

ACME-SM: A Global Climate Model Software Modernization Surge

A proposal submitted to the DOE Office of Science
May 4, 2016

Invited Proposal: CMDV-SE Activity
DOE Office of Science Program Manager: Dorothy Koch

Proposing Organization: Sandia National Laboratories

Collaborating Institutions: Lawrence Livermore National Laboratory
Argonne National Laboratory
Pacific Northwest National Laboratory
Oak Ridge National Laboratory

Principal Investigator: Andy Salinger
Computational Mathematics Department, Mail Stop 1318
Albuquerque, NM 87185-1318
Phone 505-845-3523
Email: agsalin@sandia.gov

ACME-SM Table of Contents

Abstract	iii
Proposal Narrative	1
1 Introduction	1
1.1 Motivation	1
1.2 Background	2
1.2.1 The ACME Project	2
1.2.2 Building Upon Related Experiences	3
1.3 Selection of Tasks	5
2 Proposed Research: Pervasive Testing	8
2.1 Improve Build System	9
2.2 Design and Create Unit Tests	10
2.3 Climate Reproducibility Tests	12
2.3.1 Evaluating Climate Statistics Using a Multivariate Approach	13
2.3.2 Perturbation Growth Test	16
2.3.3 Time Step Convergence Test	18
2.3.4 Unified Testing Framework	20
3 Proposed Research: Verification	21
3.1 Verification Process	23
3.2 Targeted Verification Efforts	24
3.2.1 Cloud Microphysics (MG2)	26
3.2.2 Cloud Macrophysics and Turbulence (CLUBB)	27
3.2.3 Aerosol Lifecycle (MAM)	28
3.3 Code Fixes Based on Verification	30
4 Proposed Research: Single-Column Model	30
5 Proposed Research: Code Modernization	31
5.1 Refactor Atmosphere Physics/Dynamics Driver	32
5.2 Next-Gen Coupler	34
5.2.1 Scalable Communication Infrastructure	36
5.2.2 Flexible Coupling Architecture	36
5.2.3 Improved Coupling Methods	38
5.3 Refactor Atmosphere Spectral Element Dycore	39
6 Proposed Research: Software Engineering Productivity Education	43
7 Project Timeline, Deliverables, and Tasks	44
8 Abbreviations and Code Names	45
9 Literature Cited	48

Abstract

Title: ACME-SM: A Global Climate Model Software Modernization Surge

Lead Institution: Sandia National Laboratories

Principal Investigator: Andy Salinger (phone 505-845-3523, email agsalin@sandia.gov)

Co-Investigators: Peter Caldwell (Deputy-PI, LLNL PI) (LLNL); Hui Wan (PNNL PI), Richard Easter, Phillip Rasch, Balwinder Singh (PNNL); Salil Mahajan (ORNL PI), Katherine Evans, Joseph H. Kennedy (ORNL); William Spotz, Irina Kalashnikova Tezaur, Oksana Guba, Brent Perschbacher, Michael Heroux, Michael Deakin, James Foucar (SNL); Robert Jacob (ANL PI), Anshu Dubey, Vijay Mahadevan, Iulian Grindeanu, Jason Sarich, Andreas Wilke (ANL); Vince Larson (UWM)

The ACME Software Modernization (ACME-SM) project takes advantage of a one-time, three year pulse of funding to tackle software infrastructure issues that would otherwise prevent the ACME model from reaching its potential. Specifically, we propose changes that improve the trustworthiness of the model, that prepare the code for exascale architectures, and that allow ACME to tap into cutting-edge computational science.

One focus area of our efforts is increased and improved testing, which is necessary for improved reliability and the ability to rapidly prototype cleaner and more efficient code implementations. Within three years, we intend to provide the ACME team with a more flexible and efficient (CMake-based) build system, a robust unit-testing capability, a reliable and efficient means for determining whether answer-changing modifications have a practical effect on model solutions, and a greatly expanded set of tests. Another focus of our work aims to verify that the existing code is satisfying the intended set of equations. We will target three components of the atmospheric physics for intensive analysis (MG2 microphysics, CLUBB turbulence/macrophysics/convection, and MAM aerosols), with the rest of the code receiving a more cursory evaluation. The verification effort is expected to provide many new tests which will be incorporated into the testing effort. A single-column model configuration will be needed by both the testing and verification efforts; another task in this project will harden this capability and make it available to the ACME community.

We have also targeted three specific areas of the code for intensive rewriting and modernization. The first of these is the interface between atmospheric physical parameterizations and fluid dynamics. By computing physics and dynamics in parallel, the model will run faster, scale to bigger core counts, and will be able to efficiently handle more sophisticated parameterizations without a time penalty. We will also put substantial effort into modernizing the coupler. In particular, we will revise the way communication between components occurs to take advantage of new capabilities in our mesh database library. The driver will be redesigned to allow for more efficient and flexible specification of numerical coupling algorithms, grids, and data structures. Additionally, we will develop a dynamic load balancing/automated decomposition optimization capability for the coupler, and will improve the coupler's regridding capability. Our third target for code modernization is the atmospheric spectral element dycore, which we will rewrite in C++ using Trilinos libraries. The resulting code will be performance portable, will allow for better uncertainty quantification, and will allow for continued leverage of advanced ASC and ASCR technologies.

A theme which cuts across all aspects of our proposal is to establish good programming practices within the ACME community. We will do this in part by setting a good example for code verification and testing with our own efforts. We will also make it easy to follow best-practices approaches by creating robust, easy-to-use testing and verification frameworks. Our proposal also contains an explicit education component aimed at encouraging developers to improve their coding knowledge.

In total, these efforts will put the ACME code base and development team on a much-improved footing for conducting world-class science in support of DOE missions.

1 Introduction

ACME-SM is a software modernization effort for the Accelerated Climate Model for Energy (ACME), which is the US Department of Energy’s Earth System Model. The goal of ACME-SM is to use the surge of resources made available through the three-year CMDV (Climate Model Development and Validation) funding call to bring the ACME model software to a state where the pace and quality of scientific discoveries and mission impact are greatly accelerated.

1.1 Motivation

Climate modeling at DOE is in a transition period due to the confluence of three events. (1) The recent establishment of the ACME project gives DOE the autonomy and mandate to pursue a model that focuses on DOE missions. (2) The disruptive change and continued uncertainty in high-end computer architectures will continue to place high demands on the climate modeling code base, requiring it to adjust to and then capitalize on changes in hardware and programming models. (3) The rapid changes in computational science, as it absorbs key advances in the discipline of software engineering and deploys quality algorithm libraries, are now primed to percolate through the climate modeling code development community, to increase productivity, leverage, and capabilities.

The appropriate strategic response to these three events is to undertake a concerted software modernization effort of the ACME code base. The necessary software modernization work can be mapped into three main themes, which follow directly from the three events listed above:

Theme 1: Improve the trustworthiness of the model for decision support: The focus on impacting DOE missions and moving towards a decision support role demands that we raise the bar in terms of model fidelity, which encompasses model resolution, verification, validation, and trustworthiness of the code base. *The verification, testing, and single-column modeling efforts in this proposal are directly aligned with this Theme 1.*

Theme 2: Improve code agility for adapting to exascale architectures: Being prepared for ongoing computer architecture and programming model changes requires building agility into the code base. This is achieved through modular design, flexibility in data structures, a decrease in the supported code base through code reuse and the leverage of external libraries. The critical enabler for adding agility to the code base is the existence of pervasive testing, which allows developers focused on computational science improvements to quickly verify that they are not affecting climate predictions of the model. *The large efforts in code testing that are being proposed by this project, as well as the three code modernization efforts, directly support Theme 2.*

Theme 3: Improve productivity through leverage of cutting-edge computational science: Progress in the computational science community that improves productivity in the model development process needs to be reaped by the Earth System Modeling effort. A key aspect is to adopt software tools and processes that efficiently scale across dozen of developers, across numerous locations, and with a diversity of expertise, and to institute a culture of continuous process improvement. Another thrust is to tap into the great pool of available expertise and advanced algorithmic libraries. To that end, connections will be strengthened with programs that develop these technologies – the SciDAC Institutes, the IDEAS project, and the enabling technology efforts of the ASC program – and to learn from other code projects that integrate many technologies and strive to incorporate advanced processes (e.g., Trilinos, Albany, ATDM-apps, Sierra, PETSc, Amanzi, CASL-Vera, and upcoming ECP application and software technology projects¹). *The code modernization efforts in this proposal, as well as the software engineering education, testing, and verification components, all support Theme 3.*

¹See Section 8 for a list of Abbreviations and Code Names mentioned in the proposal

Our definition of success for the ACME-SM project is to make significant progress along all three themes, so that after three years the following statements are true:

- The ACME code base and project culture set a high bar for testing and verification.
- The ACME code base has greatly enhanced agility for adjusting flexibly to new architectures due to the ACME-SM efforts in code modernization and pervasive testing.
- The ACME project has greatly expanded its leveraging of computational science expertise through ACME-SM code modernization efforts and improvements in software engineering, and the rate of impact of collaborations with computational scientists on ACME achieving its science goals is accelerating.

1.2 Background

1.2.1 The ACME Project

ACME is the name of DOE's Earth System Model as well as the name of the main project that is developing this model. The ACME code branched from the CESM (Community Earth System Model) version 1.3 in July of 2014. Since then, code development has been focused on delivering ACME v1, which will be used to investigate three main scientific questions in the DOE mission space that require a coupled Earth System Model. These focus on the water cycle, biogeochemistry, and committed sea-level rise over the next few decades.

There have been several significant changes to the model since branching from CESM. New ocean and sea ice models based on the unstructured-grid MPAS (Model Predictions Across Scales) framework have been integrated into the global climate model. A land ice model for the Antarctic Ice Sheet, also based on MPAS, is being coupled in, and includes a new fully-implicit finite element solver. ACME uses a spectral element (SE) fluid dynamics code (known as the dycore), which was previously available as an option in CESM. The v1 release will also feature a new scheme for simultaneous solution of turbulence, shallow convection, and condensation/evaporation (see Section 3.2.2 for details). For the land model, several feature developments have also been undertaken for v1.

The ACME performance team has been working on gathering data on model performance, and has begun efforts to port parts of the model to new architectures. The atmosphere dycore in particular has received attention, with separate versions of key kernels being written for working on GPUs with OpenACC directives, and on threaded machines with OpenMP directives.

While many of the above development efforts build upon existing code and experience from CESM, the software engineering effort in ACME was started from scratch two years ago. ACME adopted a fairly distinct code development workflow, to reflect the focus on the coupled model, and pulled from experiences on computational science projects such as PETSc and Trilinos. ACME uses a git repository hosted on GitHub, making extensive use of the GitHub facilities for issue tracking and code reviews. We have instituted a git workflow in which all new development occurs on feature branches, and are merged to the master branch only after a merge onto an integration branch is completed by trained Integrators, and only after the tests continue to pass.

Testing on ACME continues to be based on system tests, which do full model runs. Compile-time configuration allows a selection of what components are active, versus just a data or stub model. Currently there are only a handful of unit tests or other executables that are written to isolate functionality. The testing team has defined two test suites: a quick one that can run on any developer platform and a more extensive one that targets our production machines. These tests are now dispatched with a Jenkins server (Jenkins, 2016), and reported on a CDash site (<http://my.cdash.org/index.php?project=ACME.Climate>). ACME's configuration and build system is customized and incorporates a lot of knowledge about which configurations make scientific sense for the model. Until recently, the ACME team has had limited knowledge of the CESM

build and testing systems, and had not altered them significantly from CESM. Over the past few months, the ACME team has begun a partnership with the software engineering group of the CESM model to do a refactor of the scripts that are used to configure, build and test both models. These scripts are part of a separable component being used in both ACME and CESM, called CIME (the Common Infrastructure for Modelling the Earth). The teams are upgrading the configuration and testing infrastructure to be based on Python scripts configured with XML data.

ACME continues to use the cpl7 coupler developed for CESM (Craig et al., 2012). cpl7 includes a main driver that calls all of the ACME components that in turn have implemented the driver API defined by cpl7. Additional methods used in a coupler such as merging and diagnostics are also included. The underlying communication infrastructure is still the Modeling Coupling Toolkit (MCT) which was also part of cpl6 (Craig et al., 2005) and has been part of the CESM family for almost 15 years. All of this software is also included in CIME. While the API is very general and allows any atmosphere, ocean, or other component to be connected to the coupler, it has a fixed number of components (seven), a mostly fixed time-integration scheme for those components and does not allow multiple different atmosphere models (for example) in the same executable. Any changes to the exact choice of components requires reconfiguring and rebuilding the model. The communication infrastructure was written in the Terascale age and is not expected to be performant in the Exascale era. ACME currently has only one team member with expertise in the coupler and needs to build more depth in this vital part of the system.

ACME has adopted a code review process to manage the investments in feature development. There are several gates that must be passed along the way to integration into the model: a design document must be written describing the new development, verification tests must be described and run, performance impact must be estimated and then measured, and validation as part of a fully coupled climate model must be completed.

At this stage, the knowledge base on how to perform verification varies greatly across the ACME development groups. Also, since these requirements were added rather late in the development cycle for v1 features, there was not adequate time to adopt these protocols without missing deadlines. Most critically, the incorporation of verification steps into the formal process for accepting new code into the model shows that ACME leadership is willing to put a high value on the verification process.

The development and integration of new code into ACME for v1 has finished. While the tuning and experiments are just beginning, the code development efforts for v2 are launching. This is a key time to learn from our first iteration and put into place improvements based on our experience. With the ACME focus on the global Earth System Model, on DOE missions, and on DOE computers – and with the external factors of rapid evolution in both software and hardware – there are many reasons why it is the right time to embark on a concerted software modernization effort.

1.2.2 Building Upon Related Experiences

The formation of the ACME-SM proposal is heavily influenced by recent experiences from other projects. As stated explicitly in Theme 3 of this project (see Section 1.1), we believe there is a considerable knowledge base at DOE in computational sciences that can be leveraged to accelerate the development of new capabilities in ACME. The following sections are examples that showcase situations in which the lowering of barriers to collaborations has led to success.

Development of unstructured grid land ice solver under PISCEES SciDAC

Under the PISCEES SciDAC project (PIs Steve Price (LANL) and Esmond Ng (LBNL)), the Albany/FELIX ice sheet velocity solver went from inception to incorporation in the MPAS Land Ice component of ACME in 3.5 years (Tezaur et al., 2015a,b). Albany/FELIX is an unstructured-grid finite element code that heavily leverages the Trilinos suite of computational science libraries (with considerable investments from ASC, ASCR Math, and SciDAC) (Heroux et al., 2005). By leveraging these investments in advanced algorithms, Albany/FELIX has world-class robustness in the nonlinear solver, scalability of the

key linear solver, and uses automatic differentiation for embedded analysis capabilities (Pawlowski et al., 2012a,b; Salinger et al., 2013). Also, Albany/FELIX is also well on the way to performance portable implementations on next-generation processors using the Kokkos programming model from Trilinos (Demeshko et al., 2016).

In our view, a key aspect of the success of this project has been the collaboration. PI Steve Price (LANL) has been the science lead for the project, who (along with collaborator Matt Hoffman) set the target for equations, data sets, validation tests, and science cases. The software and algorithms on the other hand have been owned by the computational scientists of the Albany project code team of Salinger, Tezaur, Perego, and Tuminaro (SNL). By separating the job of science lead from code lead, and taking the time for several members to bridge the gap, we have made excellent progress. This management arrangement is something we look to reproduce in this proposal and its tasks (see Section ??).

Development of a software framework for land ice V&V under PISCEES SciDAC

A separate effort out of ORNL under the PISCEES project is laying the groundwork for efforts under ACME-SM. To manage the significant model development work, the PISCEES project also created and released a software toolkit for verification and validation of model results (<https://github.com/LIVVkit/LIVVkit/wiki>). Unique in the climate community is LIVVkit’s ability to provide provenance to track model developments and verification and validation of the performance of the model.

The software is open-source and of modular design so that climate scientists and developers can easily add new features and capabilities, such as those described below. LIVVkit is written in Python, the same language as the refactored CIME. It targets both desktop and Leadership Class Computing systems so that verification can proceed beyond ice sheets to include the remaining large-scale coupled climate components. LIVVkit is designed to augment embedded unit testing in several ways. It is a web-based tool that provides detailed configuration information for full provenance and pass/fail information for both simple and full model tests. With failed tests, it provides even more detail including spatial comparison plots, statistics of differences, and solver convergence information. With LIVVkit, larger, component-wide and high-resolution test cases can be assessed because it handles a batch queuing system for job submission and post-processing. LIVVkit also performs performance verification, tracking performance bugs and degradations in throughput and scaling.

This is another example of how the partnership between climate scientists and computational experts resulted in a capability beyond what either community would have been capable of alone.

The Multiscale SciDAC code modernization efforts

In the Multiscale project, lessons learned from the use of third-party solvers and mixed languages provide insight and experience that are critical to the success of this project. Whereas Albany/FELIX, discussed earlier, and Aeras, discussed next, were built from the outset to use the advanced computational science methods espoused here, the Multiscale project aimed at modernizing existing code. As an example, the Multiscale project and its predecessor created an implicit version of the SE dycore using the Trilinos code base for the solver (Evans et al., 2009) and added performance portability within the Fortran portion of the dynamical core (Archibald et al., 2015).

The Multiscale SciDAC was also the incubator for the verification efforts described in Section 3.2. Time-step convergence tests for both physical parameterizations (Wan et al., 2015) and the dycore (Evans et al., 2016) were implemented for this project. Efforts to identify the equation sets solved for each parameterization have also begun.

Efforts to run atmospheric fluid dynamics and physical parameterizations in parallel (the focus of Section 5.1) were also initiated under Multiscale funding. Exploration under SciDAC funding gives us confidence that the implementation we propose here will be successful. Collaboration between climate scientists, including Peter Caldwell (LLNL), and mathematicians, including Carol Woodward (LLNL), were instrumental to the success of this effort and provides a blueprint for the interaction between climate and computational

scientists that will be a cornerstone of this project.

