the component based architecture adopted by almost all climate modelling frameworks.
- Scientific libraries within the Python ecosystem now offer acceptable performance for computationally intensive operations typically used in climate models.
- The Python ecosystem includes a wide variety of libraries that might be useful for climate models. Examples include machine learning, graphics and web services libraries.
- Python's ability to act as a glue language allows interfacing with the large number of libraries for climate modelling already available in Fortran.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

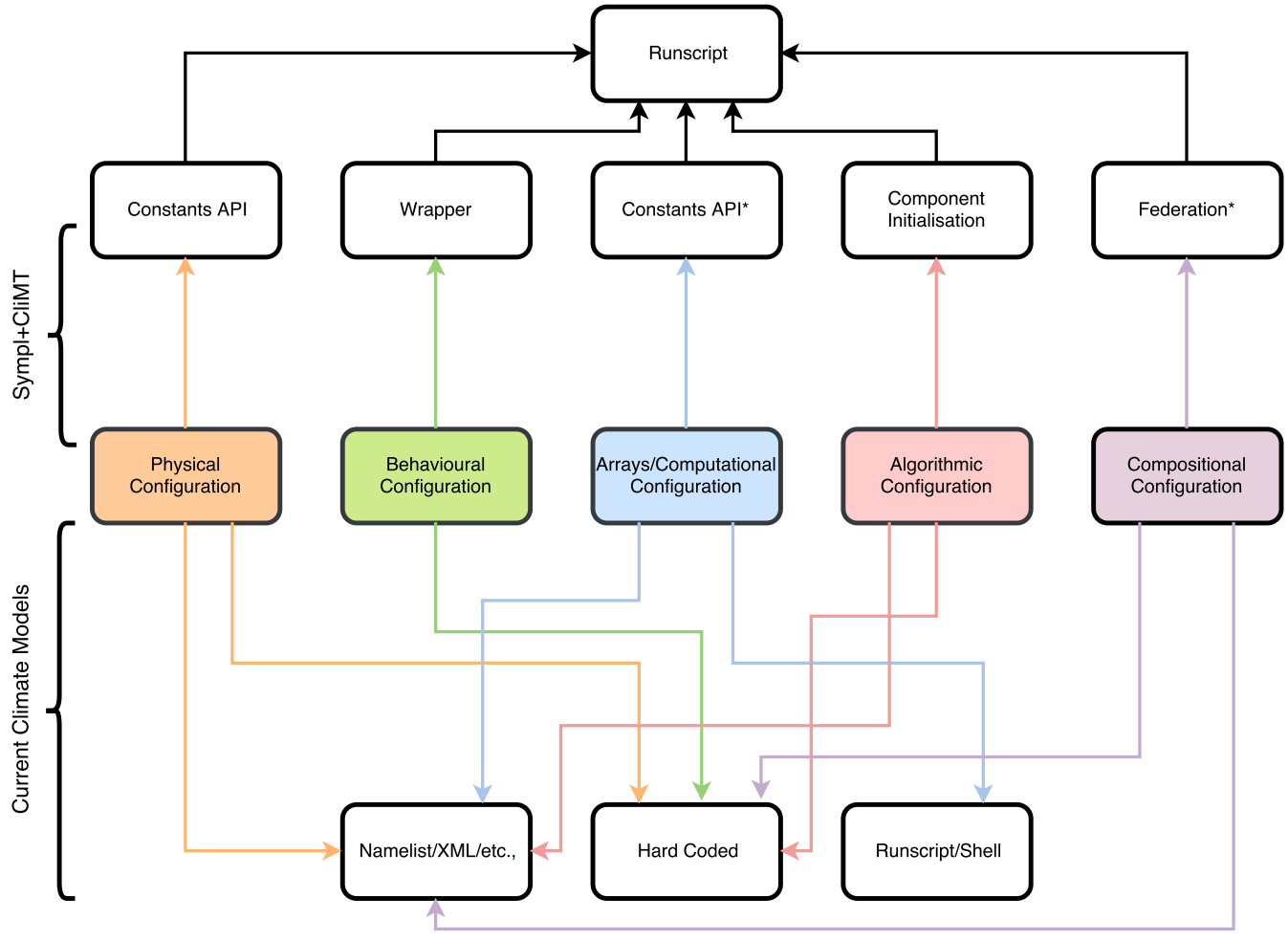

Figure 1. The variety of configuration options in a climate model and in the `sympl-climt` framework. Note that not every climate model uses all configuration options. The starred boxes in the `sympl+climt` side indicate functionality that is not yet implemented.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions

– Tools available in the Python ecosystem like `jupyter`, `pytest` and `sphinx` which enable writing reproducible workflows and code that is well documented and tested.

Model arrays are contained within the `DataArray` abstraction provided by `xarray`[1] rather than the more low-level `numpy` array. `DataArrays` are an abstraction over `numpy` arrays with a more natural fit to climate data by providing labeled dimensions and metadata storage capabilities. `sympl`'s `DataArray` object is a subclass of the `xarray DataArray` that provides units handling and conversion and will be described subsequently. A fortunate side-effect of this design choice is that the analysis capabilities built into `xarray` are automatically available, allowing users to build an end-to-end pipeline entirely within Python, from simulation to data analysis and generation of publication-ready figures. Since low-level array operations using `numpy` and `xarray` are fairly simple, especially changing coordinate ordering and C/Fortran memory ordering, `climt` only provides guidelines for memory layout of arrays. However, changing the ordering of dimensions in memory will incur a performance penalty if the model uses any components which wrap Fortran libraries.

