A next generation driver for EAMxx

Peter Caldwell[1], Andy Salinger[2], Luca Bertagna[2], Hassan Beydoun[1], Peter Bogenschutz[1], Andrew Bradley[2], Aaron Donahue[1], Jim Foucar[2], Chris Golaz[1], Oksana Guba[2], Ben Hillman[2], Noel Keen[3], Wuyin Lin[4], Kyle Pressel[5], Balwinder Singh[5], Andrew Steyer[2], Mark Taylor[2], Chris Terai[1], and Paul Ullrich[6]

[1]Lawrence Livermore National Lab, Livermore CA
[2]Sandia National Laboratories, Albuquerque, NM
[3]Lawrence Berkeley National Laboratory, Berkeley, CA
[4]Brookhaven National Laboratory, Upton, NY
[5]Pacific Northwest National Laboratory, Richland, WA
[6]University of California, Davis, Davis, CA

February 24, 2022

NOTE: author list for just this section should be culled to just the people working on this section.

1 Atmosphere Process Coupler

1.1 Introduction

This document details the development of the atmospheric model driver infrastructure for the Simplified Cloud Resolving E3SM Atmosphere Model (SCREAM) project. The atmospheric driver for SCREAM will replace the Energy Exascale Earth System atmospheric model (EAM) driver. The new

Atmospheric Driver (AD) will incorporate many of the same features as the current EAM driver, but will simplify and clean up much of the code base and will be written in a modern C++ framework.

Taking advantage of the modern C++ architecture, the SCREAM–AD will introduce an atmospheric process class which will establish a simple and straightforward interface between the driver and individual atmospheric processes. The passing of variables between processes will be greatly improved. Model developers have often complained about the opaque structure of the current physics buffer (PBUF) approach particularly with the inability to properly audit which variables are being accessed and when. In the SCREAM–AD, the PBUF will be replaced with an improved Field Manager (FM) class which will address many of these concerns.

This document is organized as follows: section 1.2 explains how a process will be implemented within the atm_process class. This will be followed by a description of how processes will be coupled together in section 1.3. Section 1.4 will provide a description of how the Field Manager allows variables to be communicated between processes. Specification of runtime options and input/output will then be briefly described in 1.5 and 1.6, respectively, before conclusions in section 1.7.

1.2 Within an Atmospheric Process

An atmospheric process is defined in SCREAM as any process that can change the model state. This is in contrast to EAM which treats physics parameterizations and fluid dynamics very differently. Handling fluid dynamics and physics parameterizations identically simplifies the AD and provides greater flexibility in the coupling infrastructure between processes (as described in section 1.3). All atmospheric processes in SCREAM will be instances of an atmospheric process class, atm_process. The atmospheric process class will have three basic functions: initialization, run and finalization. Developers of each parameterization are expected to supply implementations of the three functions customized for their scheme:

- Initialization (called during the initialization stage of the AD) will:
  - initialize all input/output (IO),
  - register all process specific variables with the field manager (FM), see section 1.4,

  - allocate all local arrays and pointers.

- Run (called each timestep during the run stage of the AD) will:
  - retrieve variables from FM to be passed to the main process.
  - conversion from the AD data structure to the data structure used within the parameterization of interest, if applicable.
  - call the main process routine,
  - postprocess output to save updated variables to the FM and to stage output for writing to NetCDF or ADIOS format

- Finalization (called during the finalization stage of the AD) will handle deallocation of all local arrays and pointers.