Aeras: a Trilinos-based atmosphere dycore

Under an LDRD research project at SNL, significant progress has been made towards the development of a prototype atmosphere dycore that is based on the same Albany code and Trilinos libraries as the Albany/FELIX ice sheet solver mentioned above. This code, Aeras (Spotz et al., 2015), has been born with some advanced capabilities by using the computational science infrastructure in Albany. Most notably, the use of C++ templates has enabled automatic differentiation for sensitivity analysis and for efficient calculation of ensembles (Phipps et al., 2015). By embedding an ensemble of calculations as an inner loop in the calculation, instead of an outer loop around the full simulations, large computational gains can be made by amortizing costs, decreasing the number of messages, adding compute intensity, and giving the compilers a trivial loop to vectorize (Phipps et al., 2015; Spotz et al., 2015).

Aeras also has prototyped the use of the Kokkos programming model for writing a single code base that is performance portable to new and emerging architectures (Edwards et al., 2014). Kokkos uses C++ templates to abstract out the kernel launch function, working on both GPUs and threaded architectures with a single code base. Kokkos is special in its approach in that it also abstracts out the array allocation step, allowing us to write array access once without hard-wiring it to a data layout. Aeras has shown promising initial results on portability to architectures with CPUs, Phis, and GPUs (Demeshko et al., 2016). More details will be given in Section 5.3.

Development processes of the MPAS Ocean dycore

The development team for the MPAS-Ocean dycore (Ringler et al., 2013), the new ocean component in ACME v1, has adopted many modern software design and development principles. This includes the process of writing and reviewing design documents for all new features to be added to the model. They have implemented strict version control protocols including code reviews. The team has a number of verification tests and model intercomparison tests that they have used to support the trustworthiness of the model. Overall, the team reports a significant increase in productivity of model development since the adoption of these processes (Ringler, 2015). The MPAS experience has already had a large impact on ACME, with many aspects of the version control and code review process being adopted from the MPAS team.

1.3 Selection of Tasks

Our proposed work is a constellation of tasks that are aimed at addressing the 3 motivating themes from Section 1.1 and which leverage the knowledge and experience gained from past projects in order to ensure a high likelihood of success. Tasks were also chosen to meet several criteria that are unique to the CMDV call. Firstly, with the expectation that CMDV funding is a one-time program, the work **needs to be finished and incorporated into ACME in a three-year time frame**. To ensure that the work truly gets incorporated into ACME, where it can have lasting impact, each task needs to include a current ACME staff member as a team member or targeted customer. The work needs to be appropriate for a surge in funding, to get over a development hump, and to not leave a mortgage for the main ACME project to pick up after ACME-SM is finished. One exception to this last requirement are cases where there is a known gap or deficiency in expertise on ACME and we are using this funding to acquire and train necessary ACME staff.

Secondly, the nature of work **needs to fit the software modernization effort**. Work that is focused on climate science investigation, or algorithmic research, is better suited for established funding programs like ACME itself or the SciDAC partnerships with the ASCR office.

Thirdly, we sought tasks that **involve small cohesive teams that have established working relationships** and/or are co-located at a single lab. This ensures that significant progress will be made within the time frame. There is not the time to attempt a single closely-coordinated effort involving 20 people.

Both the proposal and the proposed work are organized into separate elements that support the three themes defined in Section 1.1. We begin in Section 2 with several efforts in Pervasive Testing, with the

overarching goal of implementing rapid testing of the code base to improve developer productivity, code agility, and trust in the simulation outputs. The first task (Section 2.1) involves an upgrade to the build system with a focus on efficiency of code testing, and an ability to use not only Fortran but also C++ code in the model. There will also be a large effort to incorporate Unit Testing into the ACME project as a means of achieving the necessary code coverage and testing throughput (Section 2.2). Anshu Dubey, who has excellent related experience in a leadership role at the ASCI Flash Center at the University of Chicago, will lead this effort.

We also propose an effort to distinguish between answer-changing modifications which alter model climate (and thus require thorough evaluation before acceptance) and those that do not. Efficient and automated determination of climatological impact is critical for the rapid prototyping of performance changes (e.g. loop ordering or compiler flags) as well as for efficient development of new parameterizations. Because the best approach to this problem is still unclear, we will evaluate three methods for evaluating the climatological impact of model changes. See Section 2.3 for details. The end product of this effort will be a robust framework for testing whether answer-changing modifications to the ACME model have a significant climate impact or not. This framework will incorporate as many aspects of our three candidate approaches as necessary to satisfy ACME’s diverse needs without redundancy.

The next major section of the project, Section 3, targets verification. As climate projects take on more mission critical importance, the level of confidence in the model must increase correspondingly. Verification is the process of gaining confidence in the model implementation by confirming that each piece of code is behaving as intended. There are two main parts to verification: an assessment of the mathematics, or the “numerical” portion of the model; and an assessment of the correctness of the software, or model construction, including performance and output. Validation against observation is of course a critical piece as well, but is not addressed in this proposal. The first task within verification is to develop a common template for how we will perform and present verification evidence (Section 3.1). We will be tapping into the expertise of Sandia staff, who have had a prominent role in establishing V&V methodologies in the nuclear weapons missions.

We have neither time nor staff to perform numerical verification of the entire ACME model, so we will start with a pilot project to document three atmospheric parameterizations that are of critical importance to climate. These are MG2 (Section 3.2.1), CLUBB (Section 3.2.2), and MAM (Section 3.2.3). We choose to focus on physical parameterizations, which treat sub-grid and diabatic effects, because these schemes have not received as much verification attention as the fluid dynamics of the ocean and atmosphere. We will devote some effort to assembling existing verification evidence from other parts of the ACME model into the verification template (Section 3.1). We have reserved some resources in the later years of the proposal to address issues encountered during the verification effort, such as improving the robustness or accuracy of an embedded a solution algorithm (Section 3.3).

The ability of the ACME model to run in a single-column atmosphere mode (Section 4) is required by both testing and verification tasks. This will increase the testability of the code, and greatly improve the productivity of model developers in developing new features.

Our next effort, with details in Section 5, is code modernization. This work involves significant code refactorizations of three critical pieces of infrastructure in order to prepare for exascale machines (motivating theme 2) and to better leverage libraries and expertise from ASCR applied math researchers (motivating theme 3). In all cases, the resulting code base will allow research partnerships between climate scientists and applied math/computational science to flourish.

The first piece of code requiring refactoring is the atmosphere model driver (Section 5.1). This will expose more parallelism by allowing dynamics and physics to be solved concurrently on different processors. This refactor will also increase flexibility and control over timestepping schemes for atmospheric physical parameterizations and will build some necessary in-house expertise on this code base. This work is an expansion/extension of work started under the SciDAC Multiscale project by Peter Caldwell (LLNL).

The second code modernization effort is the writing of a next-generation coupler (Section 5.2). This key component is responsible for exchanging fields (velocities, fluxes, sources) between the different model components (atmosphere, ocean, land, sea ice, land ice). The coupler also includes the driver code, which is essentially the time stepping scheme for the full system. Both elements need to be upgraded for model feature extensions on the roadmap for ACME v2 and subsequent releases. We will refactor the coupler to support exascale concurrency in the full system, and we will pull in code and expertise from ASCR by using the FASTMath SciDAC-supported MOAB (Mesh Oriented datABase library), and leveraging the related CouPE coupling code (previously used for nuclear reactor models). This task is being overseen by Rob Jacob (ANL), the lead of the coupler task for ACME, and will heavily involve two key staff members from the MOAB project. Again, we are taking this opportunity to build expertise within ACME in an area where our experience is small: the coupler driver algorithm. When finished, the code base and expertise will allow us to perform investigations in time stepping / coupling algorithms and dynamic load balancing of the components.

The final code modernization effort is a rewrite of the spectral element dynamical core and transport code (Section 5.3). The new version will highly leverage the Trilinos and Kokkos libraries that are being actively developed at Sandia Labs. This approach will lead to performance portable code, where creative use of C++ templates allow for one version of the code to be performance portable on all future architectures.

Significant experience towards this goal has already been gained under the Multiscale SciDAC project (where the atmospheric dynamics code has been linked to Trilinos for implicit solves and partitioning). Also contributing significant experience in this area is the Aeras project discussed above, which has prototyped key elements of the refactor and from which we are drawing key staff members for this aspect of the ACME-SM project. To avoid the time-consuming verification steps that are needed with a new model, we will do an in-place migration of the existing spectral element dycore in the HOMME code while ensuring that each commit of new code produces results consistent with the original. This project will highlight the productivity-enhancement of performing code development with pervasive testing, and also demonstrate the leverage that is made possible by heavy usage of externally-supported libraries.

In support of Theme 3 - working to improve the productivity of the code development activities under ACME - we are adding a Software Engineering Education component to the proposal, described in Section 6. This effort will target all ACME code developers. The ASCR-funded IDEAS project has the charter to research approaches for software productivity, and ACME-SM will partner with IDEAS project co-lead Mike Heroux (SNL) to curate some educational materials for this audience.

Though described individually, the tasks in this project are strongly interconnected and combine to form a software infrastructure which is greater than the sum of its parts. For example, our verification effort requires a unit-testing capability which does not exist yet. Building the needed capability in Section 2.2 makes verification possible, and in return the verification effort will act as customer for the unit testing team, ensuring that the unit testing framework is robust and useful for model developers. The verification team will also be the primary source of unit tests to include in the ACME testing suite and to use as examples for unit test development. Similarly, the atmosphere dycore refactor requires upgrades to the build system to enable inclusion of C++ code. The single-column model from task 4 is likewise needed for both testing and verification efforts. The atmospheric physics/dynamics refactor will also make use of proposed coupler changes. Our testing and verification efforts are targeted at enabling a culture of pervasive testing in ACME. This culture will be put into practice in our code modernization efforts and single-column model task, which will all showcase this culture and make use of the testing and verification tools we put in place. These best-practices development efforts will be used as examples in our education component. We will leverage the experience of ACME-SM project team members to formulate and refine the necessary educational materials before expanding to the full ACME team. Taken all together, the ACME-SM efforts will move the ACME project to a place where the pace of discovery and innovation is greatly accelerated.

In addition to the proposed work sections above, this proposal includes a Management Plan (Section ??),

a Data Management plan, (Section ??), and a Software Productivity and Sustainability Plan (Section ??). In addition, we have included supplemental materials of a Project Staffing Overview (Section ??), a detailed timetable for deliverables and tasks (Section 7), a list of abbreviations and code names (Section 8, and short CVs for all key personnel.

2 Proposed Research: Pervasive Testing

The first thrust of this proposal will be in the area of pervasive automated code testing. We see this as the key enabler of increased reliability and productivity in the model development process. This work is highly relevant to all three themes of the proposal: improved trustworthiness of the model, preparation for exascale, and leverage of ideas/code from the DOE computational science community.

The rate of improvement of the ACME model, both for scientific discovery and numerical performance, is in large part dependent on the speed at which new model features can be developed and evaluated. The following characteristics of the ACME project present challenges to the rapid development of trusted code.

- Large team: The ACME project alone has over 100 developers, working on hundreds of feature branches, and accepts contributions from numerous collaborative projects.
- Geographically Distributed: The core ACME developers come from seven National Labs, over a dozen buildings, and four time zones.
- Numerous Platforms: To access available computational resources, ACME must run reliably on about 10 different production, testing, and development machines across the DOE labs and NERSC.
- Technically Diverse: The large breadth of climate science and computational science embedded in the ACME model means that developers will generally not fully understand all the climate and numerical implications of their code changes.
- Refactoring for New Architectures: The work to port ACME to new programming models and architectures is accelerating, requiring computational science expertise.
- Evolving Model: New features are being actively developed throughout the five main components and across numerous physics packages.
- Chaotic System: The chaotic nature of earth system dynamics means that roundoff-level numerical changes will quickly lead to order-one differences in the solution.
- Numerous Configurations: There are several options for each component of the ACME model, leading to a wide variety of supported configurations.

Pervasive automated testing is the key to achieving scalability in the code development process. For example, achieving good performance on the next generation of computer architectures may require large code changes for data structures, programming models, languages, and code design. A computational expert who is developing a new block of code needs to know if the new code is correct, and not just if it is efficient. The productivity of this expert depends on their ability to rapidly test performance options. We will create a testing system which quickly and automatically provides a pass/fail answer as to whether the modified code is still correct or not after the changes.

The ACME software engineering team is already making some significant progress. The testing system that was inherited from CESM had numerous testing categories already defined and regression testing comparisons against baselines (a.k.a. gold standard files). The ACME team has made significant progress on automating the launch of established test suites across numerous testing and production platforms (using a utility called a Jenkins tool). The results are displayed on a “CDash” web-based testing dashboard where the entire project can check results, and can scroll through log files to diagnose issues. The software engineering team will continue its effort to expand test coverage across cases and machines.

The goal of the efforts in the Pervasive Testing section is a fast and robust testing system that makes individual developers free to rapidly investigate code alternatives. We have targeted efforts that need a significant investment and new skill sets, complementary to the ongoing work under ACME. All together, we have allocated 3.4 FTEs per year to these testing tasks. The main pieces of development that we have identified as necessary are:

- Improved Build System: Create a build system for the configuration and compilation of the code that is designed to expedite the testing process.
- Unit Testing: Create a unit testing infrastructure, and begin to populate it, so that key pieces of code can be isolated and rapidly tested.
- Climate Invariance Testing: Develop and evaluate methods for automatically differentiating between roundoff-level code changes and true climate changing modifications.

2.1 Improve Build System

A critical piece of infrastructure needed for pervasive testing is the configuration and build system. The configuration and build system of ACME is responsible for choosing which files and features are compiled into the code, setting compile-time configurations, and getting the correct libraries and paths on the link line for creating each executable. The current system uses mostly CIME (Common Infrastructure for Modeling the Earth) code, which is shared between CESM and ACME models. The system is custom, using Perl and Python scripts and XML files for configuration. The system is highly geared towards the scientist user, with configuration options (such as grid specifications) that are matched to input files. With the switch in ACME to use the MPAS-based libraries for ocean, sea ice, and land ice, all of which are fully run-time configurable, there is now an opportunity to reuse much of the compiled code for different problems.

We will migrate to the widely-used CMake system and move the model towards more run-time configurability instead of compile-time. The ability to run many variants of the model without recompiling is critical to being able to rapidly test the model in all of its supported configurations. At present, each test builds its own executable and there is not much parallelism in the build process.

Advantages of migrating to CMake:

1. Open Source tool: The use of a common Open Source tool in general, and CMake in particular, will greatly reduce the future costs of maintaining and extending the system. Firstly, the tool will continue to develop and improve without ACME investment. Secondly, there is considerable online documentation and a plethora of discussion pages. Thirdly, new team members brought onto the project will have a smaller learning curve since they will likely be familiar with the tool from previous projects. The ACME-related projects of Albany, HOMME, UV-CDAT, and Trilinos already use CMake.
2. Faster Builds for Testing: **A new ACME build system will be targeted at the developer, and testability of the code base.** This means taking full advantage of parallel builds (i.e., to use “make -j 64”), in using the same executable for many tests, and in re-using component libraries when re-linking a new executable. CMake has sophisticated dependency management capabilities that make this possible.
3. Extensibility: We will be needing to include a build process for unit testing (Section 2.2) and the Single Column Model (SCM, Section 4). The CMake conversion will give us the in-house expertise to accomplish this.
4. Inclusion of C++ code: Full leveraging of other computational science code and personnel expertise at DOE will involve incorporation of C++ code into ACME. To date, there has been a barrier against inclusion of C++ code because of the difficulties encountered in mixed-language builds. CMake can handle this well and many examples of using CMake to merge F90 and C++ code already exist which

we can leverage in our own implementation. The ability to include C++ code is on the critical path for the code modernization effort proposed in Section 5.3;

5. Run-time configurability: We will work towards full run-time configurability of the model. This greatly eases the overhead of compilation and linking, which can be a particularly costly bottleneck on the file systems of the LCF machines. In addition, a single executable allows for straightforward use of a code coverage tool, that tracks which lines of source code are executed anywhere in the test suite.

Preliminary work has been done by the ACME software engineering team to scope out a CMake-based build system. Showing the promise of a new CMake build, the part of ACME converted to CMake builds and links in parallel and showed 10 or more active processes at a time, while the ACME build system typically shows two processes running at a time. We have encountered some non-standard steps in the current build process, where scripts are used to generate source code files. We know these can be addressed using CMake, since it is a full scripting language that includes the capability to execute custom commands.

We have secured time from the main CMake maintainer of the Trilinos suite (Brent Perschbacher (SNL)) to aid in the conversion of ACME. Andy Salinger and Jim Foucar (SNL) from the ACME software engineering team will ensure that both the code and the knowledge transfer to ACME staff.

The main tasks for the build system activity are:

1. Move to CMake-based build system and achieve highly-parallel builds.
2. Develop solutions to non-standard build steps.
3. Integrate builds of unit tests and SCM model into the system.
4. Handle builds of mixed language code base of C++ and Fortran on all platforms.
5. Move towards complete run-time configurability.
6. Add a code coverage tool that provides automatic reporting of coverage statistics.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Andy Salinger (SNL) and has a total staffing level of 0.45 FTEs per year in addition to some assistance from ACME-funded staff.

2.2 Design and Create Unit Tests

Testing of complex software must reflect the layered complexity and different levels of granularity within the code. The current method of testing in the ACME project has two levels of hierarchy: the lighter-weight “developer” test suite, and the more thorough “integration” test suite. Both the developer and integration test suites are system tests, which means that they run the full code stack as in a science run, although often with a course mesh for a few time steps. This methodology, although very useful, is still too coarse-grained to satisfy all the project needs. System tests alone are not adequate to cover all the paths through the source code with acceptable turn-around time to be useful as part of the development cycle. They also are not good at pinpointing the source of a testing failure. We need finer layers of testing granularity to solve these problems.