In the interest of readability of component and model code, we strongly encourage using descriptive names for model quantities, preferably adhering to the CF conventions[2]. Most pre-defined quantities in `climt` use names derived from the CF conventions. One additional suffix that we found necessary to use was `on_interface_levels` to distinguish between quantities defined on the interfaces and mid-levels of the vertical grid. For example `air_temperature` refers to the air temperature defined at the vertical grid centre, whereas `air_temperature_on_interface_levels` refers to the same quantity defined at the vertical grid edges.

5 `sympl` – Design and programming interface

`sympl` conceives of a climate model as a state that is continuously updated by various components. `sympl`'s taxonomy consists of seven kinds of components. Four of these component types are used to represent physical processes and the remaining represent other functionality required to build and run models:

– `Prognostic` components which take the model state as input and returns tendencies of certain quantities and optional diagnostics as output.

– `Implicit` components which take the model state and a timestep as input and returns a new values of certain quantities and optional diagnostics as output.

– `Diagnostic` components which take the model state as input and return diagnostics as output.

– `ImplicitPrognostic` components which return tendencies, but require the model timestep to produce these tendencies. This requirement may be to implement flux limiting or to ensure that the tendencies satisfy the CFL criterion. This kind of behaviour is mainly used in convection schemes.

[1] see xarray documentation at http://xarray.pydata.org/en/stable/

[2] http://cfconventions.org/Data/cf-standard-names/48/build/cf-standard-name-table.html

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions

- `TimeStepper` components which contain a set of `Prognostic` components and use the tendencies they output to integrate the model state forward in time.
- `Monitor` components which provide a `store` method which takes the model state as input and "stores" it. The implementation of this method is left to the user, and currently is used to implement monitors for NetCDF output and plotting.
- `Wrapper` components are a special case of other component types which contain a "wrapped" component used for computation and modify its behaviour as described in Sec. 2.2. Currently, `sympl` has the following wrappers:
  - `TimeDifferencingWrapper` creates tendencies from the output of an `Implicit` component by first order differencing.
  - `UpdateFrequencyWrapper` calls the wrapped `Prognostic` only after the user-specified time interval has elapsed, and simply outputs this value until the next time. In effect, it creates a piecewise constant output tendency which can reduce the computational load during a simulation.
  - `ScalingWrapper` scales the inputs before passing it onto the wrapped component and the outputs (new state, tendency or diagnostic) returned by the component.
  - `TendencyInDiagnosticsWrapper` returns all tendencies generated by a component as part of its diagnostics.

This ontology is larger than the ontology typically used in modelling frameworks. For example, ESMF only considers two kinds of components – Gridded and Coupler components. However, as discussed previously, this enlarged ontology is required to capture the diversity of components that arise if configuration of models is done at the process level.

5.1 Model State and the `DataArray` abstraction

The model state is a dictionary whose keys are the names of model quantities and values are `sympl DataArray` objects. The model state also contains a required keyword `time` whose value is an object that implements the Python `datetime` or `timedelta` interface. `sympl` provides an interface to use the `datetime` objects from the `cftime`[3] package to support several different calendars. A schematic of the model state is presented in Fig. 2. `sympl` does not put hard restrictions on the name of model quantities, though standardised names should be used to ensure inter-package compatibility. All `DataArray` objects must define a string attribute called `units`, which is used to convert the data contained within to the appropriate units requested by a component. The units conversion is performed using the `Pint`[4] library. Since the actual contents of the state is dependent on model details, `sympl` assumes that the initialisation of the model state will be done by a model package, or by the user.

[3] https://github.com/Unidata/cftime
[4] https://pint.readthedocs.io/en/latest/

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions

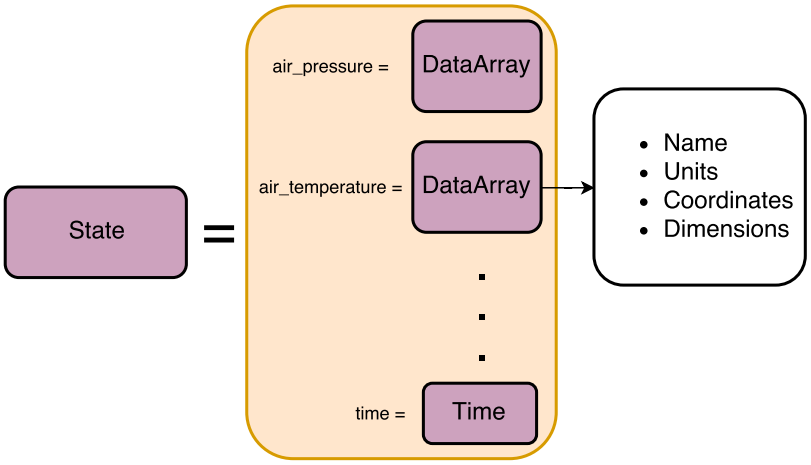

Figure 2. The model state as in the `sympl` framework. The state in the orange box contains all the information that is stepped forward in time. Each `DataArray` contains information such as the quantity name, units and dimensions/coordinates.

5.2 Model dimensions

`sympl` stores the names of spatial dimensions required by model components, but treats model coordinates (e.g. latitude) as any other state quantity. Each of the dimensions `x, y, z` can have arbitrary names and new names can be added – for instance, `climt` adds two names for the vertical dimension, `mid_levels` and `interface_levels`. This facility can be used, for instance to define different names for dimensions used in land, ocean and atmosphere models, for instance.

5.3 Physical Constants

`sympl` maintains a unit-aware library of constants which can be accessed or modified by model packages and by the user through `get_constant()` and `set_constant()` functions. For example, `planetary_rotation_rate` can be changed to perform sensitivity experiments. The unit handling is important to ensure constants are given to components in the units required. For example, the RRTMG radiative transfer code (Clough et al., 2005), requires physical constants in CGS units.