1.2.1 Making Code Portable

To make porting process representations to other models and running processes as standalone executables easy, the initialization, run, and finalize functions of each process will be broken into interface and process tasks. The interface code will grab needed input for the process from the FM and will ensure data is provided to the process calculation in the right format with the right units. It will then call the actual process code, which will do the actual calculation. After the process code completes, the interface code will convert output to the format required by the AD and/or FM. Because all dependencies on SCREAM-specific code are contained within the interface level, the process code will be easy to use outside of SCREAM. Tasks handled by the interface layer will range from simple tasks like obtaining density from the ideal gas law to complex tasks like switching to a different model grid, changing the geographic locations assigned to each processor, or even switching to a different set of processors to enable parallel execution over processes. Interface code will be distinguished from the actual process code by appending *_interface* to the appropriate file names. Developers can also append the suffix *_utils* to files to indicate these are helper files for that process.

1.2.2 Basic SCREAM–AD Utilities

It will be the responsibility of the process development team to design the code for all of the steps mentioned so far in this section. **It will**

be the responsibility of the AD development team to maintain the atm_process class and to provide a checklist for new process development. In addition, there will be a number of standard utilities developed in the SCREAM–AD that will assist process developers in bringing their processes into the SCREAM framework. The most powerful of these tools will be the field manager, which is the subject of section 1.4. The AD will also supply I/O routines, tools for accessing runtime options, energy and mass conservation checks, and timestepping schemes.

To promote consistency across the entire model, the AD will also provide a single location for all universal constants. Developers will be encouraged to search this location for any constants that their specific process uses before adding new, process specific constants. If a constant isn't really universal (for example, parameters used only by a process-specific function) then they will be included in the *<process_name_>_utils* code. Similarly, there will be a C++ class for universal functions like linear regression or conversion from potential temperature to temperature.

1.3 Coupling Between Atmospheric Processes

The initialization, run, and finalize methods for all processes included in the model will be called from top-level functions of the same name. The top-level initialization routine will also load the grid information for the SE dycore and for the physics mesh which depends on it. Our first implementation will use the existing Fortran code (with C++ wrappers) for domain initialization. To further simplify our initial version, we will use the dynamics grid decomposition for physics even though physics benefits from disbursing columns geographically while dynamics does best when all elements on a processor are neighbors. Future versions of SCREAM–AD will explore more sophisticated grid decomposition methods for physics, including use of a physics grid which is independent of dynamics (like [2]). Loading restart data will also be initially handled in the SCREAM–AD using C++ wrappers of Fortran routines from EAM. The top-level run routine will be called once per timestep, and will itself call the run interface function for each included process. These interface functions are themselves expected to call the actual process calculation as described above. It will also be responsible for ingesting inputdata, for writing output, and for interacting with the component coupler in order to obtain surface fluxes. At the finalization step the AD will call the finalization routine for each atmospheric process, deallocate the variables in the field

manager (replacement for PBUF) and will deallocate any variables specific to the AD itself.

The top-level initialization, run, and finalize methods will initially handle processes one-at-a-time, but we will ensure these methods can be extended to operate on all processes simultaneously in the future. Because all processes are instances of the generic atm_process class, it will be trivial to change process ordering or to add new processes. The SCREAM–AD will be designed to follow the process order given as a runtime option in an external YAML file (as described in section 1.5). Because there are physical reasons for preferring certain process orders [1], we envision ability to change ordering primarily as a tool for sensitivity testing and for ease of adding new processes. Attempts to use uninitialized variables will be caught by the FM.

If all processes are going to be treated identically, we can't have physics parameterizations return tendencies and dynamics return updated state. Our solution is to keep track of the state at the beginning of the timestep and a state after one or more processes have acted. All atmospheric processes will receive these two states as input. This allows processes to operate on the most recent state or to compute the tendencies from other processes (by differencing the initial and most-recent state) which can then be dribbled into the new process calculation over a series of substeps. Storing states rather than tendencies is better because converting from tendency to state can result in negative concentrations due to rounding errors which are avoided by insisting that all processes return non-negative values.

For more advanced timestepping and splitting methods, the SCREAM–AD will back out tendencies from differences in state before and after parameterizations were called, then apply those tendencies to obtain a new state for the beginning of the next step.