The finest level of testing granularity is exercised by unit tests. Our definition of a unit test diverges from their generic definition in the wider software engineering world, because our needs are specialized. Traditionally, unit tests isolate a particular function and specify both the inputs for that function and any information that function gains by calling other functions. The input arguments are mocked up by creating a separate driver code that allocates and fills all arguments to the function with realistic and consistent sizes and data. Because functions in climate code call many functions, which themselves call more functions in a deeply hierarchical way, this traditional definition of unit testing is problematic for our use. Instead we propose to develop traditional unit tests for functions that call no other functions, but we will allow our tests

of higher-level codes to call their dependent functions. By testing lowest-level codes first, then testing the codes that call these functions and so on, we will be able to test all of the code in a hierarchical way.

This approach would provide us with a continuous granularity of testing within the code, while avoiding the expense of full system testing at one extreme, or mocking up every single function on the other. For example in Figure 1, the two unit tests at the bottom work with mocked up dependencies. The test with two colors has some dependencies that can be mocked up, but it also depends on the test below it, which it touches. The mid-level test on the right has no mocked up dependencies, but depends on the test below that it touches. The test on top also has no mocked up dependencies, and it depends on the two mid-level tests. In this scenario, if the left lowest level test fails then the two tests that depend on it will become redundant. But if it passes, then the mid-level test effectively becomes a unit test, and if that passes along with the other mid-level test then the test at the top becomes a unit test. In this way, we can rapidly pinpoint the source of any failures. As a positive side effect some interoperability is also exercised. Note that interoperability is not the chief objective of this section of testing, although taking note of the interoperability exercised in unit testing can help reduce the burden on other sections.

The methodology described above is also how many developers verify their codes when it is under development. They devise tests that give them confidence that the software produces expected results: for convenience we refer to these as *dev-tests* (tests used during development). The complexity of testing grows proportionately to the size and complexity of the software. Because of this, our first line of attack will be compiling all the tests that were used from the earliest days of code development. The tests may come in many flavors, e.g. examining some defining features that only occur if the solution is right for the simplest unit tests, while higher level tests may involve confronting the numerically obtained solution of a simplified problem with an analytical or semi-analytical solution. We recognize that many of these tests may not be readily available anymore. We will have to devise other ways of filling this gap, which is described later in the section. Some work at introducing unit tests to ACME has begun, using pFUnit, a test harness for Fortran codes. The level of coverage varies greatly among different high-level modules.

The ACME project as a whole has to adopt a consistent policy for the degree and rigor of testing at finer scales, and this project will be central to that effort. Developers will not only be encouraged to run dev-tests for any nontrivial development they undertake, they will also be encouraged to turn them into unit tests, and to add them into our unit test suite. There are three advantages of this approach: (1) all new code is better tested independently; (2) unit tests are created with minimal overhead in developer effort; and (3) regular testing of the code has better coverage. Sociological issues are the biggest challenge in adopting such a policy, but growing awareness of interdependencies within the code and need for robust results can help to mitigate this challenge. Modest but significant progress made by the software engineering group in getting the project as a whole to adopt some of the best practices is encouraging. In general, science developers adopt practices when they observe a clear benefit, and we believe that ACME project has passed that threshold. It is now at the stage where policies and practices will be adopted more readily if their implementation can be made friendly to the developers – which is our task.

Providing similar coverage for existing code is a different challenge altogether. Here, the difficulty is not only that covering tests may not be readily available, but also, their usefulness is less obvious. It can be argued that as long as the newly added code is covered by unit testing, existing code modules are

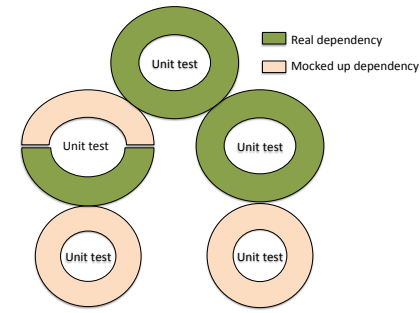


Figure 1: Hierarchical unit testing

adequately exercised by the integrated tests. Such an argument overlooks a few important points. One is that all scientific codes almost always have some tweaking done to them. The ongoing effort of the ACME performance team to have the code perform well on diverse and evolving architectures leaves a large fraction of the code base subject to modification. Anytime a code undergoes such tweaking, it is always more cost effective to use a unit test to verify that it still operates as expected. The second, and less obvious reason is diagnosing the cause of a failure. When a newly added unit, that passed its own unit test, causes a failure in integration, a hierarchical testing regime is much more efficient in pin-pointing the cause of the failure. Additionally, if any code section undergoes refactoring it has to have a layered and hierarchical testing regime to ensure expected results.

In addition to devising good tests, the wide adoption of testing practices also requires easy adoptability, which in turn requires a user-friendly interface. The interface of pFUnit, the unit test harness that ACME has been prototyping, is based on Python scripts. Although not difficult, the use of pFUnit still requires following documentation. We will look to leverage “FlashTest”, a web-based user interface for custom testing developed by the Flash Center at the University of Chicago. The advantage of using this system is that its interface provides web-based editing of the test specification, and also a way to update testing parameters. We can run a pilot project with the available unit tests and customize it based on user feedback. The tool is written in Python, and there is enough expertise in the project to allow such customization. Documentation already exists to support the current unit testing practices. We will continue to maintain the same or a higher standard in documentation. This will include a brief description of each unit test so that a non-expert can also use the output to diagnose the cause of any failure. The procedures, policies and “how to” for creating unit tests will also be created. Additionally, we will periodically hold tutorials designed to introduce new members of the team to various software engineering practices that the project has adopted. We will immerse software engineering team members into science teams for modest periods of time to encourage better understanding and adoption of testing practices. ACME-SM project team members who are working on targeted verification (Section 3.2) and development of the Single Column Model (Section 4) will be early adopters of unit testing.

The proposed tasks for the unit testing effort are as follows:

1. Perform a pilot project on using FlashTest for ACME unit testing.
2. Conduct an inventory of where unit testing will have the most value, and collaborate with developers to begin adding tests.
3. Put unit testing infrastructure and documentation in place so that the barrier is low for adoption by developers.
4. Conduct hackathons and tutorials for targeted ACME developers.
5. Coordinate with the team from the build system task (Section 2.1) and the ACME software engineering team to get unit testing into the nightly test harness and to get those tests reported on the ACME dashboard.
6. Make testing coverage reportable under the code coverage tool, and use it to help target efforts to increase coverage.

See Section 7 for more details on the timeline of deliverables and tasks. This work is led by Anshu Dubey (ANL) and has a total staffing level of 1.3 FTEs per year.

2.3 Climate Reproducibility Tests

Requiring model changes to pass stringent tests before being accepted as part of ACME’s main development branch is critical for quickly and efficiently producing a trustworthy model. Depending on their impacts on model output, code modifications can be classified into three types:

1. Technical changes that continue to produce bit-for-bit identical solutions

2. Changes that cause the model solution to differ, yet produce a statistically identical climate when averaged over a sufficiently long time
3. Changes that lead to a different model climate

Only the third category of changes impacts model climate, and changes of this type should only be implemented within the code after an in-depth demonstration of improvement. However, ACME does not yet possess the ability to distinguish between the second and third categories. As a result, all answer-changing modifications have to be treated as climate-changing, placing an impossible burden on climate scientists to evaluate the credibility of each change. This caused problems with ACME’s ability to incorporate new developments into the model before the v1 code freeze deadline.

The “Climate Reproducibility Testing” task will provide ACME with a robust testing capability to determine whether non-bit-for-bit changes to the model result in a different model climate. This capability, delivered as a software tool, will prioritize efficiency and ease-of-use. In addition to accelerating model development, having a tool that computational scientists could use to quickly accept or reject answer-changing performance modifications will be hugely useful for testing new strategies to improve model efficiency. This capability could be used, for example, to evaluate the impact of compiler optimization or to find kernels where reduced precision or relaxed convergence criteria could be used without affecting model climate.

Testing for climate-changing modifications is difficult because climate simulations are weakly forced and chaotic, so minor changes (e.g. those related to software, programming environment, or hardware) grow rapidly from round-off level differences into completely different weather states. The most obvious approach for testing whether two simulations provide the same climate is to simply compare their climates. A drawback to this approach is that it requires long simulations in order to damp out sampling uncertainty and determine whether long-term trends are altered. Another issue is that deciding whether simulations are climatologically consistent or not can be challenging. Expert judgment is most commonly used, but this is subjective, time-consuming, and often done in a rushed and incomplete way.

We will explore three more efficient and robust approaches for testing whether answer-changing code modifications also change the model climate. We target these three methods because each of them has benefits and drawbacks and it is currently unclear which approach is best or if any single method is sufficient for all use cases. Our first (Section 2.3.1) method is more expensive but provides a richer representation of climate features and tests a wider swath of code. The other two approaches (Sections 2.3.2 and 2.3.3) are considerably more efficient but are correspondingly more limited in their climatic representation. One is more direct but requires more intensive code modifications, while the other requires limited code modifications but is somewhat indirect. Section 2.3.4 describes our unified strategy for implementing and evaluating these methods as well as our plan for delivering a climate-reproducibility test for ACME that is as simple as possible but as complex as needed by the end of the project.

2.3.1 Evaluating Climate Statistics Using a Multivariate Approach

Our first approach evaluates climate statistics of a modified model short simulation (~ 1 year) ensemble with perturbed initial conditions against that of a baseline model. An ensemble provides independent samples and thus is more efficient for computing robust climate statistics than a long simulation where long autocorrelation times reduce the effective sample size (Wan et al., 2014). Although these short runs do not capture low-frequency climate variability, they are representative of the statistical distribution of the possible states of the system after the effects of initial perturbations subside. In addition to quick throughput, running these ensemble simulations as a bundle is computationally more efficient than running a long simulation because with bundling more computations can be packed on each node on machine architectures with accelerators (Norman et al., 2016).

The problem of comparing climate statistics of a modified model run against a baseline model run can be framed as a test of equality of two multivariate empirical non-normal distributions. To decide whether mod-

ifications are climate-changing or not, we propose to use several modern non-parametric (distribution-free) two-sample statistical tests for multivariate data developed by the statistics and machine learning community and widely used in other scientific communities (e.g. digital image analysis, bioinformatics, and genomics). These tests are suitable for multivariate data with high dimensions and low sample sizes.

These multivariate two-sample tests of equality of distributions have never been applied to climate modeling studies and it is not obvious *a priori* which tests would be most suitable for distinguishing climate statistics. We propose to use three popular classes of tests to test the null hypothesis that the baseline and the modified model simulation ensembles belong to the same population. We will generate short simulation (~ 1 year) ensembles for both the baseline and the perturbed model. Our tests will use all the standard model output variables from the two ensembles. These tests are briefly illustrated for the global annual mean of the output variables below:

- **Cross Match Test:** Standardized global annual means of all output variables are concatenated into a single multivariate vector for each ensemble member. The n baseline and m perturbed multivariate vectors are pooled together into a single set of size $N = n + m$ and each vector in the resulting set is optimally paired with the vector closest to it, such that the total distance between each pair is minimized, for some distance metric (e.g. L1-norm, L2-norm, Mahalanobis distance, fractional distance, etc.). The cross-match test statistic, T , is defined as the number of pairs with one vector from each of the control and perturbed ensembles (cross-match). The probability that T is equal to some specific value a_1 if the null hypothesis is true is given by:

$$P(T = a_1) = \frac{2^{a_1} (N/2)!}{\binom{N}{n} \left(\frac{n-a_1}{2}\right)! a_1! \left(\frac{m-a_1}{2}\right)!} \quad (1)$$

The distribution T is based on simple combinatorics, so it does not depend on the assumed distribution of the baseline or perturbed data vectors (Rosenbaum, 2005). When the baseline and the perturbed distributions are similar, cross-matches should occur more frequently. The null hypothesis is rejected if $T > t$ for a critical value t computed from Eq. 1 to match a desired significance level.

- **Energy test:** The energy test is based on the concept of energy statistics of Székely and Rizzo (2004), where they define the test statistic e-distance, e , as:

$$e = \frac{nm}{n+m} \left(\frac{2}{nm} \sum_{i=1}^n \sum_{k=1}^m \|X_i - Y_k\| - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|X_i - X_j\| - \frac{1}{m^2} \sum_{l=1}^m \sum_{k=1}^m \|Y_l - Y_k\| \right) \quad (2)$$

where X_1, \dots, X_n and Y_1, \dots, Y_m are the multivariate vectors of the baseline and perturbed ensembles. Large values of e correspond to different distributions of the two samples. Here, $\|A - B\|$ represents some measure of distance between two vectors. The null distribution of e is not distribution-free but a permutation test approach provides a distribution-free test: the data vectors are pooled together and randomly resampled into the two groups and the e-distance e_k is computed for each such permutation, k . If all distinct possible permutations are drawn, then the permutation test is exact. The values of e from these permutations then describe the null distribution of e . For, a significance level of α , the null hypothesis is rejected if $e > 100(1 - \alpha)\%$ of the permuted estimates of e_k .

- **Kernel Test:** In a kernel two-sample test, a smooth function (kernel) is fitted to each of the multivariate vectors of the two ensembles such that the function values are large for vectors of one ensemble and small for the other (Gretton et al., 2006). The test statistic is the distance between the mean of function values over the two ensembles, called the Maximum Mean Discrepancy (MMD). If the two ensembles

belong to the same population then we expect small values of MMD . Functions in the reproducing kernel Hilbert space (RKHS) like the Gaussian and the Laplace kernels have favorable properties suitable for the kernel test (Gretton et al., 2006). An empirical estimate of the test statistic is given by:

$$MMD = \left(\frac{1}{n^2} \sum_{i,j=1}^n k(X_i, X_j) - \frac{2}{nm} \sum_{i,j=1}^{n,m} k(X_i, Y_j) + \frac{1}{m^2} \sum_{i,j=1}^m k(Y_i, Y_j) \right)^{\frac{1}{2}} \quad (3)$$

where k represents the kernel in its class of functions that maximizes MMD and X_1, \dots, X_n and Y_1, \dots, Y_m are the multivariate vectors of the baseline and perturbed ensembles. The null distribution of MMD has a complex form, but the permutation test approach is routinely applied to establish the null distribution of MMD .

The significance level of these null hypothesis tests determines the false positive (erroneously rejecting a true null hypothesis, Type I error) rates. We will conduct a power analysis of these tests using a large controlled set of ensembles to empirically determine the optimal sample size required to detect a given degree of difference between two ensembles with a given false negative (erroneously accepting a false null hypothesis, Type II error) rate.

Our approach is similar to the approach of Baker et al. (2015) developed by the CESM software engineering group and included in public releases since CESM version 1.4. They use one-year simulations in a principal component based approach to evaluate if global annual means of output variables of a modified model simulation belong to the statistical distribution derived from a large baseline ensemble. Baker et al. (2015) reduce the multivariate hypothesis test to a set of uncorrelated univariate hypothesis tests. The pass/fail criterion of the multivariate hypothesis test is determined empirically as the number of univariate hypothesis tests that are allowed to fail at a particular significance level. Principal component analysis requires that the sample size be larger than the number of variables. Because a typical simulation outputs more than a hundred variables, establishing the climate statistics of the baseline model becomes quite expensive. Our approach is superior to the Baker et al. (2015) approach because the multivariate tests do not require large sample sizes for the baseline model. Being able to use smaller ensemble sizes is particularly useful for ACME development, where climate-changing feature changes are frequently introduced. Also, instead of conducting several univariate hypothesis tests as in the Baker et al. (2015) approach, we only test one null hypothesis (equality of two multivariate sample distributions) with the multivariate tests that are based on robust and satisfying theoretical foundations.

Another issue with the Baker et al. (2015) approach is that in the current form it only works on globally, annually-averaged quantities. This prevents identification of differences that show up at smaller spatial and temporal scales or in the treatment of climate extremes. We plan to implement our tests not only for global means as illustrated above, but also for individual grid points to identify geographic locations where model differences occur. An issue with performing statistical tests for each grid cell is that if one applies a test at 95% confidence level to 100 independent cells, 5 of them will fail the test due to chance alone. Correcting for this is complicated by the fact that global climate model (GCM) grid cells are not independent of each other because large and important spatial correlations exist in the climate system at different temporal scales, much like the field significance of regression patterns. We plan to test the null hypothesis that the spatial pattern of climate variables is similar between the two model variants by using a permutation test. We define a spatial pattern test statistic, t , as the number of grid points that fail the multivariate test. We can derive the null distribution of t by resampling from the pooled data of n baseline and m perturbed ensemble members. We will randomly resample and put them into n and m sized groups. For each such resampling, i , we will then compute t_i , i.e. the number of grid points that fail the test, leading to the empirical null distribution. If all possible permutations are drawn, then the null distribution is exact for n baseline and m perturbed runs.

If $t > 100(1 - a)\%$ of the permuted values t_i , for a significance level a for the grid point multivariate test, the null hypothesis does not hold.