For the purposes of logging, physical constants are classified into various categories:

- Planetary constants like rotation rate, acceleration due to gravity.
- physical constants like the speed of light.
- atmospheric constants like specific heat of dry air and reference air pressure.
- stellar constants like stellar irradiance

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions

```
class FictionalComponent(Prognostic):

    input_properties = {
        'air_temperature' = {
            'dims' = ['latitude', 'longitude', 'mid_levels'
             'units' = 'degK',
        }
    }

    tendency_properties = {
        'air_pressure' = {
            'dims_like' = 'air_temperature',
            'units' = 'mbar/s',
        }
    }

    def __init__(self, *kwargs):
        ...

    def __call__(self, state):
        ...
```

Figure 3. The general code layout for a `sympl` component.

- condensible constants which refer to the thermodynamic properties of the condensible (in all three phases) in the atmosphere.
- Oceanographic constants such as the reference sea water density.

We chose to keep the constants related to the condensible component of the atmosphere separate to ensure `sympl` is flexible enough to handle general planetary atmospheres. `sympl` provides a function `set_condensible()` which allows switching all constants related to the condensible. For example `set_condensible('methane')` will replace all condensible constants (such as density of liquid/solid/gaseous phases, latent heat of condensation) to those corresponding to methane, provided such constants are already in the constants dictionary. The default condensible is water, which is currently the only condensible compound for which default values are given.

5.4 Anatomy of a `sympl` component

A `sympl` component is described in Fig. 3. `input_properties` and `tendency_properties` are examples of property attributes that contain details of the required inputs and returned outputs, tendencies or diagnostics. Here, the input is a quantity

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions

called `'air_temperature'` whose horizontal dimensions are `latitude`, `longitude` and whose values in the vertical are defined at model mid-levels. The units are specified for each quantity.

Dimension and unit requirements in `sympl` are not restrictions on the inputs, but rather describe the internal representation used by the component. `sympl` provides helper functions that will automatically convert the input state to satisfy these requirements, and raise an exception if that is not possible.

Since this component is a `Prognostic`, it outputs tendencies of a quantity called `air_pressure` whose dimensions are the same as that of the input quantity. The `__call__()` method accepts the state dictionary as input and returns tendencies as specified in the component properties. If the component was an `Implicit`, then the above method would also require the timestep as an argument to step the quantities forward in time.

5.5 Modelling using `sympl`

A typical workflow when using a model written using `sympl` might involve the following steps:

1. Initialise model components, providing configuration information.
2. Use `Wrapper` components to modify the behaviour of any components if necessary.
3. Intialise model state which contains all quantities required by the selected components.
4. Use `TimeStepper` to collect all `Prognostic` components into a component that can be stepped forward in time.
5. Call `Diagnostic` to compute any derived quantities from prognostic quantities or provide forcing quantities at a given time step.
6. Call `Implicit` components and get a new state dictionary with the updated model quantities and any diagnostics. Update model state with new values and diagnostics.
7. Call `TimeStepper` and get a new state dictionary with the updated model quantities and any diagnostics. Update model state.
8. Call any `Monitor` components to store model state[5].
9. Increment model time.

A schematic version of the data flow in a model run is presented in Fig. 4. The `TimeStepper` provides the same model state to all `Prognostics` it contains and sums the tendencies before stepping forward in time. Using other time marching algorithms such as sequential tendency or sequential update splitting (Donahue and Caldwell, 2018) will require the user to implement their own `TimeStepper` subclass. All components are called in serial order, though there is no design restriction which forbids calling the components in parallel at each time step.

[5]Note that "store" could mean to store to disk, display in real time, send over the network, or anything else.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