1.4 Passing Data Between Processes: The Field Manager

Passing variables between processes is an important task for the AD. There are many options for how to do this, but most of them are problematic. For example, including all variables in a single derived datatype is unworkable because thousands of variables are passed between parameterizations and passing them all into/out of each parameterization would severely impact performance. Defining all variables required by multiple parameterizations

in the AD-level "run" routine and passing just the variables needed between routines violates the "uniform input/output" requirement needed for each process to be a generic member of the atm_process class. Including all variables in a single module loaded by all processes makes it too easy to access variables which aren't initialized yet. All of these methods are also problematic from the standpoint of writing restarts. Managing variables in a way which allows all variables needed for restarts to be written out in an automated loop has huge advantages. Ability to associate metadata with each variable is also very helpful. Not only does metadata allow for writing data to a more intelligible format than pure binary (e.g. a netCDF file), it also enables error checking that data dimensions and variable definitions are compatible.

The Physics BUFfer (PBUF) used in E3SMv1 provides a solution to these problems. It requires each parameterization that creates a field to give it the necessary data to allocate the field and assign it a name, as well as whether or not it will have to be written to a restart file. Then any other process can request access to the same field through a pointer. The use of pointers instead of full data arrays reduces the memory footprint of storing multiple copies of the same data. The Field Manager (FM) in SCREAM will extend the PBUF from EAM, with all the functionality of the PBUF along with new features. The essential differences between PBUF and the FM will be described in the next subsections.

1.4.1 Specified intent of variables

When an atmospheric process requests access to a variable in the FM it will also designate how the variable will be used, similar to intent IN, OUT, and IN/OUT in Fortran. This provides the FM with the ability to designate how a variable will be used when it is accessed.

Designating how a variable is used will make it easier to track which atmospheric processes access the variable and where it is changed. Each atmospheric process will have a set of two methods, *required_fields* and *computed_fields*, which will designate the list of fields needed by the process and changed by the process. This will enable the field manager to provide a list of pointers to access those fields that includes that appropriate const correctness, depending on if we expect the field to be changed or not. The field manager will have routines that create a schematic of all field managed variables, what routines use them and for what purpose. By default, variables

declared as *required* will be controlled by declaring these variables as constants such that an error is raised if they are changed (const–correctness). They will be further checked by comparing the time when the variable was last changed against the time when it was last expected to change. For more robust variable audits, the field manager can keep a local copy of the entire set of field managed variables and compare with the same variables after each process is called to make sure only those variables that are intended to be changed are changed. This would be a DEBUG flag compile option.

1.4.2 Variables handled by the Field Manager

Unlike EAM, which used derived datatypes to pass state and tendency information between physics parameterizations and PBUF to pass other variables, the FM will be used to pass *all* variables. This will eliminate the need to pass multiple structures between atmospheric processes. Prognostic variables traditionally included in the EAM 'state' variable will continue to be treated differently. They will be given a special designation in FM which ensures they will be advected and prevents them from being designated output variables by parameterizations. Only allowing the AD layer to update these variables ensures that timestepping is handled centrally and important variables aren't updated by accident. If an atmospheric process needs to change a prognostic variable for internal calculations, it have to create a local copy of that variable for this purpose.

1.4.3 AD data structure

All fields stored in the FM will share a common data structure for consistency. Given that most of the atmospheric processes in SCREAM use the physics domain the initial implementation of the FM will use the physics data structure on the physics map as the standard universal data structure. This means that atmospheric processes that use a different data or a different domain decomposition will need to map the FM managed variable to a local variable in the appropriate data structure in their interface layer. For example, the dynamics interface will have to map data from the physics structure to the dynamics structure before calling the main dynamics routine and will need to map the output from dynamics back to the physics data structure before returning control to the AD. This is what is currently done in E3SM via the *pd_coupling* and *dp_coupling* routines respectively. The FM will have a

set of mapping tools specifically designed to efficiently map between different data structures.