We will also test for differences in extremes. Classical non-parametric distance-based tests of equality of univariate distributions, e.g. Kolmogorov-Smirnov (KS) test, are not robust for distributions with different tails because the cumulative distribution functions converge at the tails. The robustness of the non-parametric multivariate two sample tests for distributions with different tails remains unexplored. We thus plan to evaluate the extremes separately. Extremes of a random variable belong to the family of three-parameter Generalized Extreme Value distributions (GEV) irrespective of the distribution of the variable itself. The parameters of the GEV are normally distributed asymptotically (Coles, 2001). While the univariate GEV theory is extensible to multivariate distributions, high-dimensionality may create difficulties in model computation and validation (e.g. Salvadori and DeMichele, 2013). Sample size is also an important issue with extremes. We have implemented a parallel regionalization framework that evaluates the simulation of climate extremes (seasonal maximum) at each model grid point against observations for the univariate case (Mahajan et al., 2015) using the block maximum approach (Coles, 2001). We improve upon the sample sizes at each grid point by pooling uncorrelated extremes data from surrounding grid points with a homogeneous climate. We plan to implement a similar framework to evaluate the simulation of extremes in a perturbed model run against a baseline model run, testing if the GEV parameters are statistically equal for each grid point. The spatial pattern test statistic would again be the number of grid points where at least one of the GEV parameters is statistically different between the baseline and the perturbed run. We will again derive the null distribution using the permutation approach. The standard model output includes the maximum and minimum values of surface temperature and precipitation. We will implement our test for all these four quantities on a univariate basis.

Short simulations can also be useful for addressing outstanding science questions on shorter time-scales. For example, to study the impact of dust aerosols on Atlantic hurricanes, one could integrate an ensemble of control and dust-forced short simulations only spanning the months of May through December, instead of integrating long control and forced simulations as is the norm. Our set up of the short simulation framework for testing will thus also facilitate the implementation of such experiment designs.

We have organized our proposed work under the following tasks:

1. Implement the ensemble-based multivariate testing strategy
2. Optimize the multivariate tests for ensemble size and simulation length to confidently detect a given degree of difference

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Salil Mahajan (ORNL) and has a total staffing level of 0.7 FTEs per year.

2.3.2 Perturbation Growth Test

Our second approach tests whether differences in atmospheric state (initially we focus on temperature) between a simulation produced in a trusted environment and a “test” simulation are larger than expected. Our expectations are defined by documenting the differences in two simulations produced by roundoff-level changes in initial conditions in a trusted environment. This type of approach was first advocated by Rosinski and Williamson (1997). In the original formulation, the simulation was evaluated over a two-day period. Simulation differences were examined after each model timestep, and the test solution deemed a failure when departures between the test and trusted solution substantially exceeded the difference between the original and perturbed solution in a trusted environment. The increased complexity and strong nonlinearities in new parameterizations in recent versions of CAM have made the test unusable for verification of solution integrity. Therefore, we have developed a variant of this strategy where the magnitude of differences between simulations are checked to be within expected tolerances after each parameterization call over one

timestep. To assure that all parameterizations are exercised many times for many situations we repeat the test for many initial conditions, and each initial condition is subjected to multiple perturbations. The initial conditions are currently chosen to be from a single day each month. Deviations between trusted and test simulations are monitored for each ensemble member, and growth at any point in the timestep update at any location, by any ensemble member is an indication of a simulation failure. By monitoring the solution after each parameterization updates the model state, the source of model differences can be pinpointed to a specific parameterization, facilitating identification of code that is misbehaving (e.g produced by a compiler, optimization, or hardware problem). We currently test initial conditions with both positive and negative perturbations to increase sampling conditions.

The perturbation growth test currently requires that the equations and algorithms described by the model be “non-stochastic,” using that term in preference to “deterministic because there is no requirement that the equations cannot be chaotic in the Lorenz sense, but the divergence of two solutions must occur in a predictable way.

Our first study used three simulations (a control, a positive initial condition perturbation, and a negative initial condition perturbation) initialized at the beginning of each month of the year. The model was first evaluated in a trusted machine environment to prepare the code. Code preparation (evaluation to identify whether and why two solutions initially differing by a small perturbation diverge rapidly in a trusted environment) revealed a few poor algorithmic choices, or poor coding. We revised poor coding and consulted with parameterization developers to identify whether algorithm choices that introduced discontinuities were intended. They were not, so we revised those algorithms to produce a modified model with better perturbation growth characteristics. We identified about 10 issues in earlier versions of CESM/ACME. Some were climate changing, some were not. The method also required some very minor code rewrites to make sure that for example, random number generators produce the same value at the same geographic location regardless of PE layout.

Trusted solutions produced maximum temperature differences between solutions initiated with and without a perturbation at the end of the first time step at approximately single precision accuracy (i.e. the relative difference is below $10E-7$, which means the absolute difference is below $10E-5$ for the temperature field as temperature is measured in Kelvins and has nominal values of the order $10E+2$). This served as a rough threshold for the expected differences for any simulation started in a trusted configuration.

These same simulations can then be performed in a test environment (e.g. the simulations are repeated with a new compiler, new set of optimizations, new set of math libraries, etc). We compare the divergence of perturbed and unperturbed simulations in the new environment to that of the trusted environment. We also compare the difference between a solution started from a particular initial condition in a trusted environment to the solution started from that same initial condition in the test environment. Our hypothesis is that any simulation started from one of the 12 chosen initial conditions (with or without a perturbation) on a new “test” configuration should show approximately the same rate of solution divergence.

In the remainder of this section we demonstrate our perturbation-growth methodology by applying it to test cases from Baker et al. (2015). For this exercise, we tested results from

- NCAR’s Yellowstone - Intel 15.0 with O3 optimization level - labeled as YS (Intel .O3)
- ALCF’s Mira - IBM XL Fortran compiler with O3 optimization level - labeled as Mira (XLF-O3)
- ALCF’s Mira - IBM XL Fortran compiler with O0 optimization level - labeled as Mira (XLF-O0)
- PNNL’s Cascade computing cluster- Intel with O0 optimization level - labeled as Cascade (Intel-O0)

against trusted baselines computed on PNNL’s Constance computing cluster using the Intel compiler with O0 optimization level. As will become evident, it is not critical what compiler and optimization choice is used to define our baseline trusted configuration. Figure 2 compares error growth on each of these platforms. The Intel 15.0 compiler with O3 optimizations on Yellowstone always fails our test. Runs on Mira also

occasionally fail, with O3 optimization more likely to cause failures than O0. The rest of the computing environments shown in Figure 2 are well below the failure threshold. These results are identical to those found by Baker et al. (2015) but our method is orders of magnitude faster and more efficient. We have tested the method on a wide variety of other cases and found similarly encouraging results.

Based on this demonstration of skill as well as the desirable attributes of our approach (efficiency and ability to identify what parameterization causes solution differences), we would like to further explore and refine this method under ACME-SM funding. We propose to do the following:

1. Assess how many initial conditions and perturbations are needed to be comprehensive in assessing the model behavior.
2. Employ statistical analysis (rather than looking at the plots) to conclude pass/fail objectively for new test scenarios. We would like to explore the use of student's t-tests or Kolmogorov- Smirnov statistical tests in this regard.
3. Implement this method in the ACME v1 model.
4. Modify current workflow so that it can be integrated into the existing ACME testing framework.
5. Confront this method with new scenarios and assess its sensitivity to further explore its strengths and weakness.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Balwinder Singh (PNNL) and has a total staffing level of 0.25 FTEs per year.

2.3.3 Time Step Convergence Test

The time step convergence test is another ensemble-based deterministic test method. Like the perturbation growth test described in the previous subsection, it monitors the model's short-term behavior and thus is computationally inexpensive. Multiple ensemble members starting from different initial conditions are included to account for uncertainties associated with the nonlinear nature of the model equation system. The philosophy behind the method is illustrated in Fig. 3. The ACME v0 atmosphere model shows a positive *self*-convergence rate with respect to time step (blue dots in Fig. 3). When the model has changed (in this example a parameter in the atmospheric physics is perturbed) or when the code is exercised incorrectly, the solutions will not converge to the reference solution of the control model. Therefore, by checking whether the root-mean-square difference (RMSD) between a test simulation and a reference solution is substantially larger than the RMSD between a trusted simulation and the reference, we will be able to determine whether the solution changes are physically significant.

The first version of our convergence test has been implemented and evaluated using the ACME v0 model; but we note that the test procedure does not require any code changes, so the algorithm described below is directly applicable to the v1 model as well. The test procedure includes three steps as described below. Steps 1 and 2 need to be performed every time a new baseline model with modified climate characteristics is established. Between such baseline releases, only step 3 is needed to test a new code version or computing environment.

Step 1: Create a 12-member simulation ensemble with a control version of the ACME atmosphere model in a trusted computing environment, using 1 s time step for a simulation length of 5 min. These are

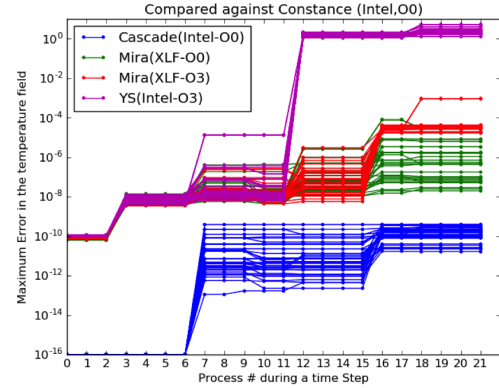


Figure 2: The evolution of maximum temperature difference (unit: K) in various computing environments. The process indices shown on the x-axis refer to different physics parameterizations and/or Fortran code modules executed within one model time step.

considered the reference solutions. The 12 simulations are initialized using model states sampled from different months of a previously performed climate simulation. At the end of the 5 min simulations, save the 3D instantaneous values of the following $N_{var} = 10$ prognostic variables in double precision: temperature (T), meridional wind (V), specific humidity (Q), stratiform cloud liquid and cloud ice mass concentrations (CLDLIQ, CLDICE) and number concentrations (NUMLIQ, NUMICE), and the number concentrations of the three aerosol modes (num_a1, num_a2, num_a3). For post-processing, also save the surface pressure (PS) and land fraction (LANDFRAC).

Step 2: Obtain a 12-member ensemble using the same initial conditions as in step 1, again with the control model in a trusted computing environment, but this time using a 2 s time step. Compute the RMSD between each pair of simulations conducted using the same initial conditions but using different time steps, for each of the 10 prognostic variables listed above. We consider land and ocean as two different domains ($N_{dom} = 2$) and calculate the RMSDs separately. For a generic prognostic variable ψ , we define

$$\text{RMSD}(\psi) = \left\{ \frac{\sum_i \sum_k w_i [\Delta\psi(i, k)]^2 \Delta\bar{p}(i, k)}{\sum_i \sum_k w_i \Delta\bar{p}(i, k)} \right\}^{1/2}, \quad (4)$$

$$\Delta\psi(i, k) = \psi(i, k) - \psi_r(i, k), \quad (5)$$

$$\Delta\bar{p}(i, k) = [\Delta p(i, k) + \Delta p_r(i, k)] / 2. \quad (6)$$

Here $\Delta p(i, k)$ denotes the pressure layer thickness at level k and cell i , and w_i is the area of cell i . Subscript r indicates the reference solution conducted using 1 s time step. This set of $N_{var} \times N_{dom} = 20$ RMSDs are denoted as $\text{RMSD}_{\text{trusted}}$.

Step 3: Repeat Step 2 with a modified code or in a different computing environment. Compute the RMSDs with respect to the reference solutions created in Step 1, and denote the results as $\text{RMSD}_{\text{test}}$. Now define

$$\Delta\text{RMSD}_{j,m} = \text{RMSD}_{\text{test},j,m} - \text{RMSD}_{\text{trusted},j,m} \quad (m = 1, \dots, 12; j = 1, \dots, N_{var} \times N_{dom}) \quad (7)$$

For each prognostic variable and region (i.e. each j), statistical testing is performed to accept or reject the null hypothesis that the ensemble mean ΔRMSD of the $m = 1, \dots, 12$ members is zero. The null hypothesis is rejected (i.e., the ensemble simulations fail the convergence test; the code or software/hardware change is considered climate-changing) if the probability of $\overline{\Delta\text{RMSD}}_j = 0$ is below 0.05% for any prognostic variable and region (any $j = 1, \dots, N_{var} \times N_{dom}$).

We have evaluated this test method using representative test cases from Baker et al. (2015) for three types of situations: (i) compiler or computer changes that were known to produce the same climate; (ii) a compiler optimization setup that was known to produce incorrect results; (iii) various perturbations in model parameters that were expected to produce differences in model climate. In all those cases, our new test was able to issue the expected “pass” or “fail” flag. In particular, our test was able to correctly issue a “fail” flag in a case described in Baker et al. (2015) where NCAR’s climate consistency test incorrectly issued a “pass” flag when a numerical diffusion coefficient was modified in the dynamical core. This is not surprising because our test calculates the RMSDs using instantaneous grid-point values, thus it can detect solution changes at all spatial scales resolved by the model. It is important to point out that while our test has shown higher sensitivity than the NCAR method, the computational cost is much lower. Using the current implementation based on 12-member 5-min simulations, the computational cost of generating the reference solutions and the trusted solutions is a factor of 500 lower than the NCAR method, and the cost of testing a new code or environment is a factor of 30 lower.

The work we propose to conduct in this project includes:

1. Carry out further evaluation of the method using the ACME v1 model, revise the test diagnostics if needed;
2. Optimize the test procedure for ensemble size and simulation length. Assess the resulting false positive rates, and further evaluate the method using additional test scenarios, e.g., code bugs, and compiler optimization settings that caused marginally failing results from the NCAR test;
3. Fully automate the test procedure and integrate it into the ACME testing infrastructure.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Hui Wan (PNNL) and has a total staffing level of 0.1 FTEs per year.

2.3.4 Unified Testing Framework

As outlined above, we will implement and evaluate several methods for assessing whether answer-changing modifications also affect model climate. As much as possible, we will use a common infrastructure to perform these analyses.

In order to perform the ensemble calculations and evaluation needed for a complete testing strategy, a unified ensemble testing framework (UTF) that can launch several model runs and then post-process the results is needed. The development of this key piece of infrastructure will take advantage of both the test harness already existing in ACME and the Land Ice Verification and Validation toolkit (LIVVkit; see Section 1.2). LIVVkit was designed for the ice sheet model component, but most of the capability is easily extensible to other climate components because: (1) LIVVkit was designed for this extensibility, (2) each component uses similar code, libraries, and input/output infrastructures, and (3) the main developer of LIVVkit is the also the principal developer for this project’s UTF. In addition, because the ACME test harness and LIVVkit are both Python-based, we will be able to easily use features from both tools in our analysis. UTF will initially target the multivariate testing approach, because multivariate testing will benefit most from automation of execution and follows the project goal of having co-located teams for each subtask. Subsequently, UTF will be extended further to include the Perturbation Growth and Time Step Convergence tests. Having a common interface will minimize redundant efforts between groups and will prepare these capabilities from research to production mode for all ACME users.

Using the UTF to test our three approaches will allow us to identify the strengths, weaknesses, and redundancies in the methods. Coming up with test cases spanning the circumstances important for ACME will be a major task for this group. Previous studies have only tested for aggressive compiler optimization and model parameter perturbation, but in order to satisfy ACME’s needs we will also have to create test cases involving code bugs and refactoring.

Insights from applying our methodologies to each of our suite of test cases as well as computational cost and time-to-solution will help us identify a test suite that meets the diverse needs of ACME’s software

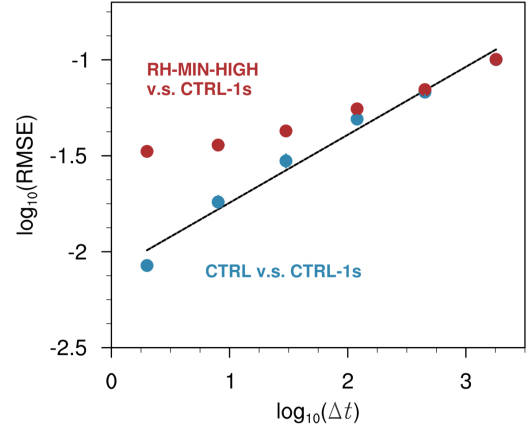


Figure 3: Time step convergence diagram showing the RMS solution differences calculated using the instantaneous 3D temperature field after 1 h of model integration. CTRL refers to the ACME v0 atmosphere model with its default parameter values and at NE30 resolution with 30 vertical layers. RH-MIN-HIGH refers to a model configuration at the same spatial resolution but with a modified value for the threshold relative humidity for ice cloud formation. Blue dots indicate the RMS solution differences calculated relative to reference solutions of the same model configuration and with a 1 s time step. The black line indicate a linear regression between $\log_{10} \text{RMSE}$ and $\log_{10} \Delta t$. Red dots are the RMS solution differences of the RH-MIN-HIGH simulations but calculated against the reference solution (1 s time step) of the CTRL model.

engineers and domain scientists. We will develop a detailed set of documentation for ACME users describing the range of applicability for each method, the types of changes they can and cannot measure and/or characterize, and the optimal tests for each type of model configuration. In the third year of the project we will decide on an optimal climate-reproducibility testing strategy for ACME (which could involve one or all of the methods we are exploring depending on the results of our exploration) and will work to make that method as robust and easy to use as possible.

The unified testing framework tasks are:

1. Develop infrastructure to coordinate the execution and analysis of ensembles, and document the methodology for easy use by the ACME team.
2. Include and demonstrate multivariate testing within UTF.
3. Develop a suite of model modifications to act as test cases for all of our methods.
4. Optimize multivariate testing capability and include perturbation growth and time step convergence approaches in the UTF.
5. Compare performance of each method across our suite of test cases and document areas of strength/weakness, cost, and best usage for each method.
6. Produce climate-reproducibility test for ACME, including code and documentation.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Joseph Kennedy (ORNL) and has a total staffing level of 0.5 FTEs per year.

3 Proposed Research: Verification

Verification and *validation* are distinct yet related terms that have specific definitions across a spectrum of professional communities. Within the computational sciences community, the widely-accepted definitions (Missile Defense Agency, 2008) are:

Verification the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer’s conceptual description and specifications.

Validation the process of determining the degree to which a model, simulation, or federation of models and simulations and their associated data are accurate representations of the real world from the perspective of the intended uses.