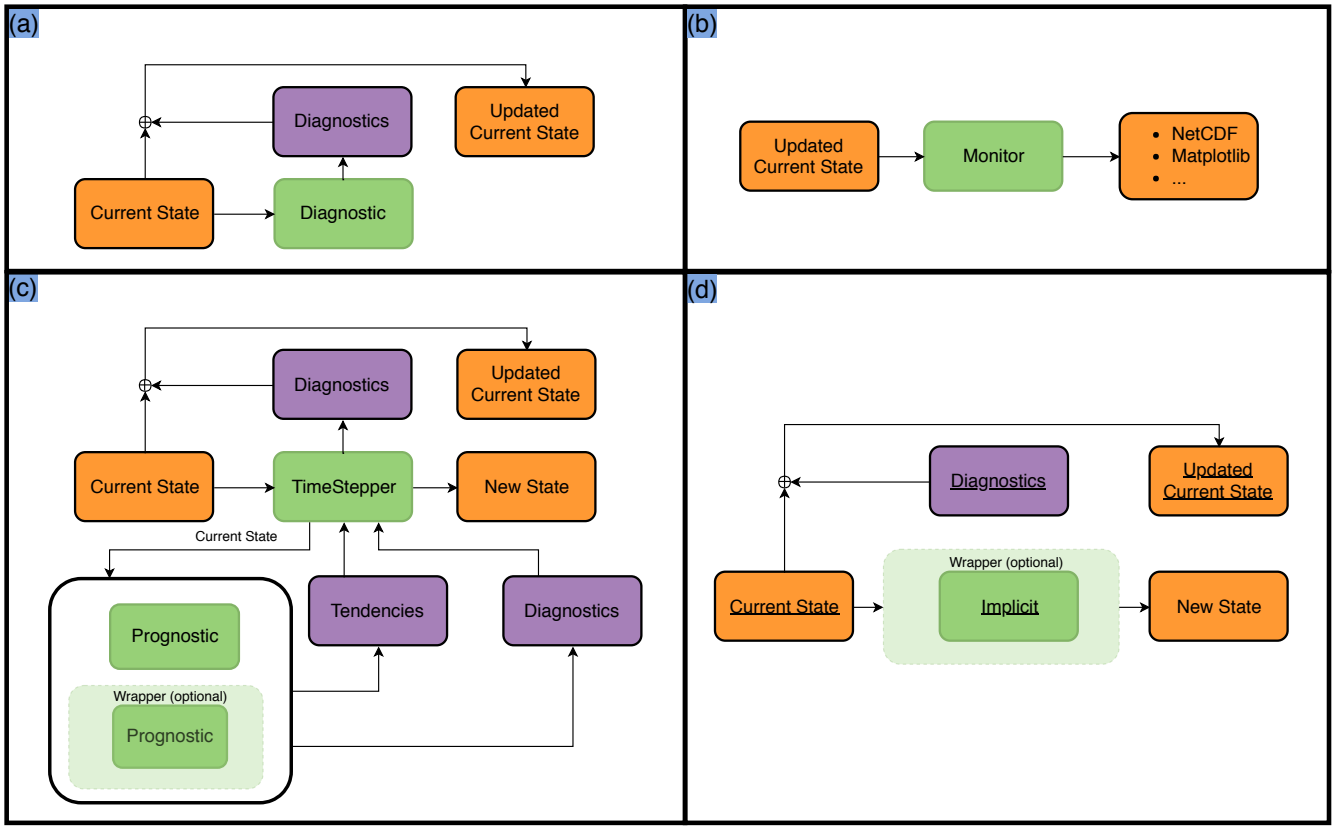

Figure 4. Flow of data for each type of component in a typical `sympl+climt` modelling run. The four panels show **a**) `Diagnostic` components **b**) `Monitor` components **c**) `TimeStepper` with `Prognostic` components **d**) `Implicit` components. The dark green boxes denote components, light green boxes denote optional `Wrappers`, the orange boxes indicate the state dictionary at different times and the purple boxes indicate tendencies and diagnostics generated by components.

6 `climt` – Design and programming interface

While `sympl` focuses on providing a programming model, a rich ontology of components and model agnostic configuration options, `climt` focuses on adding the actual scientific components, model dependent configuration options and some helper functions to reduce boilerplate code while writing components. `climt` also adds additional attributes to existing components required to initialise model state, which is model dependent and not handled by `sympl`.

`climt` also introduces the `ClimtSpectralDynamicalCore` component which is a subclass of `TimeStepper` and is used to represent spectral dynamical cores. Spectral dynamical cores typically step the model forward in spectral space, and therefore require tendencies in spectral space.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

6.1 Model state, quantity dimensions and output dictionaries

`climt` provides a function `get_default_state()` which accepts a list of components (of any kind) and optional coordinate values for the four basic dimensions (defined as `x`, `y`, `interface_levels` and `mid_levels`) and creates a state dictionary. The shape of the arrays in the resulting state dictionary is determined by the values of these coordinates. In addition to dimensions and units, `climt` also requires component properties to specify default values for each model quantity. A dictionary is maintained internally in `climt` for the most commonly used model quantities[6]. The default values for these internally defined quantities are scientifically meaningful, and can be used without modification for certain simulations.

Certain complications arise when creating the model state, and these are handled by `climt` in the following manner:

- Certain model quantities may have dimensions which do not correspond to any spatial dimension. For instance, correlated-k radiative transfer codes require arrays with an additional dimension that corresponds to the number of bands used to discretise the electromagnetic spectrum. `climt` allows components to specify these dimensions and their coordinate values by allowing an optional attribute called `extra_dimensions`.

- Certain model quantities may already have a description in the internal `climt` dictionary, but a component may want to describe it differently for algorithmic purposes. To allow for this use case, `climt` allows components to redefine the dimensions, units, and other attributes of a model quantity by defining an optional attribute `quantity_descriptions`.

For components with a large number of outputs, tendencies or diagnostics, creating the output dictionaries can consume many lines of code. `climt` components define a method `create_state_dict_for()` to reduce this boilerplate and keep component code readable.

6.2 Model Composition

Currently, the creation of the model and running the simulation loop is done by hand, which provides better understanding of what the model is doing but in many cases is standardised and repetitive. In the near future, `climt` will provide an additional class called `Federation` which automates the process of creating a model from its components. As mentioned before, this automation is possible only because of the rich taxonomy of components `sympl` provides.

7 Features and software engineering

`climt` currently has the following components that can be used to build models:

- RRTMG longwave and shortwave radiative transfer (Clough et al., 2005): This is a Fortran component accessed via a cython wrapper. RRTMG is a state-of-the-art radiative transfer code used in many climate models.

- Grey radiation scheme: along with a `Diagnostic` component that provides an optical depth distribution which mimics the effect of water vapour (Frierson et al., 2006). These components are written in pure Python. This radiative scheme

[6] As of this writing, these quantities are mainly from the atmosphere and land domains.

has been used in many idealised climate dynamics simulations to isolate the thermodynamic effects of latent heat release from the radiative effects of water vapour, which is a strong greenhouse gas.

- Insolation: This component is written in pure Python. It provides the solar zenith angle based on the time available in the model state. This zenith angle is used in radiative transfer codes. Currently, this component uses approximations and orbital parameters which make it highly accurate for earth but inapplicable to other planets.

- Emanuel convection scheme (Emanuel and Živković Rothman, 1999): This is a Fortran component accessed via a cython wrapper. It is a mass flux based convection scheme which is based on the boundary layer quasi-equilibrium hypothesis (Raymond, 1995).

- Grid scale condensation. This is written in pure Python. It calculates the water vapour and temperature fields in the atmosphere after condensing out excess water vapour to keep the atmospheric column from becoming super-saturated.