1.4.4 Vectorization

Every process which requests a field will include an optional input that specifies the length of vectorization, N, the process wants to use for that specific field. If the optional input is not exercised than the assumption will be $N = 1$, no vectorization. The FM will then make sure the allocation for that field can accommodate all the requested vector lengths.

1.5 Specifying Runtime Options

Runtime options are a powerful tool from EAM that will be adopted in SCREAM. In EAM, there are two sets of runtime options, those controlled by XML (e.g. env_run.xml) and those controlled by namelist (see user_nl_cam). Currently, the runtime options controlled by XML are related to CIME and involve global run settings such as run length, timestep, coupling frequency, run type, etc. Model specific runtime options are handled by namelist. The C++ coding language does not have an equivalent to the Fortran namelist parser, so a task of developing the SCREAM–AD is to design a user–interface in the C++ architecture. The SCREAM–AD will adopt a YAML interface for handling runtime options.

The combination of treating all processes as objects of the same class and using YAML to parse runtime options provides the flexibility to have the coupling mechanism between atmospheric processes be a runtime option. Similarly, defining the process order will be straightforward using this interface.

1.6 Input/Output

In its initial implementation, SCREAM–AD will adopt PIO2 for input and output. Because PIO2 supports ADIOS, this will allow us to take advantage of ADIOS' ability to write output to many local files rather than enduring the large communication costs needed to write to a single file at very high core counts. Although PIO2 is written in C++, the F90 code layer needed to actually use it is complicated. Thus our initial version of SCREAM will call EAM's F90 I/O routines using C-to-Fortran wrappers. These include

the *addfld*, *outfld*, *add_default* and *whist* routines. Future versions of the AD will address rewriting the input/output routines in C++. Because all variables handled by the FM will use the same data structure format, it will be easier to write output for any variable. Like EAM, it will be possible to write out a variable's state at any point within a timestep even though most output will be sampled at the end of a timestep. It will be possible to write restart files at any frequency, for example hourly, daily or monthly. There will be a change to the standard naming convention used for scream based output, the current filename convention of RUNNAME.cam.hX.DATE.nc will be replaced with the more informative RUNNAME.**scream.TYPE**.DATE.nc. Where TYPE will be the frequency of output, e.g. daily, monthly, yearly, etc.

SCREAM–AD will be built using the same external libraries as CIME and as such will have access to all of the standard CIME tools including the *perf_mod* package of performance metrics. Phase 1 of the SCREAM–AD will use a C++ wrapper to call the set of performance Fortran routines. Taking advantage of the generalized "atmospheric process" object class, the placement of performance timer will be automatic for the initialization, running and finalization of all atmospheric processes. This will be enforced by hard–coding the call to the performance timers inside the atmospheric process class. There will also be the option to include extra timer flags to be called within a specific atmospheric process.

1.7 Concluding Remarks

The list of utilities described here is not an exhaustive list, but it represents the set of tasks and goals we believe are required to produce a working model in the three–year timeframe of the SCREAM project. There are very likely other functions and utilities that will arise over the course of the project. As such, the SCREAM–AD development process will be flexible and prepared to tackle any new issues as they come. For example, when the new component coupler becomes available we anticipate devoting a major amount of resources towards incorporating this into the SCREAM–AD interface.

1.8 Bibliography

References

[1] Aaron S. Donahue and Peter M. Caldwell. Impact of physics parameterization ordering in a global atmosphere model. *Journal of Advances in Modeling Earth Systems*, 10(2):481–499, 2018.

[2] Adam R. Herrington, Peter H. Lauritzen, Mark A. Taylor, Steve Goldhaber, Brian E. Eaton, Julio T. Bacmeister, Kevin A. Reed, and Paul A. Ullrich. Physics–dynamics coupling with element-based high-order galerkin methods: Quasi-equal-area physics grid. *Monthly Weather Review*, 147(1):69–84, 2019.