Verification has a distinctly mathematical, or analytic context, ensuring that results match the expected behavior from a mathematical perspective. Validation has a distinctly scientific, or data-driven context, revealing the similarities and discrepancies between model results and real-world phenomena, for the purpose of assessing the appropriateness of the chosen equations. Verification is useful for debugging code and reducing numerical artifacts, while validation is useful for determining a model’s biases. Verification can permit unrealistic or purely mathematical testing, while validation is devoted to comparing simulation results to real-world data. **The ACME-SM project will focus solely on verification.**

Many of the components of a climate model such as ACME are developed by scientists with expertise in the relevant academic domain. As scientists, it is human nature that they wish to determine how their model compares to the real world, i.e. they engage almost immediately in the validation process. This is not to say that the verification process is skipped entirely. Typically, a bare minimum of verification is performed to give scientists and developers sufficient hope that their code is bug-free, before moving on to validation. The jump to validation, however, is especially true for the development of subgrid-scale physics parameterizations. The traditional mindset in this case is that inaccuracies caused by approximations underlying the

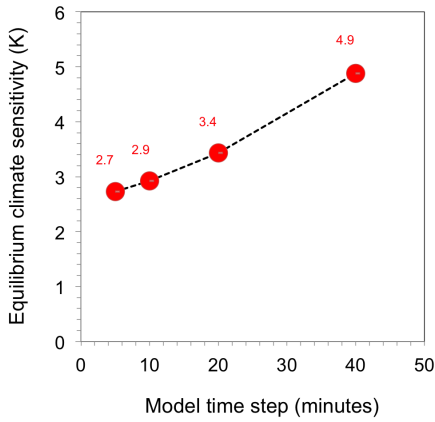


Figure 4: Strong dependence of climate sensitivity (y-axis) on model time step (x-axis) was found in a version of the climate model ECHAM5 of the Max Planck Institute for Meteorology. 40 min was the model’s default time step at T31L19 resolution used by the simulations shown here. The equilibrium climate sensitivity was defined as the 10-year mean, globally averaged surface temperature difference caused by a doubling of CO_2 concentration. Figure was reproduced with permission from Daniel Klocke, German Weather Service.

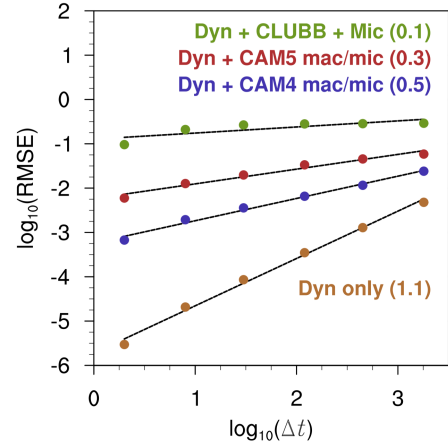


Figure 5: Time step convergence in different configurations of the CESM/ACME atmosphere model. Dyn only: spectral-element dynamical core without parameterized physics; Dyn + CAM4 mac/mic: dynamical core coupled to the CAM4 stratiform cloud parameterizations developed by Rasch and Kristjánsson (Rasch and Kristjánsson, 1998); Dyn + CAM5 mac/mic: dynamical core coupled to the CAM5 stratiform macrophysics (Park et al., 2014) and microphysics (Gettelman et al., 2008; Morrison and Gettelman, 2008); Dyn + CLUBB + mic: dynamical core coupled to CLUBB (Golaz et al., 2002; Larson et al., 2002) and the MG2 stratiform cloud microphysics (Gettelman and Morrison, 2015; Gettelman et al., 2015). The RMSE was calculated using instantaneous temperature written out after 1 h of model integration, with respect to reference solutions using each configuration and with 1 s time step. All simulations were conducted in aqua-planet mode.

chosen format of equations dominate the behavior of the numerical solution, and discretization errors are comparatively small. While that may be the case in some models and with the older and simpler parameterizations, several studies have shown that numerical artifacts can have large impacts on the simulation characteristics (Beljaars et al., 2004; Wan et al., 2013). The unpublished example shown in Figure 4, where the ECHAM model’s climate sensitivity to a doubling of CO_2 was found to be quite sensitive to model time step, is particularly disconcerting because it implies large uncertainties in the climate projection made with such models. Furthermore, our recent analysis with the CESM/ACME-atmosphere model showed that as the model equations become more complex due to the introduction of new, process-based parameterization schemes, numerical problems such as poor convergence rates can become worse rather than better (Figure 5).

In light of our movement towards a decision support role (as articulated in Theme 1 of Section 1.1), we assert that the climate modeling community has an insufficiently pervasive culture of verification testing. The ACME model, like many climate models, would greatly benefit from more extensive testing. A comprehensive improvement of verification within ACME would entail considering *every* equation or set of equations being solved. Resource constraints will require us to prioritize the parts of the model that we consider, as outlined in Section 3.2 and in ongoing consultation with ACME leadership. For those models within ACME targeted for verification improvements, we will develop isolated (and potentially multiple) tests for each equation set, provide public documentation in the form of a website that describes the equations and tests, and report the results of those tests, run on a regular basis. Our approach will be guided, in part, by the experience of colleagues in the nuclear weapons modeling community (e.g., Hodges et al.

(2001)).

This expanded emphasis on verification testing will provide many benefits to the ACME project. It will increase the trustworthiness of the code, either by exposing numerical issues and requiring that those issues be addressed, or by confirming that the model is correctly coded. Early detection and resolution of those issues can help prevent a situation in which problems in the parameterization formulation are concealed by compensating errors in the numerical implementation; it can also help reduce the risk of rejecting new and superior parameterizations because of the exposure of compensating errors in the old model. The expanded emphasis on verification testing will force code modularity and agility, in turn improving developer productivity. Verification testing will also identify opportunities for software and algorithm improvement. When new equations or algorithms are proposed for the model, the existing tests can be leveraged for the new code. And finally, when attempts at validation fail, verification testing will provide a wealth of information for determining the source of those failures.

There is no single prototype for a verification test; it depends upon the nature of the equation set being solved and the algorithm used to solve it. Sometimes analytic solutions are appropriate, often not. The method of Manufactured Solutions, in which a source term is co-opted to generate a created analytic solution, is both popular and effective (see, e.g., Sargent and Fastook (2010)). Convergence tests are appropriate for numerical approximation methods with theoretical convergence rates. Asymptotic behavior can be studied, as well as comparison against benchmark results. The nature of this proposed research is that individual sets of equations and solution algorithms will be considered, and the best verification test, or set of verification tests, will be chosen, documented, implemented, and reported.

There is a lot of commonality in the framework needed for both the ensemble-based testing, previously described in Section 2.3.4 and these verification efforts. For example, one class of verification tests generate convergence plots, and compute the convergence rate, of a series of runs as a function of a discretization parameter (e.g., the time step). The functionality needed to generate these series of runs can leverage the ensemble generation in the Unified Testing Framework (UTF). That way, these type of verification tests will be automatically incorporated into the existing testing harness for ACME, and will be accessed by a common tool for ease of use by ACME users.

There are certainly ACME model components that already employ effective verification testing, most notably the atmosphere and ocean dycores. For these components, we will centralize documentation and reporting. For those components that do not (e.g. most of the physics parameterizations), we will collect existing documentation; augment and post that documentation; discuss, propose and choose relevant verification tests; implement those tests; and report the results of those tests on a regular schedule. Note that due to time and funding constraints, we do not propose in this project to implement verification testing for the entire ACME model. In this section, we outline the targeting of specific model components, plus a strategy to expand these efforts as resources permit. All together, we have allocated 1.8 FTEs per year, plus a modest university subcontract, to these verification tasks.

3.1 Verification Process

Key components of the proposed work include transparency and reproducibility. We believe the *literate programming paradigm* (Knuth, 1984) provides the best approach for achieving both of these goals simultaneously. Literate programming documents contain both source code and documentation for that code. Literate programming tools can extract from the documents the code for compilation or execution, *or* the documentation for processing to generate materials such as PDF or HTML documents.

ACME verification documents will therefore be source code that resides in the ACME repository. Automated tools will utilize those documents to generate documentation for the tests, which can be made accessible from a central web site, such as the ACME Confluence pages, or as PDF manuals. Related tools will also extract code, compile and run tests, and report the results of those tests (integrated with the documentation) to the web.

Documentation for each verification test will include the following:

Description: The description of the model component will include an HTML-based narrative, as well as links to any journal articles, reports, or other documents that illuminate the intent, assumptions, expected inputs, and expected outputs of the model component.

Equations: The equations to be solved will be presented in HTML form, using a utility such as MathJax (<https://www.mathjax.org>) or its equivalent. This will include both governing or constitutive equations, as well as any numerical approximations.

Test descriptions: It is expected that each model component will have multiple verification tests associated with it. Each test will be categorized (analytic, manufactured solution, asymptotic, benchmark, etc.) and given a complete mathematical description.

Expected results: Each description of a verification test will include a statement of the expected results (e.g. expected convergence behavior with respect to time step, resolution, etc.) as well as expected behavior when given unphysical or inconsistent inputs.

Instructions: Each verification test will include instructions on how to build and execute the test.

Results: Verification test results will be reported on a scheduled basis (e.g. short tests may be reported nightly, while longer tests may be less frequent) to a “dashboard” style web page.

Choosing the appropriate literate programming tools will be the first part of this proposed work. As the first physics parameterizations are incorporated into this formal testing infrastructure, the interface that provides the above information will be developed and refined for optimal user experience. Once settled upon, a template of this interface will be developed to ease the incorporation of other model components into verification testing.

Work under this task will transition from creating and modifying the template, to assisting scientists in creating the verification evidence that will complete the template. We will develop tools that aid in creating verification tests, such as performing convergence studies, building on the work in LIVVkit. As time and resources permit, we will extend the subsequently described verification efforts to other model components, prioritized in consultation with ACME leadership. The target for this project will be the models in the ACME v1 release that are also expected to be part of v2.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Bill Spatz (SNL) and has a total staffing level of 0.5 FTEs per year.

3.2 Targeted Verification Efforts

We envisage that ACME should ultimately have the goal of verifying every equation solved in the fully coupled model, while for this project we have selected three initial targets (Sections 3.2.1-3.2.3) based on current awareness of numerical issues in the atmospheric physics and the big impact of those processes on the future climate projection. The three early adopters will serve as a proof-of-concept for further efforts; they will also help improve the verification template outlined in Section 3.1, help prototype the web-based presentation of verification evidence and the python tools for automatically calculating convergence rates and make plots, and help in setting up the infrastructure and documentation so that other parts of ACME can go through the process more easily.

All three targets we have selected are subgrid-scale parameterizations in the ACME atmosphere model (ACME-A). ACME-A started as a fork of the Community Atmosphere Model (CAM) v5.3.83 (Neale et al., 2012), the atmospheric component of the Community Earth System Model (CESM). ACME-A v0 was established following some retuning of parameterizations, and choices of alternate CAM5 options. The main physical and chemical processes that are represented by parameterizations include surface mass and energy exchanges, turbulence, radiation, shallow and deep convection, stratiform cloud and precipitation

formation (macrophysics and microphysics), aerosol lifecycle, and the interactions between radiation and clouds. Certain extensions of the model, such as the version that includes the middle and upper atmosphere, also include representations of additional processes such as the gravity wave drag caused by subgrid-scale sources and turbulent mountain stress (Neale et al., 2012).

Recent work conducted under the SciDAC program has shown that characteristics of clouds and precipitation in CAM5 are strongly sensitive to model time step (Wan et al., 2014); further investigations have revealed that the stratiform cloud macrophysics (Park et al., 2014) and microphysics (Gettelman et al., 2008; Morrison and Gettelman, 2008) are the main sources of time stepping error and the culprits for poor convergence in CAM5 and ACME-A v0 (Wan et al., 2015). Gettelman et al. (2015) showed that the time stepping methods employed in cloud microphysics can cause non-physical artifacts when used with long time steps. Based on these results, and considering the need to document the numerical implementation in detail, we propose to have the stratiform cloud microphysics as one of the main targets for the verification task (Section 3.2.1).

As the ACME-A model evolved from v0 to v1 (the latter is currently under Alpha testing), many additional features were added. A higher-order turbulence parameterization called CLUBB (Cloud Layers Unified by Binormals, Bogenschutz et al., 2013; Golaz et al., 2002; Larson et al., 2002) replaced the older parameterizations for shallow convection, turbulence, and stratiform cloud macrophysics. Initial tests on time step convergence showed that the v1 model has unexpected convergence behavior and substantially larger time step sensitivity (Figure 5), suggesting that in-depth evaluation of the numerical implementation is needed. Furthermore, the CLUBB developer’s group has established a set of clearly defined coding standards and practices, and have created relatively detailed documentation on the continuous and discrete equations. Both the numerical issues and the software readiness make CLUBB a good candidate for the proposed verification task (Section 3.2.2).

The Modal Aerosol Module (MAM) (Ghan et al., 2012; Liu et al., 2012) is a suite of parameterizations developed in DOE-supported projects that provide a comprehensive representation of the lifecycle of natural and anthropogenic aerosols in the atmosphere. It also has gone through substantial changes during the development of ACME-A v1, with new aerosol species and size mode added, and the conceptual and numerical treatments of multiple processes updated. It is important to conduct in-depth verification of the aerosol module since the representation of direct and indirect aerosol effects will have substantial impacts on the climate projection produced by the ACME model.

All three targets listed above have already been reasonably modularized in the current ACME code, meaning that we will be able to start creating and improving the documentation and designing and performing verification tests without having to do any major refactoring of the code. Given the complexity of the parameterizations and the many different types of verification test we plan to implement (e.g. analytic solutions, manufactured solutions, convergence tests, and asymptotic behavior), a hierarchy of codes and model configurations will be used that range from box models of individual processes and off-line drivers for a single parameterization to the single-column model that includes all the subgrid-scale physics as well as the global model in its “operational” configuration. In the first half of the project lifetime, the verification of atmospheric physics will focus mainly on the design of the test cases from perspectives of atmospheric sciences and applied mathematics, and use tools and programming languages that the domain scientists are most familiar with. Increased level of interactions with the other tasks teams are planned for the second half of the project. In particular, we plan to make extensive use of the improved single-column model (Section 4) and provide feedback to that team, and coordinate with the “Pervasive Testing” task team (Section 2) to ensure that all of the tests developed for numerical verification are easily runnable on demand and that the results are reproducible. More detailed verification plans are provided for individual targets in the next three subsections, with bold text referring to the components of the verification template described in Section 3.1.

The verification of atmospheric physics tasks is being overseen by Hui Wan (PNNL), at a level of 0.5 FTEs. She will be contributing to all the tasks in the verification section of the proposal.

3.2.1 Cloud Microphysics (MG2)

The microphysics scheme handles cloud processes that operate on spatial scales comparable to the size of a rain drop or snowflake ($\approx 1 \mu\text{m}$). Capturing the effect of these small scale processes on the GCM grid scale ($\approx 25\text{--}100 \text{ km}$) is challenging and requires assumptions about sub-grid variations in cloud properties and in drop and crystal size. Because many climate processes operate on small scales, microphysics is itself a collection of independent subprocesses. This allows us to divide microphysics into smaller pieces for verification, which makes our task much simpler. Additionally, while the original version of the Morrison-Gottelman microphysics, hereafter referred to as MG1 (Morrison and Gottelman, 2008) was written as a single complicated subroutine, MG v2, or MG2 (Gottelman and Morrison, 2015), was written with modularity in mind. As a result, most of its subprocesses are contained within their own subroutines, simplifying the creation of unit tests. All variables are passed as explicit arguments to the microphysics subroutine, which makes tracking variables straightforward. MG2 will be used in v1 of the ACME model.

The parameterization of each individual microphysical process is relatively straightforward; our main verification challenge will be ensuring that processes interact in a numerically accurate way. MG2 microphysics is implemented in several phases. First, droplet nucleation and processes that are assumed to act instantly (snow melting and rain freezing) are applied. This provides the input state for most processes (autoconversion, heterogeneous cloud and rain freezing, aggregation, accretion, evaporation and sublimation of precipitation, ice deposition including the Bergeron process, and sublimation), which are combined in a parallel-split fashion to once again update the model state. A problem with parallel splitting is that each process has the opportunity to independently consume all of a resource, so overconsumption can occur when processes are combined. This can lead to negative cloud water, for example. To fix this problem, the MG2 code has a section that checks for negative concentrations and rescales the process rates transferring a quantity from one species to another to avoid negative concentrations while maintaining conservation. Once an acceptable updated state is computed, it is used to compute the gravitational sedimentation of cloud, rain, and snow particles. This is followed by final correction for consistency with instantaneous processes (rain freezing, snow melting, removal of grid-scale supersaturation) and diagnostic calculations for radiation (e.g. droplet effective radius).

Our verification efforts will break down into several tasks. First, we will create unit tests for each individual subprocess to ensure it is behaving according to its intended governing equation. Next, we will investigate the rescaling that was performed to avoid negative concentrations. This rescaling was intended to maintain model stability with negligible impact on model behavior, but tests with MG1 show that rescaling occurred often (see Fig. 6). This is problematic because rescaling violates the governing equations for each individual subprocess. Gottelman et al. (2015) show that the impact of rescaling is reduced by the substepping applied in MG2 in a single-column test case using CESM1. We will verify that this result holds in global ACME simulations. Next, we will analyze whether coupling between microphysical processes is adequate by analyzing the microphysics subroutine as a whole. To do this, we will use the Kinematic Driver (KiD) microphysics intercomparison framework produced by Shipway and Hill (2012) as modified by Andrew Gottelman to use in MG2. This framework provides several idealized cases for testing microphysics. We will probably focus primarily on a case of a single column with a perpetual moisture source. We will perform time convergence tests in the KiD context to identify microphysical subprocess coupling problems. All of these tests will be made more robust and incorporated into the routine testing framework described in Section 2. For easy reference, the **descriptions**, **expected results**, and **instructions** for each of these tests will be added to the MG2 verification website.