- Spectral dynamical core, derived from the General Forecast System (https://github.com/jswhit/gfs-dycore). This is a Fortran module accessed via a cython wrapper. It uses a high performance spherical harmonics library `shtns` (https://bitbucket.org/nschaeff/shtns). It is parallelised using OpenMP, and therefore is most effective on shared memory systems. The dynamics is stepped using an implicit-explicit total variation diminishing Runge-Kutta 3 time-stepper. The physics tendencies are stepped forward using a forward Euler scheme.

- Simple Physics package for idealised simulations (Reed and Jablonowski, 2012). This is a fortran module accessed via a cython wrapper. It provides initial conditions which can be used for testing moist dynamical cores, and also provides a simple diffusive boundary layer suitable for idealised simulations.

- Slab surface. This component is written in pure Python. It allows for a prognostic surface temperature by calculating the surface energy budget. It is flexible enough to represent land or ocean. It currently does not account for localised heat fluxes.

- Sea/land ice model which allows for snow and ice layers, and energy balanced top and bottom surfaces. This component is written in pure Python. It is flexible enough to represent ice/snow growth and melting. It is capable of representing sea or land ice based on the surface type available in the model state. It currently cannot handle fractional land surface types.

- Held-Suarez forcing (Held and Suarez, 1994). This component is written in pure Python. It provides an idealised set of model physics which can be used for testing dry dynamical cores and idealised simulations.

- Initial conditions from the dynamical core MIP (DCMIP). This is a Fortran module accessed via a cython wrapper. It provides initial conditions for a wide variety of tests which allow assessing the conservation properties of dynamical cores.

This set of components allow building a hierarchy of models ranging from single column radiative-convective models to energy balanced moist atmospheric general circulation models. Because of the fine-grained configurability of `sympl`/`climt`,

the difference between the number of lines of code required to build a single column model and a moist GCM is only around 40 lines of Python code; More importantly, most of the code is reusable when moving from a simpler to a more complex model.

Both `sympl` and `climt` are open source projects, licensed under a permissive BSD license. Both packages are available on Mac, Linux and Windows platforms, and can be directly installed from the Python Package Index using one line commands:

```
pip install sympl
pip install climt
```

eliminating the need to download source code from GitHub. The Python Package Index projects are located at https://pypi.python.org/pypi/sympl and https://pypi.python.org/pypi/climt respectively.

`climt` also provides binary releases on all supported platforms, eliminating the need to have a compiler on the user's system. `sympl` is written in pure Python, and does not have any compiler requirements. Both packages are regression tested using the online services TravisCI (https://travis-ci.org/) and AppVeyor (https://www.appveyor.com/). Both packages also maintain regularly updated documentation at http://sympl.readthedocs.io/en/latest/ and http://climt.readthedocs.io/en/latest/.

8 Example Script

Figure 5 shows a typical script used to build a model using `sympl` and `climt`. The first few lines simply import components that are required to build the model. These imports also serve to inform the user which components are required to build the required model.

Lines 8-20 initialise the components and the model timestep. This includes a variety of components, including `Monitor`, `Prognostic` and `Implicit` components. Algorithmic configuration information is passed to the components as keyword arguments while initialising them.

Lines 22-31 are concerned with creating a model state and initialising suitable values. This initialisation is dependent on the scientific question at hand.

A TimeStepper is created on Line 33 which combines the tendencies from the various `Prognostics` and will eventually produce new values of state quantities.

Lines 37-49 are concerned with running the model itself. The state updated by the `TimeStepper` is used as input to the `simple_physics` component which generates a new state. The diagnostics generated during these calculations are collected in the old state variable which can then be stored. Finally, the state variable is updated and the model timestep is incremented.

Changing the simulation to something other than a single-column model requires that information about the spatial dimensions be passed to the `get_default_state` function. Therefore, converting this script to a GCM will only require initialising the dynamical core component (which acts as the time stepper) and specifying the appropriate dimensional information while creating model state.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

Geoscientific Model Development Discussions
Open Access
EGU

```python
from sympl import (
    AdamsBashforth, NetCDFMonitor)
from datetime import timedelta
from climt import (
    SimplePhysics, get_default_state,
    EmanuelConvection, RRTMGShortwave, RRTMGLongwave, SlabSurface)

# Create output Monitor
netcdf_monitor = NetCDFMonitor('test_sw.nc', write_on_store=True)

# Set timestep using timedelta
timestep = timedelta(minutes=5)

# Initialise Components
convection = EmanuelConvection()
radiation_sw = RRTMGShortwave()
radiation_lw = RRTMGLongwave()
slab = SlabSurface()
simple_physics = SimplePhysics()

# Create model state
state = get_default_state([simple_physics, convection,
                           radiation_lw, radiation_sw, slab])

# Set initial values and other parameters
state['air_temperature'].values[:] = 270
state['zenith_angle'].values[:] = np.pi/4
state['surface_temperature'].values[:] = 300.
state['ocean_mixed_layer_thickness'].values[:] = 50
state['area_type'].values[:] = 'sea'

# Create TimeStepper using all Prognostics
time_stepper = AdamsBashforth([convection, radiation_lw,
                               radiation_sw, slab])

# Run model
for i in range(60000):
    # First run TimeStepper
    diagnostics, state = time_stepper(state, timestep)
    state.update(diagnostics)
    # Then run Implicit components. This order is arbitrary
    diagnostics, new_state = simple_physics(state, timestep)
    state.update(diagnostics)
    if i % 20 == 0:
        # Store data at regular intervals
        netcdf_monitor.store(state)
    # Update model state
    state.update(new_state)
    state['time'] += timestep
```

Figure 5. Example script for a single-column radiative-convective model. Some code was omitted to enhance the presentation.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

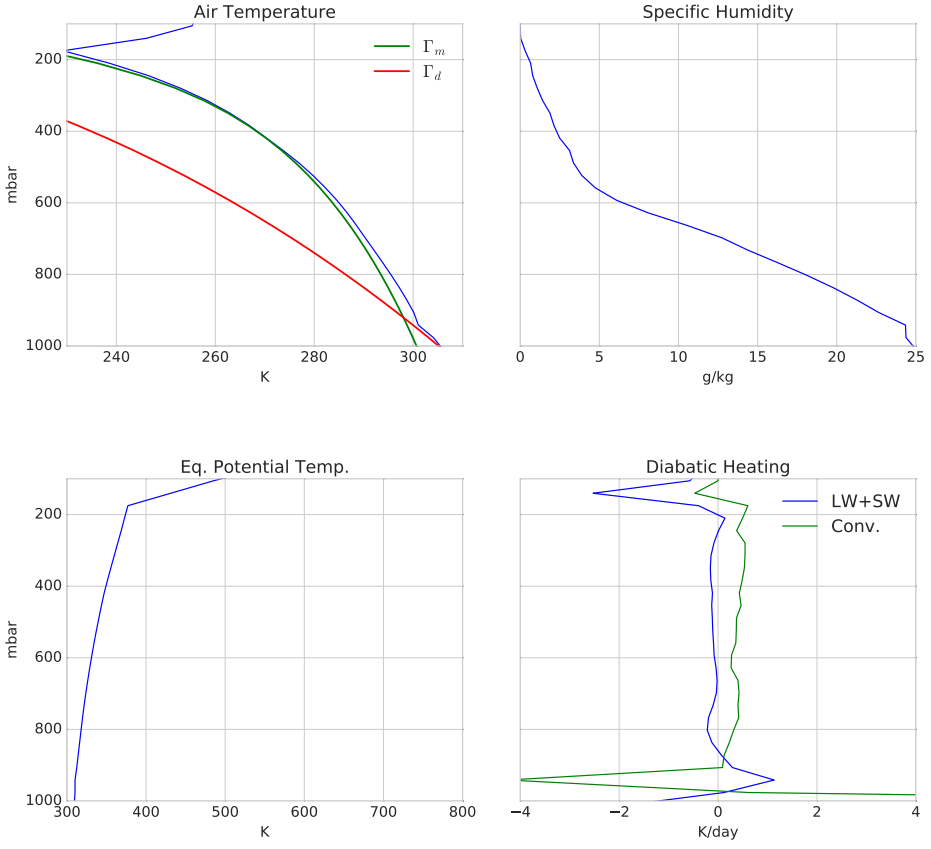

Figure 6. The mean equilibrium profiles in the radiative convective single column model. In the top left panel, the dry and moist adiabats are plotted in red and green respectively.

9 Some benchmark simulations

The first simulation is that of an atmospheric column that is run to equilibrium in the presence of radiation and convection. This model uses the RRTMG longwave and shortwave components, the Emanuel convection scheme, the Simple Physics component as its boundary layer scheme and a slab ocean of thickness 50 meters at the surface. The model timestep is 5 minutes and the results are the mean between 8000 and 10000 timesteps. The results are presented in Fig. 6. The air temperature transitions from a dry adiabat in the boundary layer to a moist adiabat in the free atmosphere until the tropopause. Correspondingly, the equivalent potential temperature increases slowly until the tropopause and rapidly thereafter.

The second simulation is of a idealised aquaplanet GCM with fixed equinoctial insolation. As mentioned before, the modular nature of our framework allows re-use of much of the runscript code from the above single column model. It consists of all the components used in the previous model along with a dynamical core which is used as the time stepper. The simulated climate of the model is as expected from such a configuration: the zonal mean zonal winds show two strong westerly jets which