Most of the basic equations used in MG2 are documented in Morrison and Gottelman (2008). The new precipitation scheme is described in Gottelman and Morrison (2015). Some equations (like vapor deposition), however, have been modified and are not documented in these papers. In addition to providing a general purpose **description** of the MG2 code, our documentation webpage will list all of these **equations**

along with their published reference where available.

We envision the following tasks for MG2 verification:

1. Establish unit tests for each microphysical process.
2. Implement methods for quantifying the effect of conservation corrections.
3. Modernize existing KiD code to work with the version of MG2 used by ACME.
4. Develop tests for identifying error due to time-stepping/time-splitting schemes.
5. Perform convergence studies in both time and vertical-resolution.
6. Document microphysics scheme, tests, and results on webpage.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Peter Caldwell (LLNL) and has a total staffing level of 0.3 FTEs per year.

3.2.2 Cloud Macrophysics and Turbulence (CLUBB)

CLUBB is a parameterization of cloud macrophysics, turbulence, and shallow convection that uses higher-order turbulence closure coupled with an assumed multivariate subgrid probability density function (PDF) of vertical velocity (w), heat content (θ_l , the liquid water potential temperature), moisture content (\bar{r}_l , the total water mixing ratio), and hydrometeors. CLUBB extends the fluid dynamics and transport equations solved by the dycore with extra equations that transport, generate, and dissipate moments of the subgrid PDF. The current version of CLUBB predicts four grid-mean tendencies — $\bar{\theta}_l$, \bar{r}_l , \bar{u} (zonal wind), and \bar{v} (meridional wind) — and supplies them to the host model. In addition, CLUBB predicts the following nine higher-order moments: $\overline{u'^2}$, $\overline{v'^2}$, $\overline{\theta_l'^2}$, $\overline{r_l'^2}$, $\overline{\theta_l' r_l'}$, $\overline{w' \theta_l'}$, $\overline{w' r_l'}$, $\overline{w'^2}$, and $\overline{w'^3}$. Here an overbar denotes a grid-box mean, and a prime denotes a deviation from the mean.

For CLUBB, there already exists a detailed documentation of the continuous equations and their numerical implementation, available from the developers' group website (<http://clubb.larson-group.com>). Portions of this document will be used, along with a new narrative to be created, in order to provide a concise yet complete **description** of the basic concepts behind the parameterization and the theoretical development of the formulation. This updated document will be made available at the verification website with the **equations** presented using MathJax or its equivalent (Section 3.1). An annotated bibliography will also be provided in HTML format to summarize highlights of the more than 30 journal articles on CLUBB published to date.

The design and execution of the verification tests will involve three main focal points. Here we provide only a brief summary, although detailed **test descriptions** will be created during the project.

1. *Monitor the conservation and positive semi-definiteness of the vertically integrated total water and liquid water potential temperature.* Conservation and positive semi-definiteness are **expected** from the definition of the two quantities and the characteristics of the continuous equations; but those properties can be lost in practice because CLUBB uses an implicit time stepping method and a sequentially split solver algorithm. We will implement functionalities to systematically monitor the numerical solutions.

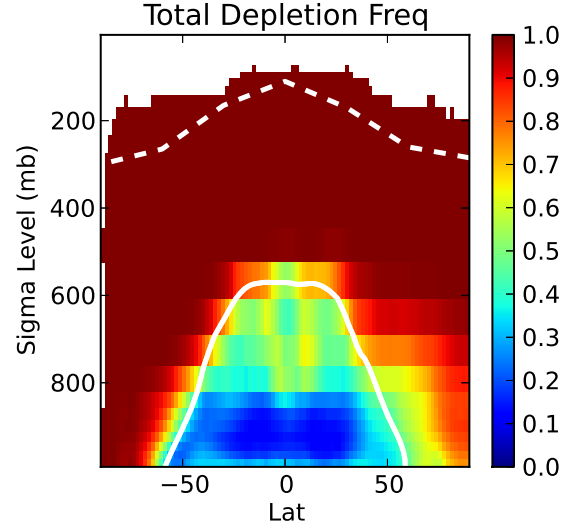


Figure 6: Zonally-averaged frequency of microphysics starting with liquid water mixing ratio $> 10^{-3} \text{ g kg}^{-1}$ and depleting it all within that step (from last 19 years of a 20-year perpetual year 2000 forcing run). Tropopause height (Hoskins, 1998) and average freezing level are included as dashed and solid lines, respectively.

2. *Assess convergence with respect to time step size and vertical resolution.* We **expect** to see first-order self-convergence when the time step is reduced, and we will evaluate the convergence behavior with respect to vertical resolutions much higher than the default v1 model, which has 72 layers. We plan to both assess the combined time-space convergence and to evaluate the convergence in time and space separately, in order to obtain insights from different perspectives.
3. *Implement budget analysis for all prognostic variables in CLUBB.* The budgets are **expected** to be closed (i.e., all source and sink terms sum up to zero) and dominated by physical tendencies rather than numerical artifacts. We plan to add a functionality to create warning messages when artificial sources and sinks, for example those resulting from limiters, exceed a certain threshold. Combined with the two bullets above, the budget analysis can provide an assessment on the quality of the numerical solvers used in the parameterization.

For the verification tests listed above, we will build the capabilities with sufficient flexibility so that they can be used to test either the complete CLUBB parameterization or a selection of simplified configurations with only a subset of the source/sink terms or with certain non-constant quantities replaced by prescribed constants. Such flexibility will facilitate the testing of sub-components and subroutines of the parameterization, and help pinpoint the sources of problems when unexpected results occur. Currently CLUBB can be run either as part of a global atmosphere model or in single-column mode; the latter could be further simplified to produce analytic or manufactured solutions using individual subroutines. We plan to use both frameworks in this task, and will recommend a best setup for each of the verification tests. **Instructions** on how to conduct the recommended tests, as well as the **results** from the ACME v1 model, will be provided at the verification website.

The main tasks for this activity are:

1. Implement and document verification tests for conservation and positive semi-definiteness of the vertically integrated total water and heat content of CLUBB.
2. Implement and document verification tests for assessing CLUBBs convergence with respect to time step size and vertical resolution.
3. Implement and document budget analysis for all prognostic variables in CLUBB.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Professor Vince Larson (UWM), working under a subcontract from PNNL.

3.2.3 Aerosol Lifecycle (MAM)

The atmospheric aerosol is a population of particles having a wide range of sizes (nm to tens of μm), composition, morphologies, and attachment states that are affected by a large number of source, sink, transformation, and transport processes. The Modal Aerosol Module (MAM) developed for CAM5 (Liu et al., 2016, 2012) provides a simplified but fairly complete treatment of the aerosol lifecycle that is suitable for global climate models, and is similar (in terms of complexity) to aerosol treatments in other global models. Some key assumptions of MAM include:

- The aerosol population consists of a small number (three to seven) of log-normally distributed modes that have fixed widths (geometric standard deviations)
- Particles within a particular mode have the same composition (i.e., are internally mixed)
- The composition can be represented by seven to eight species—sulfate, ammonium (optional), black carbon, primary and secondary organic aerosol, soil dust, sea salt, and water

- Only two attachment states need to be explicitly treated—interstitial and cloud-borne, comprising particles suspended in air and particles dissolved or suspended in cloud droplets, respectively

Based on these assumptions, ordinary differential equations (ODEs) or partial differential equations (PDEs) involving the vertical coordinate were derived for two categories of aerosol processes:

- Clear-air, or all-air, processes: emissions, water uptake, condensation and evaporation of trace gases, nucleation (new particle formation), coagulation, sedimentation, and dry deposition;
- Cloud-related processes: aqueous chemistry in cloud droplets, activation and resuspension, wet removal, and subgrid vertical transport by convective clouds.

Verification of MAM is expected to be conceptually straightforward in that the above-listed processes can be isolated to produce manufactured or asymptotic solutions. On the other hand, we anticipate the verification to be a sizable effort considering the large number of processes involved, and the fact that the MAM package consists of numerous code modules and subroutines that interact with the host model at different stages of the physics time integration (in contrast to the implementation of MG2 and CLUBB in CAM5). Currently there are different configurations of MAM that use three, four, or seven modes. Verification in this project will focus on MAM4, which is the default choice in the ACME v1 model.

The existing documentation for MAM (i.e. the journal articles by Liu et al. (2016, 2012) and the CAM5 technical note by Neale et al. (2012)) provides a basic **description** of how individual processes are treated, but contain limited detail and few governing equations. Early papers that developed the modal approach (Binkowski and Shankar, 1995; Whitby and McMurry, 1997) contain **equations** for many of the clear air processes. The changes to MAM for the ACME v1 model have yet to be documented. In this project we will create complete, detailed, and up-to-date MAM documentation and make it available at the verification website.

For testing, our initial plan is to rely heavily on offline testing of individual modules and subsets of modules in isolation. (The modularity of the MAM code facilitates this.) An offline driver exists for several of the clear-air processes; it will be extended to work with the other MAM process modules and to allow testing of individual, subset, or all process modules. In addition to performing analytic or manufactured tests for specific process modules, the offline driver will be used to address overall accuracy and convergence rates caused by operator splitting and the sequential calculation of source and sink processes (e.g., emissions and wet removal of primary aerosol species like black carbon and sea salt). **Test descriptions** and **expected results** will be provided at the verification website, together with detailed **instructions** on the usage of the offline driver and the **results** from the ACME v1 model. As mentioned earlier, aerosol process modules are called from several places in the top-level physics modules of the host model, which poses a challenge for assessing the overall accuracy and convergence properties of MAM in a global-model setup. We plan to develop initial ideas to address this issue.

The main tasks for this activity are:

1. Create and document tests to verify key aerosol processes that are already included in the offline driver.
2. Extend the driver and create and document tests for all other clear-air aerosol processes.
3. Create and document offline tests to verify the accuracy and convergence properties for multiple interacting processes, and explore methods for carrying out such verification in a global model setup.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Richard Easter (PNNL) and has a total staffing level of 0.3 FTEs per year.

3.3 Code Fixes Based on Verification

This proposed verification research and development activity necessarily involves substantial software development. New tests will need to be coded, existing model code may have to be modularized to facilitate testing, and existing tests outside of ACME will need to be integrated into the ACME repository. Common support, such as for convergence analysis, will be added. This leads to the question of responsibility for fixing bugs, especially when verification testing exposes bugs in code supported by other funding mechanisms.

In-depth and comprehensive numerical verification as outlined in Section 3.2 have not been conducted in the past with the CESM/ACME models, hence our efforts will likely uncover numerous bugs and numerical artifacts in the model. Bugs that involve only a few lines of code will be fixed immediately once identified, while flaws related to larger blocks of code might need substantial re-coding. Furthermore, recent experience from the SciDAC Multiscale project has shown that addressing numerical issues such as unexpected slow convergence might require revisiting basic assumptions of a parameterization or substantially revising the numerical implementation, thus requiring resources beyond the scope of this project. Our assumption is that within this project we will focus on revealing problems and implementing simple bug fixes. In consultation with the model developers and the ACME Group Leads, the responsibility for resolving larger issues will be passed to a more appropriate party.

See Section 7 for more details on the timeline for deliverables and tasks. This work will be managed by Andy Salinger and has a average total staffing level of 0.3 FTEs per year, weighted towards the later years of the project.

4 Proposed Research: Single-Column Model

Simulations that test specific aspects of the code while holding other aspects fixed are extremely useful for identifying and understanding errors in complex models. Idealized configurations like this are often the only way to expose compensation between errors in the full model. The single-column model (SCM) exercises the physical parameterizations of a single column of the atmosphere from the full ACME model while specifying fluid dynamics tendencies and surface fluxes. Because the physical parameterizations are one of the most complex parts of a GCM and among the most challenging to develop, having a working SCM is essential for efficient and high-quality model development. It is also an important tool for testing and verification efforts (as noted in Sections 2.2 and 3.2). Having a functioning SCM has been a goal of the ACME atmosphere group since the inception of the project. The surge of resources available in this software modernization project finally allows us to make this investment. This task supports project Themes 1 and 3 by improving model trustworthiness for decision support and by improving developer productivity.

A single-column version of the ACME model already exists in the sense that the model runs in SCM mode, but the existing version has not been tested in many years and is known to contain errors which limit its scientific usefulness. In particular, the current SCM initializes all aerosol to zero, which results in unrealistically pristine simulations which rain far too much (Lebassi-Habtezion and Caldwell, 2014). This problem does not affect purely convective cases in ACME v0 because the v0 convection schemes did not use aerosol information; the ACME v1 model will use aerosol information for shallow convection, however, so fixing this problem becomes essential for the SCM to be useful at all. Because we have already tested several fixes for this problem in Lebassi-Habtezion and Caldwell (2014), we expect implementation to go quickly.

Another issue with the current SCM is that it is hardcoded to work only with the old Eulerian dycore. We would like to modify the code to be dycore-independent but that goal may prove too difficult. Thus we will also explore the easier option of porting the SCM to the SE dycore. Another pathway we will explore is the idea (suggested by David Romps) of replacing the SCM with a global model with 6 identically-initialized and forced grid cells with fluid dynamics specified using the model's specified-dynamics func-

tionality. This last method takes advantage of existing ACME code to reproduce the SCM functionality in a dycore-independent way.

A third deficiency with the SCM in ACME (and in its predecessor, CESM) is lack of publicly available forcing datasets to drive the model. Because the SCM simulates only a single column of the atmosphere at a time, initial and boundary conditions representing a specific geographic location are needed. SCM intercomparisons have been performed for a great variety of climate regimes by the World Climate Research Program’s Global Energy and Water Cycle Experiment Cloud System Study (GCSS) (Randall et al., 2003) and its successor the Global Atmospheric Systems Studies (GASS) Panel, but the forcings for these cases have never been archived in a readily-available way. We will create an online library of forcing and evaluation datasets for these cases to make using the SCM easy. We will also extend this library to include new SCM cases produced by the Atmospheric Radiation Measurement-Best Estimate program (Xie et al., 2010). One feature of the GCSS/GASS test cases is that they idealize certain aspects of their SCM simulations (e.g. no solar insolation, fixed surface fluxes, or specified cloud droplet number) in order to tightly control the sources of differences between models. Implementing these idealizations requires extensive code modifications. Rather than blindly implementing all GCSS/GASS idealizations, we will only implement the idealizations needed to make our simulations comparable with observations (e.g. fixed droplet concentrations) or to avoid forcing inconsistencies (e.g. by applying both computed and specified heating).

The deliverables from this task will be a well-tested and well-documented ACME SCM and an online library of available SCM cases. In year one we will update the existing SCM to work in a trustworthy way; in the remaining years we will work on modernizing the SCM to be consistent with the ACME dycore. Our online library will provide details and references for all available cases (which we will glean from the community) as well as links for downloading the input data, forcing files, run scripts, and available evaluation data for each case. We will also generate extensive tests for ensuring that the SCM mode continues to work correctly as the model evolves. Ability to easily perform in-depth, process-level tests of new parameterizations across a wide variety of test cases will be invaluable for quickly identifying and solving problems with new schemes. Because exercising a single atmospheric column requires negligible computational time and resources, the SCM will also be extremely valuable for in-depth testing of the atmospheric physics component of the ACME model.

A summary of our tasks follows. See Section 7 for a more detailed description of this effort. This work is led by Peter Caldwell (LLNL) and will have a total staffing level of 1.1 FTEs per year.

1. Get ACME SCM working and tested using the Eulerian dycore.
2. Develop website with documentation and library of SCM test cases.
3. Prototype dycore-independent or spectral-element version of SCM.
4. Develop tests to verify that the model is working correctly.

5 Proposed Research: Code Modernization

The code modernization thrust of the proposal is the section of the proposal with both the highest risk and reward potential. Here we are proposing three large code refactoring efforts. These align strongly with Theme 2, the preparation for exascale computers, and Theme 3, the leveraging of computational science libraries, as motivated in Section 1.1. The three tasks that we have selected are:

- A refactor of the atmosphere physics/dynamics driver to access more concurrency and to prepare the code for investigations of time stepping and load balancing/scheduling schemes
- A rewrite of the coupler to add in necessary features for upcoming ACME models, to achieve high performance at high levels of concurrency, and to leverage ASCR investments in mesh databases
- The full refactor of the spectral element atmospheric dynamics dycore to into C++ to have access to

new algorithms and performance portable implementation. All together, we have allocated 5.1 FTEs per year to these code modernization tasks.

These refactors will lay the groundwork for future collaborations with computational scientists, such as those funded by the SciDAC program. By using ASCR libraries, such as MOAB, Trilinos, and Kokkos, – and by the success of the Pervasive Testing improvements that are detailed in Section 2 – the barriers to collaborations with their developers is greatly diminished. In addition, the amount of code base that the ACME project is responsible for maintaining and porting will decrease as a results of these dependencies.

These three efforts will themselves demonstrate the improved productivity that can be achieved by the use of pervasive testing as early adopters of the improvements in all parts of Section 2. We will use a rapid development cycle, in which a few lines of code are changed at a time before testing and committing, that will greatly decrease the costs associated with finding bugs and with merging code bases that have diverged. Another benefit of this work will be to drive acceptance of C++ as a valid choice, along with FORTRAN, as a programming language for climate models.

The primary risks of the three code modernization efforts are that the work will not be completed within three years. In each case, we will carefully monitor progress towards completion. If it becomes evident that we will not reach our goals, we will decide if a more modest goal would still be worthwhile, or if the effort needs to be cut. Another risk for the efforts that use external libraries is if the refactor is a success, the new code moves into ACME, and later proves to be difficult to maintain. To mitigate this possibility, we have chosen external libraries with a long history of support, and in which the maintainers have close relationships with ACME developers.