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

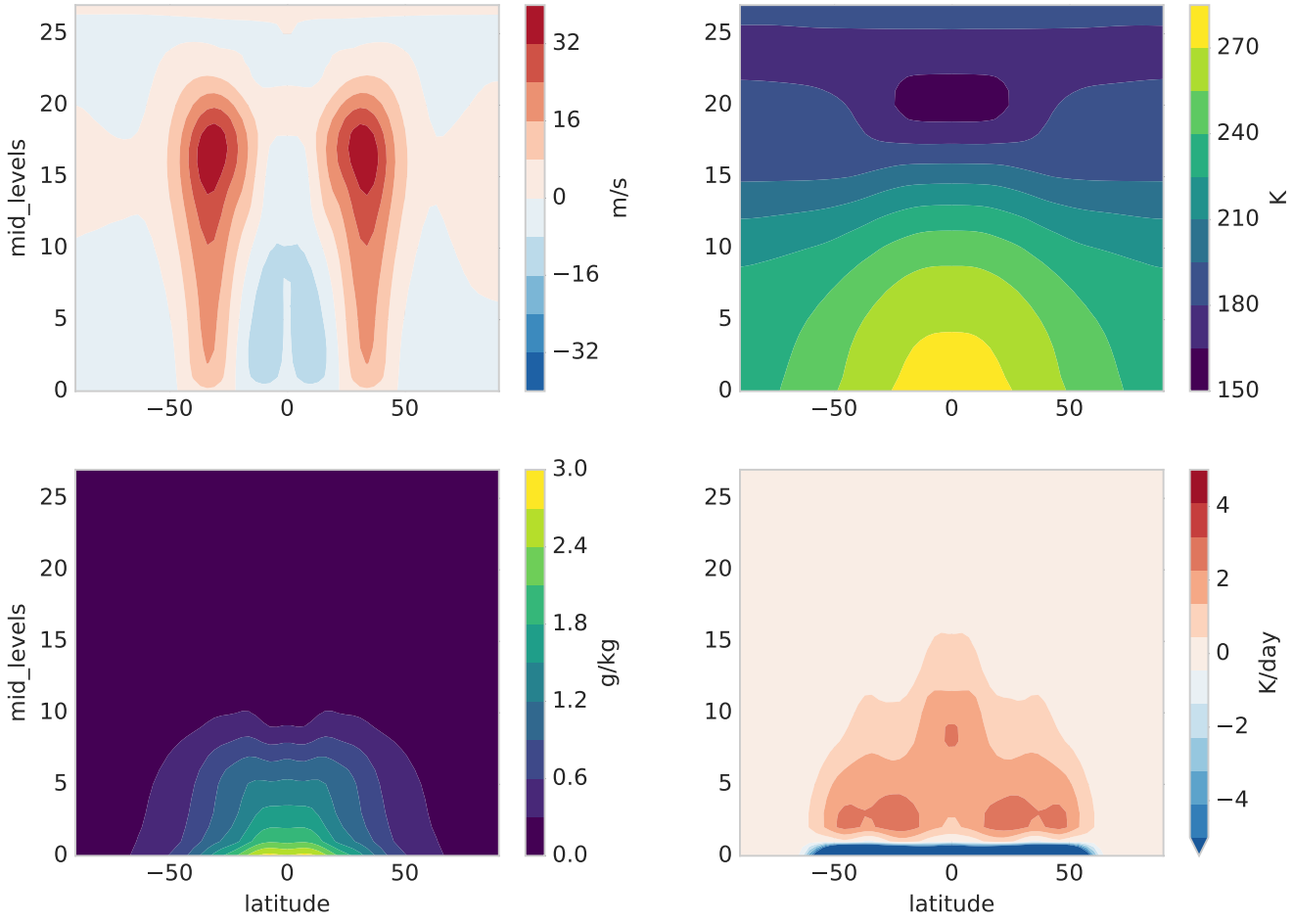

Figure 7. The zonal mean equilibrium profiles in the idealised GCM runs with no seasonal cycle. The plotted fields are, in clockwise order from the top left, the zonal winds, air temperature, convective heating rate and specific humidity respectively.

penetrate to the surface. The zonal mean temperature shows a distinct tropical cold point and an increase in the temperature above the tropopause. The zonal mean convective heating rate shows deep heating in the tropics and much shallower heating in the subtropical areas dominated by descent of air. This simulation ran at a resolution of 128 longitudes, 62 latitudes (or T42 resolution) and 28 levels. It was run using 16 cores (Intel Xeon, 3.10 GHz), and the throughput was approximately 7 hours per model year. The server on which it was run was a shared server which means that these performance figures should probably be a lower bound. We also expect the performance to improve further since currently only the dynamical core and the radiative transfer components run in parallel (using openMP).

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-100
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 April 2018

10 Conclusions and Future Avenues

`sympl` and `climt` represent a novel approach to climate modelling which provides the user with fine-grained control over the configuration of the model. `sympl` provides a rich set of entities which describe all functionality typically expected of a climate model. This set of entities (or classes) allows `climt` to be an easy to use climate modelling toolkit by allowing decisions about model creation and configuration to be made at a single location (the run script) and without ambiguity. The modular nature of the packages allows for code reuse as one traverses the hierarchy of models from single column model to three dimensional GCMs. We attempt to address concerns about plug-and-play type architectures (Randall, 1996) by ensuring the inputs and outputs of each model are cleanly documented, which makes it clear whether components are compatible or not. The use of Python allows for delegating computationally intensive code to compiled languages while still providing an intuitive and clean interface to the user. This choice also allows users access to a large variety of libraries written in Python for purposes ranging from machine learning to visualisation.

The main focus in the near future would be to add more components to allow `sympl`/`climt` to simulate a more realistic benchmark climate, especially a cloud microphysics scheme. Due to its flexibility, we believe our modelling framework is well suited to the simulation of planetary atmosphere and for exoplanet research, and efforts towards adding components relevant to these fields will also be a priority. Another important component to add would be a flexible grid interpolation component to allow interaction between components based on different model grids. Modifications are also being made to `sympl` to further simplify the writing of model components, with automatic validation of input and output properties and automatic preparation of `numpy` arrays for use by components.

While care has been taken to ensure that parallel computing is possible, we have yet to address the question of distributed memory and computing. While running our model in a simple MPI scenario seems feasible in the near future, more sophisticated configurations with components running in parallel will need some thought and design.

Code availability. `sympl` is available at https://github.com/mcgibbon/sympl. `climt` is available at https://github.com/CliMT/climt.

References

Balaji, V., Benson, R., Wyman, B., and Held, I.: Coarse-grained component concurrency in Earth system modeling: parallelizing atmospheric radiative transfer in the GFDL AM3 model using the Flexible Modeling System coupling framework, Geosci. Model Dev., 9, 3605–3616, https://doi.org/10.5194/gmd-9-3605-2016, https://www.geosci-model-dev.net/9/3605/2016/, 2016.

Clough, S. A., Shephard, M. W., Mlawer, E. J., Delamere, J. S., Iacono, M. J., Cady-Pereira, K., Boukabara, S., and Brown, P. D.: Atmospheric radiative transfer modeling: a summary of the AER codes, Journal of Quantitative Spectroscopy and Radiative Transfer, 91, 233–244, https://doi.org/10.1016/j.jqsrt.2004.05.058, http://www.sciencedirect.com/science/article/pii/S0022407304002158, 2005.

DeLuca, C., Theurich, G., and Balaji, V.: The Earth System Modeling Framework, in: Earth System Modelling - Volume 3, SpringerBriefs in Earth System Sciences, pp. 43–54, Springer, Berlin, Heidelberg, https://link-springer-com.ezp.sub.su.se/chapter/10.1007/978-3-642-23360-9_6, dOI: 10.1007/978-3-642-23360-9_6, 2012.

Donahue, A. S. and Caldwell, P. M.: Impact of Physics Parameterization Ordering in A Global Atmosphere Model, Journal of Advances in Modeling Earth Systems, pp. n/a–n/a, https://doi.org/10.1002/2017MS001067, http://onlinelibrary.wiley.com/doi/10.1002/2017MS001067/abstract, 2018.

Emanuel, K. A. and Živković Rothman, M.: Development and Evaluation of a Convection Scheme for Use in Climate Models, Journal of the Atmospheric Sciences, 56, 1766–1782, https://doi.org/10.1175/1520-0469(1999)056<1766:DAEOAC>2.0.CO;2, http://journals.ametsoc.org.ezp.sub.su.se/doi/abs/10.1175/1520-0469(1999)056%3C1766%3ADAEOAC%3E2.0.CO%3B2, 00553, 1999.

Fraedrich, K., Jansen, H., Kirk, E., Luksch, U., and Lunkeit, F.: The Planet Simulator: Towards a user friendly model, Meteorologische Zeitschrift, 14, 299–304, https://doi.org/10.1127/0941-2948/2005/0043, 2005.

Frierson, D. M. W., Held, I. M., and Zurita-Gotor, P.: A Gray-Radiation Aquaplanet Moist GCM. Part I: Static Stability and Eddy Scale, Journal of the Atmospheric Sciences, 63, 2548–2566, https://doi.org/10.1175/JAS3753.1, http://journals.ametsoc.org/doi/abs/10.1175/JAS3753.1, 2006.

Held, I. M.: The Gap between Simulation and Understanding in Climate Modeling, Bulletin of the American Meteorological Society, 86, 1609–1614, https://doi.org/10.1175/BAMS-86-11-1609, http://journals.ametsoc.org/doi/abs/10.1175/BAMS-86-11-1609, 2005.

Held, I. M. and Suarez, M. J.: A Proposal for the Intercomparison of the Dynamical Cores of Atmospheric General Circulation Models, Bulletin of the American Meteorological Society, 75, 1825–1830, https://doi.org/10.1175/1520-0477(1994)075<1825:APFTIO>2.0.CO;2, http://journals.ametsoc.org/doi/abs/10.1175/1520-0477%281994%29075%3C1825%3AAPFTIO%3E2.0.CO%3B2, 1994.

Jeevanjee, N., Hassanzadeh, P., Hill, S., and Sheshadri, A.: A perspective on climate model hierarchies, Journal of Advances in Modeling Earth Systems, pp. n/a–n/a, https://doi.org/10.1002/2017MS001038, http://onlinelibrary.wiley.com.ezp.sub.su.se/doi/10.1002/2017MS001038/abstract, 2017.

Peng, R. D.: Reproducible Research in Computational Science, Science, 334, 1226–1227, https://doi.org/10.1126/science.1213847, http://science.sciencemag.org.ezp.sub.su.se/content/334/6060/1226, 2011.

Randall, D. A.: A University Perspective on Global Climate Modeling, Bulletin of the American Meteorological Society, 77, 2685–2690, https://doi.org/10.1175/1520-0477(1996)077<2685:AUPOGC>2.0.CO;2, https://journals.ametsoc.org/doi/abs/10.1175/1520-0477%281996%29077%3C2685%3AAUPOGC%3E2.0.CO%3B2, 1996.

Raymond, D. J.: Regulation of Moist Convection over the West Pacific Warm Pool, Journal of the Atmospheric Sciences, 52, 3945–3959, https://doi.org/10.1175/1520-0469(1995)052<3945:ROMCOT>2.0.CO;2, https://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281995%29052%3C3945%3AROMCOT%3E2.0.CO%3B2, 1995.

Reed, K. A. and Jablonowski, C.: Idealized tropical cyclone simulations of intermediate complexity: a test case for AGCMs, Journal of Advances in Modeling Earth Systems, 4, http://onlinelibrary.wiley.com/doi/10.1029/2011MS000099/full, 2012.

Theurich, G., DeLuca, C., Campbell, T., Liu, F., Saint, K., Vertenstein, M., Chen, J., Oehmke, R., Doyle, J., Whitcomb, T., Wallcraft, A., Iredell, M., Black, T., Da Silva, A. M., Clune, T., Ferraro, R., Li, P., Kelley, M., Aleinov, I., Balaji, V., Zadeh, N., Jacob, R., Kirtman, B., Giraldo, F., McCarren, D., Sandgathe, S., Peckham, S., and Dunlap, R.: The Earth System Prediction Suite: Toward a Coordinated U.S. Modeling Capability, Bulletin of the American Meteorological Society, 97, 1229–1247, https://doi.org/10.1175/BAMS-D-14-00164.1, https://journals.ametsoc.org/doi/abs/10.1175/BAMS-D-14-00164.1, 2015.

Valcke, S., Redler, R., and Budich, R.: Earth System Modelling - Volume 3, SpringerBriefs in Earth System Sciences, Springer Berlin Heidelberg, Berlin, Heidelberg, http://link.springer.com/10.1007/978-3-642-23360-9, dOI: 10.1007/978-3-642-23360-9, 2012.

Vallis, G. K., Colyer, G., Geen, R., Gerber, E., Jucker, M., Maher, P., Paterson, A., Pietschnig, M., Penn, J., and Thomson, S. I.: Isca, v1.0: A Framework for the Global Modelling of the Atmospheres of Earth and Other Planets at Varying Levels of Complexity, Geosci. Model Dev. Discuss., 2017, 1–25, https://doi.org/10.5194/gmd-2017-243, https://www.geosci-model-dev-discuss.net/gmd-2017-243/, 2017.