5.1 Refactor Atmosphere Physics/Dynamics Driver

As noted earlier, atmospheric GCMs consist of a fluid dynamics component (commonly called dynamics or the dycore) which handles resolved-scale fluid motions and a set of schemes (collectively referred to as physics) for handling sub-grid scale motions and diabatic processes. The dynamics component of these models is based on the Navier-Stokes equations. There are many suitable numerical methods for solving these equations, so dycore research focuses on creating maximally efficient solvers for a desired level of accuracy. The processes included in physics, on the other hand, are extremely complex and poorly understood. The cloud component of the physics suite has in particular been implicated in many studies as the main source of the uncertainty surrounding the severity of anthropogenic climate change (e.g. Bony and Dufresne, 2005; Flato et al., 2013). In an attempt to better capture the complex behavior of clouds, the physics component of atmospheric climate models has become more and more complex and expensive over time. As a result, finding ways to perform the physics computations more efficiently is a growing priority.

Parallelization of these models is typically accomplished by assigning a number of atmospheric columns to each processing element. This decomposition works well for physics, which does not require any communication between neighboring grid columns. However, it limits the parallel scalability of the dynamics because splitting the domain over more cores means more frequent communication between neighboring columns on different cores. Because the path to exascale involves very large numbers of processing elements and because communication will continue to be expensive compared to computation, lack of atmospheric scalability is a big problem.

How can ACME improve its parallelism? We will accomplish this by having physics and dynamics run in parallel on separate cores in order to leverage the perfect scalability of the physics component. Since physics and dynamics each take roughly half of the atmosphere-model run time, parallelizing them to run concurrently could result in a speedup of almost 50% (after accounting for additional time spent communicating information between physics and dynamics). Parallelizing physics and dynamics is more efficient than parallelizing individual physics processes (as others have proposed) because it provides the computer with larger chunks of independent work.

There are several challenges to parallelization. First, the ACME model currently uses two pieces of information (the model state and the time derivative for physics) to update dynamics, but only the state will be available if physics is computed concurrently. This reduction of information is likely to degrade model skill. Empirical results by Beljaars et al. (2004) confirm this expectation. We advocate parallel splitting in spite of this deficiency in hopes that the computational gains from this approach (in terms of being able to run at finer resolution, with more complex physics, and/or using a smaller global timestep) will compensate for degradation due to using a cruder time splitting approach. Another problem with parallel splitting is that each process can remove all of a resource when applied individually, so combining tendencies often results in overconsumption of available resources. This results, for example, in regions of negative cloud water. The correct way to deal with these issues is to use a timestep small enough to prevent negative concentrations. Using such a small timestep is not practical, however, so parallel-split approaches typically just replace negative concentrations with zeros. Doing so creates an artificial source term in the governing equations, causing conservation problems. Our initial approach will be to let the global energy fixer, which fixes non-conservation errors by spreading a correction over all grid cells, compensate for lack of local conservation. Addition of a global mass fixer may be necessary. We will also explore more sophisticated strategies for maintaining conservation (e.g. analytic integration using integrating factors over each timestep, careful accounting of missing mass, etc), but ultimately we see small errors in local conservation to be a small price to pay for substantially increased efficiency.

In addition to the mathematical and scientific challenges to parallel splitting, the software engineering needed to run physics and dynamics in parallel on separate cores is formidable. The tools needed for running physics and dynamics on separate cores are similar to those used for coupling the atmosphere, land, ocean, sea ice, and land ice components of the ACME model (See Section 5.2). The coupler already includes code for mapping between components on different grids, for using different domain decompositions, and for coupling components in sequential or parallel fashion. Because the coupler is designed for generality, using it for physics-dynamics coupling will probably be slower than the current approach which is custom-written for physics and dynamics data types. We will explore two solutions to this problem. The higher-risk solution is to convert physics and dynamics to natively use the coupler datatypes. This would allow us to simplify the code base by reusing existing coupler code for physics and dynamics. By replacing the atmosphere-internal data structures with structures that can be used directly by the coupler, the atmosphere-coupler interaction would also be more efficient. The problem with this approach is that it requires changing a great deal of low-level atmosphere model code. We will attempt to minimize these changes by using these new data structures only at the top level of atmosphere subroutines and explicitly passing individual local variables to lower levels of the code. This limits the amount of code changes needed to use coupler data structures and makes unit testing (Section 2.2) easier. Because rewriting the atmosphere data structures is a large task which may run into time-intensive roadblocks, we will also pursue the lower-risk approach of implementing parallel physics and dynamics using the current code for coupling the two. Instead of being satisfied with the forward-Euler timestepping currently used, we will also research better numerical methods for applying physics tendencies in our parallel implementation. We have already tested parallel physics/dynamics coupling in the ACME model by implementing parallel splitting without putting physics and dynamics on separate cores. Initial results indicate that it is running stably and producing a reasonable climate (see Fig. 7).

The ability to use different decompositions and grids for physics and dynamics is not only interesting from an efficiency standpoint, but may also have significant advantages for accuracy. In particular, several physics parameterizations (e.g. deep convection) are based on assumptions which become invalid at fine resolution, while dynamics always becomes more accurate as resolution increases. In addition, parameterizations tend to have ad hoc tuning parameters whose optimal values vary as a function of model resolution. This causes problems when using grids with spatially-varying (a.k.a regionally-refined) resolution. We will test the hypothesis that using a constant-resolution, moderately fine grid for physics provides better results

when dynamics uses regionally-refined grid than with the current approach of refining physics and dynamics in unison.

We will also explore whether there is any advantage to using high-resolution dynamics while keeping physics parameterizations at the grid spacings for which they were designed. It is currently unclear whether the benefits from ensuring that parameterizations operate at their originally-intended resolution outweigh the advantages of better resolving interactions between physical processes. Other studies have explored the impact of coarser physics for an earlier version of ACME's predecessor Williamson (1999) and a version of WRF with a set of physics models similar to those used in ACME (Gustafson et al., 2013), but these studies had a focus on sensitivity to resolution rather than simulation skill. As a capstone effort we will write a journal article demonstrating the practical benefit of using high resolution for dynamics and coarse resolution for physics within the ACME model.

Another benefit of the refactoring of the atmosphere driver code in this way is that it will prepare the code base for future programming models. Specifically, there is significant interest and investment in Asynchronous Multitask (AMT) methods for achieving high throughput on exascale architectures. The investments in more flexible data structures and in looser assumptions about partitioning will improve the code base and create the depth in expertise necessary to adopt AMT programming models as they mature.

To accomplish these goals we propose the following tasks:

1. Produce an implementation of the ACME model where physics and dynamics are calculated in parallel.
2. Publish an article describing the benefits and drawbacks of this approach as well as opportunities for future improvement.
3. Prototype a version of the ACME atmosphere which works natively with the data structures which will be used by the Next Gen Coupler (see Section 5.2).
4. Publish a paper evaluating the benefits of using a coarser, uniform resolution grid for physics while using a high-resolution or regionally-refined grid for dynamics.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Peter Caldwell (LLNL) and has a total staffing level of 1.15 FTEs per year.

5.2 Next-Gen Coupler

An Earth System Model like ACME is a quintessential example of a multi-physics and multi-scale coupled model, which involves several interacting components that simulate the atmosphere, oceans, land, sea ice, land ice, and rivers. This results in components that typically have no datatypes, methods or even implementation language in common. But to simulate the full Earth system, these models must exchange information at their natural boundaries (e.g. the air-sea interface) consistently, while integrating forward in time. The software that is written to link together these model components and to perform the integration is called a *coupler*. The coupler of a climate model serves both a science and a software function. The science role includes spatial interpolation, control of the overall time integration, and occasionally the calculation

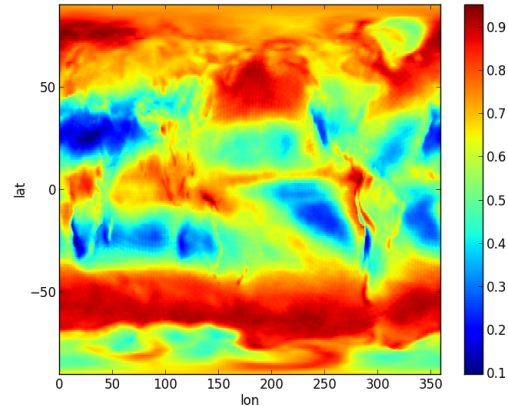


Figure 7: Vertically projected total cloud fraction averaged over a 1 year 1° atmosphere-only simulation with parallel-split physics and dynamics.

of fluxes between models. The software function is to bring the separate components together into one application, facilitating data and control flow between the components.

Couplers are a key component of climate models. Coupling of models normally involves at least three different aspects: the coupling methods (a numerical aspect), the communication infrastructure (a computational aspect), and the coupling architecture (a software aspect). The success of the coupler relies on the underlying numerical methods that provide capabilities such as mapping (interpolation) between different component grids, merging several source components to a destination component, computation of physical quantities such as fluxes, or the computation of convergence diagnostics. The communication infrastructure supports data transfer between components and, depending on the overall design, can be implemented from a high-level driver through abstractions or from within the model components directly. Finally, the coupling architecture is generally associated with the overall control of the system, including the temporal sequencing of the model components and the workflow of the spatiotemporal coupling in multi-physics and multi-scale models. Every coupled climate model application needs to address these aspects, and they are generally implemented by using a combination of community tools and custom-built software (Valcke et al., 2012).

ACME currently uses the coupler included in CIME, the Common Infrastructure for Modeling the Earth. Its coupling architecture is a hub-and-spoke, where a central “driver” component controls the flow of data between the other components, performs any necessary interpolation between numerical grids and controls the time execution of the model and provides the `main()` entry point in the executable. The number of components is fixed apriori and only operator-split time integration is supported with a few slight variations in the underlying sequencing. The coupler provides an API for connecting to the driver to which each component must adapt, typically via the coupled system developer. This API allows for the compile time replacement of, for example, a fully dynamic ocean model with a “data” version that sends the same fields to the coupled system from observations or other “data” in files. But the driver does not allow data and prognostic versions to be in the same executable or for different total numbers of components. CIME provides data and “stub” versions of all expected components in case a developer does not have a complete set.

The communication infrastructure in ACME’s coupler is provided by the Model Coupling Toolkit, or MCT (Jacob et al., 2005). MCT provides a library for the general communication needs of coupled models. Although written to address the coupling needs of an Earth System Model, MCT is a general-purpose library that can be used to couple any models exchanging data represented on a numerical grid, whether structured or unstructured, parallel or serial. MCT requires the user to deep copy their data from their internal types to MCT’s and also to describe the decomposition of their grid to MCT by numbering each grid point. MCT can then determine the minimum number of messages needed to communicate data between any two parallel decompositions of the same grid and provides methods for performing that communication.

MCT also provides parallel interpolation methods for CIME but needs external tools such as the ESMF regridding or Tempest (Ullrich and Taylor, 2015) to calculate the interpolation weights. These tools make up part of the coupling methods in CIME but are typically not included in the CIME distribution. CIME does include other coupling methods such as flux calculations between the atmosphere and ocean and methods for merging data between multiple source models into one destination model.

The above system has served CESM and its family of models well since it was first used in pre-releases of CESM in 2008. Elements of the ACME coupler are used in major European climate models through OASIS-MCT (Valcke et al., 2015). Meeting ACME’s science and technical goals, however, will require rethinking and rewriting many parts of the coupler. The scalability is beginning to be impacted by MCT’s decomposition descriptors and its algorithms for deriving and performing the communication. The effects of the integration and interpolation errors on the accuracy of the coupled simulation have also largely been unexplored. Our proposed work on a next-generation coupler will address all three Themes of this proposal, will make productive use of existing libraries to maximize code reusability, and will primarily improve the agility of the coupler for emerging architectures.

5.2.1 Scalable Communication Infrastructure

MCT’s communication algorithms start with a description of the decomposed numerical grid stored in the GlobalSegmentMap (GSMap). The size of the GSMap scales at slightly less than N , the number of grid points, and a copy is stored on every processor involved in coupler communication. This datatype and the algorithm that compares two GSMaps to find the communication pattern is showing signs of increasing cost (in both memory and computation time) as the grid sizes and number of nodes increases with ACME’s high resolution. We will address these and other limitations in the spatial coupler by building new versions of MCT methods based on the Mesh-Oriented datABase (MOAB) package (Tautges et al., 2004) that is part of the SIGMA toolkit (SIGMA, 2016). MOAB is an array-based data-structure for representing grids, their topology and solution fields associated with grid entities. It natively supports in-memory grid manipulation, efficient adjacency queries for discrete operator assembly, parallel I/O (HDF5-based) and visualization plugins. MOAB has already been successfully integrated in multi-physics coupled models for high-fidelity nuclear engineering (Mahadevan et al., 2014; Tautges and Caceres, 2009) and basin-seismic modeling applications (Yan et al., 2012). MOAB can represent the full range of grid types, including structured, unstructured, and polyhedral grids used in most finite volume/element codes. The full domain decomposition-based parallel model in MOAB provides information about grid entities shared between processors and can represent an arbitrary number ghost entity layers from neighboring processors (Tautges et al., 2012). Additionally, various simulation services that have already interfaced with MOAB include parallel partitioning, based on the ParMetis and Zoltan partitioning libraries (Devine et al., 2002); visualization using the ParaView (Ahrens et al., 2005) and VisIt (Childs et al., 2012) tools; and mesh generation using a variety of meshing tools, including CUBIT (Sjaardema et al., 1994) and MeshKit (Jain and Tautges, 2014).

We will implement MCTs communication methods in a MOAB-based framework. The parallel communication in MOAB is based on the “CrystalRouter” algorithm (Fox et al., 1988), which implements an optimized all-to-all message passing mechanism during the initial setup phase to determine the processor distribution and shared interface resolution. In this algorithm, the message packets containing Tuples of data are sent from a *src* processor to a *dest* processor and are then combined and routed across progressively smaller planes of processors. This adaptively parallel communication algorithm is invoked when either reading an HDF5-based file in parallel using the partition information stored as sets of grid entities in the file, or when we construct a mesh representation in memory and need to resolve the shared interfaces. Hence in a distributed setting after the initialization stage, for any entity shared with other processors, MOAB can explicitly store both the remote processor rank(s) and handle(s) of the entity on those processor(s), on all processors sharing the entity. This cached data structure allows point-to-point communication once the initial mesh is loaded and the shared interface between processors has been resolved. The abstractions to expose data storage defined on mesh entities and sets of entities can also be used to store the coupled fields in parallel, without explicit management of ghost exchanges in the driver. The aggregation based parallel communication algorithms are also utilized extensively in multi-mesh solution remapping strategies explained in Section 5.2.3.

5.2.2 Flexible Coupling Architecture

ACME’s driver currently supports a small number of operator-split integration schemes. While this approach has worked well for recent climate models, it lacks flexibility that could be used to improve accuracy or throughput, and makes it difficult to try higher-order time stepping approaches. It also requires compile-time specification of numerical resolution, components, and other parameters. We will build a new version of the driver to enable runtime specification/determination of simulation characteristics, including grids, numerical coupling algorithms, and component to processor mapping for possible inclusion in future versions of ACME. Such added flexibility will improve both throughput and accuracy, by allowing exploration of a wider set of runtime configurations and by simplifying the overall simulation process. Since many of these coupling capabilities are available in the Coupled Physics Environment, or CouPE (Mahade-

van et al., 2014) that is built based on MOAB and PETSc, we propose to utilize these tools to implement flexible drivers for ACME. The runtime plugin-based architecture allows control of the solver parameters, the underlying spatial projection schemes, the convergence criteria for the global nonlinear solvers at every time-step and the overall time discretization settings to accurately resolve the nonlinearities in the coupled models. Through the usage of MOAB tag descriptions to define the coupled fields between models, the conservative spatial projection of the fields can be driven uniformly from CouPE. The registration based approach in CouPE also allows non-intrusive integration of physics components through language-agnostic MOAB wrappers to describe the mesh along with solution data, and to expose the solver workflow through a unified API. While the availability of nonlinear residuals and Jacobian approximations help to improve the numerical and computational efficiency of the coupled solvers through use of Newton-Krylov or nonlinear Richardson methods, the accelerated variants of Picard iteration will serve as an initial approach to verify the convergence behavior and accuracy impact of the current operator-split integration schemes in ACME.

The coupler allows all fields exchanged to be specified at runtime but the method of specification assumes each physical field passed from a particular model has a unique name (e.g. temperature, humidity, pressure, etc.) which can be associated with a unique text string. But the sub-grid orography scheme planned for ACME v2 requires multiple instantiations of the same physical field, one for each elevation class. These specifications can be cumbersome in terms of user productivity and so we will also add the capability to specify that multiple instances of certain fields are being passed through unique application identifiers.

The current approach of passing coupled information from one component to another necessitates that a translation is required for the data to be communicated in a form that is natively understood by the coupler. This abstraction allows the data to be passed to the driver layer for interpolation seamlessly, while utilizing the processor rearrangement, and finally to be migrated to the target component, where it is converted back to its native data structure representation. This approach simplifies the conceptual data flow, but is inefficient, since it requires extra communication between the components and the coupler “middleman, in addition to requiring a processor partition for the coupler, which could otherwise be used to speed up individual components. This design may also limit flexibility in how components are coupled together numerically. To overcome these drawbacks, we will utilize the parallel decomposition exposed by MOAB and use the aggregation-based data transfer mechanics to reduce the communication cost of this middleman. The data replication can also be avoided by describing the raw component field data pointers in memory to MOAB directly and associating them with data entities according to the physics numbering schema.

There is ongoing work on the coupler within the ACME project that will allow for flexible treatment of time-varying domains in the ocean, land, and land-ice. We will implement this new capability into the new coupler as well.

Dynamic partitioning and load balancing

When the configuration of the model is static, and the load is somewhat uniform during the simulation run, solutions already exist to find the optimum load balance in a static pool of processors using offline tools (Alexeev et al., 2014) and expert knowledge. However, the offline tool requires running three or four simulations at different processor counts for each configuration of the model to build the knowledge base. A dynamic load balancing capability would allow un-tested configurations that often occur during development, to find the optimal balance (which would then remain fixed for the duration of a simulation). Future architectures will create situations where the pool of processors may change during a run because of node failures. Additionally, queuing systems may allow running jobs to dynamically add processors to the available pool, as other jobs finish. In such situations, to keep the model running efficiently with possible changes in the total processor count, dynamic load balancing capability will prove to be crucial.

Note that we are primarily considering handling cases in which either the initial default configuration of processors is not ideal for throughput, which is often the case, or in which the total pool of processors is changing because of machine events external to the running program. We do not expect a great need for

more fine-grained load balancing, where, for example, processors must be shifted from the atmosphere to the ocean after the initialization phase. This is because the regularity of forcing and response in the climate system does not lead to large inter-model differences in the load during a multi-year climate simulation. Similarly, there is little need for dynamic load balance within a component like the atmosphere during a run. Most sources of load imbalance in the atmosphere, such as day-night differences, can be known a priori and efficient static solutions can be found.

To introduce dynamic load balancing ability into ACME, it will first be necessary to make each component's total number of tasks adjustable at runtime. Some of the algorithms that are currently used to perform offline load-balance calculations will need to be added to the runtime system. The load balance calculation should not only consider the FLOPS per node but, to minimize the amount of communication, also the topology of the communication network and the decomposition strategy of each model relative to others. Some of this capability already exists in ASCR libraries such as Zoltan. Zoltan provides a common interface to numerous algorithms for partitioning, including graph-based and geometric methods. It also has a task mapping algorithm that can use information about the machine node layout to improve the association of MPI ranks and tasks, thereby decreasing communication distance and latency. The model will still have the capability of saving the dynamically derived optimum load balance so that subsequent runs on the same machine with the same initial processor count and configuration can skip the possibly expensive load-balancing calculations and will alleviate concerns of determinism, which is essential for solution reproducibility.

The ACME driver will need to interface with system libraries and queue software that can detect and signal a change in the processor pool both available and already allocated to a job. We can compare the dynamically load-balanced configurations with those from the offline programs.

5.2.3 Improved Coupling Methods

The offline interpolation generation programs will likely not be able to meet ACME's future needs. The new watershed-based "grid in the land model planned for v2 will require more advanced "meshless" interpolation methods to move quantities between the land and other models. Throughout the coupled system, solution transfer between various grid types currently ignores the actual structure of the grids and demands that all data be placed on an "A" grid before being passed to the coupler. This transfer must be done directly in order to preserve the accuracy inherent in the discretization method used for each grid type. To support these advanced interpolation methods, we will again rely on the mapping methods implemented within the MOAB library. A more complete description of the mesh in the coupler will allow us to implement more accurate and more powerful interpolation methods such as second-order conservative interpolation. We will also be able to calculate the interpolation weights online as part of the model initialization without resorting to the offline-online model currently in use. As ACME explores more combinations of regionally refined grids, the tool chain for generating interpolation weights has to be rerun, generating files that must be distributed and tracked for provenance. Runtime generation of weights can provide more reliable, scalable and repeatable methods for interpolation in a coupled model like ACME.

The flexible data-structure exposed by MOAB can be accessed through a language-agnostic API to construct a distributed Kd -tree to enable location of points required for interpolation on a target physics mesh. Once the tree construction is complete, the parallel communication pattern is determined once, during the initialization or setup stage and reused throughout the simulation. MOAB supports several mechanisms to compute the interpolation weights using nearest neighbor queries, consistent L_2 projection, and locally conservative intersection-based remapping for unstructured grids. Such higher-order remapping schemes, however, may require limiters to bound the projected data according to the original distribution, and we will employ "callbacks" to let the physics components dictate these filtering mechanisms. More specialized remapping procedures can be flexibly developed within this framework by reusing the parallel communication infrastructure already in place. Additionally, we will verify the spatial coupling methodology that is used in MOAB for the interpolation of both scalar and vector data represented on a sphere, while optionally

also providing interfaces to the Tempest library (Ullrich and Taylor, 2015) to leverage ongoing research and to support the reproducibility of results. Past studies on BG/Q (Mira at ALCF) have shown good strong scalability of the point location and interpolation stages performed on distributed meshes. Hence, the integration of MOAB with the ACME components will enable the spatial coupler to leverage the verified coupling strategies without sacrificing overall parallel efficiency of the solvers.

The work we propose to conduct in this project includes:

1. Modifying the existing coupler to allow runtime selection of component sets, resolution and decompositions.
2. Modifying the existing coupler to use ASCR load balance libraries to provide component processor allocations.
3. Constructing a new alternate version of MCT and the driver that uses MOAB for communication and interpolation between components and getting it to work for an ACME atmosphere-land case.
4. Further extending the new driver to allow different integration schemes.
5. As they become defined, modifying the coupler to use system level information on load balance and resilience.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Robert Jacob (ANL) and has a total staffing level of 1.7 FTEs per year.

5.3 Refactor Atmosphere Spectral Element Dycore

Here we propose to refactor the spectral element dynamical core (SE dycore) for atmospheric dynamics to be written in C++ and use libraries from the Trilinos suite (Heroux et al., 2005), including the Kokkos performance portable programming model (Edwards et al., 2014). This focused effort is directly aligned with Theme 3 of the proposal – leverage of computational science technologies – as well as Theme 2 – preparation for exascale computers.

The goal of this effort is to have an efficient and extensible library-based replacement for the SE dycore in three years. Our goal is to have the new code perform on existing platforms as well as the current SE dycore, or very nearly so (within 15%), while positioning it to efficiently utilize emerging platforms. The new code will have additional capabilities for solving ensembles of solutions concurrently as well as the ability to perform sensitivity analysis. The new code will have a single performance-portable implementation that will run well on all targeted machines, and be prepared to run efficiently on new architectures of interest to DOE. Most critically, the new code base will leverage the ASC- and ASCR-supported Trilinos and Kokkos projects.

What we learn from this endeavor will greatly influence the future of the ACME code, in terms of programming languages, programming models, and the ability to leverage computational science efforts at DOE.

We are cognizant of the opinion that climate component refactoring efforts, such as we propose here, do not lead to improved code because general libraries are not tailored to the climate problem and that computational scientists prefer to “throw their libraries over the fence” and not engage for the long term. We are also aware that this is an opinion based on experience. On the other hand, the PISCEES project and Aeras LDRD are demonstrations that mutually beneficial collaborations are possible, and a high-level goal of this effort is to show that this type of success can be replicated.

The CAM-SE Code

The SE dycore (Dennis et al., 2012) is a central component of the ACME model, solving for the fluid flow and transport of the atmosphere. The code uses explicit integration of the hydrostatic formulation, and a spectral element discretization in the horizontal dimensions with co-located nodes and integration points, leading to a diagonal mass matrix. To stabilize the scheme, a hyperviscosity $\tau \nabla^4$ term is added.

The SE dycore has been verified for numerous idealized test cases and benchmark problems (Dennis et al., 2005; Lauritzen et al., 2010; Taylor and Fournier, 2010), and many others. Also, parallel performance to very large processor counts (strong scaling) has been shown to be excellent (Dennis et al., 2005; Fournier et al., 2010). Work is underway to implement a non-hydrostatic version as well, which is generally needed when horizontal discretizations go below 10km.

On the performance side, there has been ongoing effort to achieve performance on threaded and GPU architectures. Most of the key kernels have been written to use OpenACC directives so they can run on the nVidia GPUs of the Titan machine at Oak Ridge Leadership Computing Facility (Norman et al., 2015). Separate kernels are compiled for threaded architectures, using OpenMP directives.

The Kokkos Programming Model

A key motivator of this refactor is to prepare the SE dycore code base for all future architectures in one pass. This is the promise, now backed by results, of the Kokkos programming model (Edwards et al., 2014). Kokkos, both part of Trilinos and available independently, is a library that uses standard C++ language constructs to create a programming model for on-node parallel, performance-portable code execution. (Kokkos' role is to exploit parallelism on a node – the inter-node parallelism is addressed separately through MPI, or potentially through the schedulers of asynchronous multi-task libraries). A code that uses Kokkos ends up with a single code base that is executed differently based on C++ template parameters.

To achieve performance on diverse architectures, Kokkos employs two key abstractions. The first is the launch of an on-node parallel operation, in which an outer loop can be executed concurrently. The second abstraction of Kokkos builds in the flexibility of the data layout of the arrays. The array access syntax for a Kokkos multi-dimensional array is independent of the actual layout in memory. So, memory layouts for data vectors can be changed without changing the code that uses them.

The Kokkos team now has considerable experience and results that show that the single Kokkos implementation can achieve results as good as results achieved when hand-coding the kernels for each specific programming model (e.g. CUDA, OpenMP) (Edwards et al., 2014).

Trilinos and Albany/FELIX

The plan to refactor the SE dycore using Trilinos is influenced by the notable success on the PISCEES SciDAC project in developing the Albany/FELIX ice sheet velocity solver (Tezaur et al., 2015a), as mentioned in Section 1.2. We were able to rapidly develop a fully implicit, unstructured-grid finite element code for the first-order approximation of the nonlinear Stokes equations. The code was born with flexibility in element shape and order, with automatic differentiation for Jacobians and sensitivities, and efficient MPI-based parallelism, achieving scalability to over *one billion* unknowns. Albany/FELIX goes beyond forward simulation, with an adjoint-based inversion capability and uncertainty-quantification methods. All this was made possible in such a short time period because of the leverage of the Trilinos suite and deep collaborations with computational scientists wielding their own libraries.

The Aeras LDRD

Another key piece of the groundwork for this refactoring task has been put in place by the investment of an SNL LDRD (Laboratory Directed Research and Development) project to develop the Aeras code (Spotz et al., 2015). Aeras has made progress on developing a spectral element atmosphere dycore using the same Trilinos-based Albany finite element code that was used for the ice sheet simulation. The team has successfully developed a 2D shallow-water model and is finalizing a 3D hydrostatic model.

There have been two key advances confirmed by the development of the Aeras prototype. The first is the use of embedded ensembles. Using C++ templates and the infrastructure already in Albany for automatic differentiation, we were able to overload all operations in the Aeras finite element assembly to work on an array of data instead of single values. For instance, we can compute an ensemble of 4 or 32 velocity field solutions by changing the data structures to have an additional dimension of that length, and by redefining all operations (e.g. $*$, $+$, \log) to work on an array of data. Since our code was already templated on the

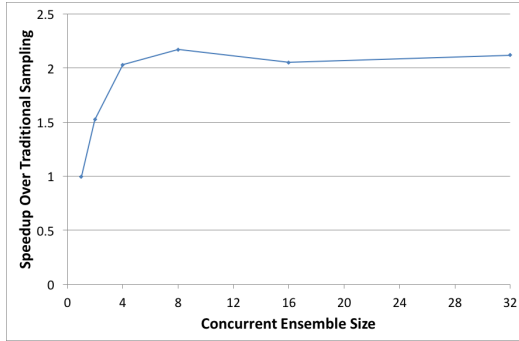


Figure 8: The speedup of calculating ensembles of solves as an embedded inner loop, as opposed to an outer loop, is shown for the shallow water equations in the Aeras code. With the use of C++ templating on the data type, the PDE assembly code does not need to be modified for this capability.

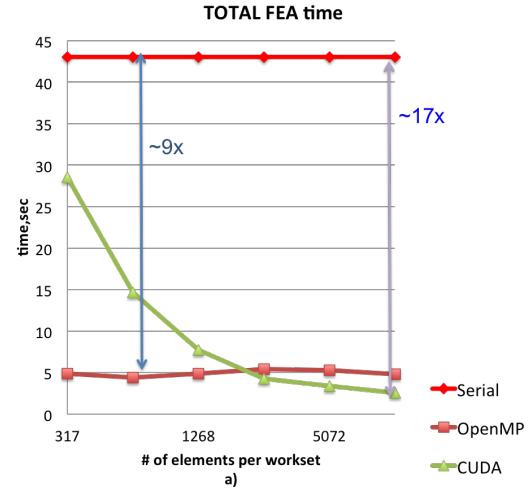


Figure 9: Relative compute time on 3 different compute architectures is shown for running the shallow-water equations in the Aeras code. With the use of the Kokkos programming model, a single code base is used for all three runs. The number of elements in an on-node parallel block is varied as the x-axis.

data type, there is absolutely no modification necessary to the computational kernels of the atmospheric physics needed to achieve this. There is just a seeding stage to put in the ensemble of inputs to the finite element assemble, and an extraction phase to pull out the ensemble of results. To accomplish this without the use of C++ templates to swap out the data type, one would need to undertake a large code refactoring to add an extra dimension for the ensemble length to all arrays, which would likely slow down the code for single-execution runs. As can be seen in Figure 8, using ensembles can give a $2\times$ speedup, even for small ensemble sizes.

The second advance seen in Aeras is the use of Kokkos. Using the programming model described above, a single code base was written that runs in serial mode, with OpenMP threads, and on NVIDIA GPUs using CUDA. To this point, we have demonstrated that the same code base can compile and run reasonably on all architectures but we have not yet done a detailed analysis of the absolute performance. A striking feature of our experience is that the GPU-based architecture needs a lot of work to achieve high performance. Figure 9 shows these results from the submitted paper by (Demeshko et al., 2016).

Plans for the SE Dycore Refactor:

Building on the experiences listed above, we are proposing a refactor of the SE dycore code to use C++, Trilinos libraries, embedded ensembles, and Kokkos for use in ACME. Our approach is a departure from that of the Aeras LDRD, which started with a Trilinos-based code and worked to adapt it to the atmospheric dynamics equations. Instead, we are proposing an **incremental in-place migration** from one code base to the other. We have decided that the costs of verification and performance tuning, which have been done extensively in the SE dycore, are too high to do in an independent code base.

By starting with the SE dycore Fortran code, we have a battery of tests to assure correctness of the code. We will work with the Unit Testing task team (Section 2.2) to expand coverage and reduce testing time. Our migration path will stay on the manifold of all tests passing. By taking on very small migration tasks and testing for correctness, the code migration process will move quickly.

The process of converting to C++, Kokkos data structures, and Kokkos dispatch of on-node parallel kernels will begin with key kernels of the code, such as the implementation of the divergence operator, that have no dependencies. Over time, we will move up the call stack, to the functions that call these kernels, and convert them. We will develop helpful infrastructure to replace kernel calls in Fortran with calls to

intermediary 'C' code that copy the data into Kokkos arrays, and use Kokkos to dispatch C++ kernels. By putting timers around the copies, and discounting this cost, we will be able to monitor the performance characteristics of the code. In the refactor, we will use a templated data type to enable the embedded ensemble capability, as well as sensitivity analysis with respect to parameters, and will put in a capacity to use automatic differentiation for implicit methods.

As the refactor progresses, the position of this interface between the languages, and the data copies that go with it, will migrate up the code stack. Only when we reach the layer of primary data allocations and replace them with Kokkos arrays will the costs of data copies disappear. The migration will convert the key dynamics and transport engines of the code. At this time, it is difficult to predict if we will have time to convert the extra pieces of code, such as post-processing of climate variables.

The End Game and Risk Mitigation:

The SE dycore refactor task is likely the highest-risk highest-reward task in the proposal. The potential rewards of having a performance portable implementation that will leverage the considerable ongoing investment in Kokkos from other funding sources to prepare for new architectures is significant. We believe the use of embedded ensembles may point the way to large gains, particularly on future architectures, which will be particularly significant in the calibration and tuning stage of the model development. The refactor also opens the door to the type of collaborations that we have seen on the Albany/FELIX project, in which library developers will choose this application to drive new developments and to demonstrate their newest algorithms. In this manner, the new code will be well prepared for successful SciDAC collaborations in the future.

At the end of the three-year project, the refactored code must be able be usable in ACME. The code must be verified and validated to give equivalent results, must be highly performant, and must have a path to being supported and improved in the future. We will periodically assess that the project is on track to meet these criteria of success.

Main Tasks of the SE Dycore Refactor:

1. Establish mixed-language C++ Fortran build, regression tests, and unit tests for key kernel.
2. Make choices on dimensions for Kokkos array orderings and templating of data type.
3. Migrate all kernels in transport and dynamics to new C++/Kokkos code base, ensuring that tests always pass.
4. Seize opportunities to leverage code base from Aeras project, Trilinos libraries, and other SciDAC projects. This will include the embedded ensembles capability.
5. When available, migrate non-hydrostatic code base into refactoring path.
6. Finish migration of code base to C++/Kokkos, including key step of data allocation.
7. Conduct more intensive investigations of data layout choices on performance, on architectures matching LCF platforms.
8. Demonstrate potential of embedded ensemble propagation and sensitivity analysis, facilitated by templating.
9. Finish documentation of redesigned code base.
10. Perform migration into ACME.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Andy Salinger (SNL) and has a total staffing level of 2.25 FTEs per year.

6 Proposed Research: Software Engineering Productivity Education

A repeated assertion in this proposal is that there are opportunities for improved productivity of code development in the service of scientific discovery. Above, we are proposing significant investments in the ACME code in support of this, in particular the build system testing infrastructure, a targeted verification effort, creation of models with idealized configurations, and three code modernization efforts. In addition to those focused efforts, we are here proposing a software engineering education element with broader, albeit shallower, impact.

Our plan is to partner with the IDEAS project, a unique effort from ASCR and BER, which has been focused on methods for improving the productivity of scientific software development (see IDEAS webpage <http://ideas-productivity.org>).

The first stage of this proposed effort will focus on discovery. We will assess the current habits of ACME developers to understand their current code development workflow and pain points. Our evaluation will begin with interviews and move to a questionnaire.

Based on what is learned in the discovery stage, we will curate online content on the ACME Confluence site that addresses the needs and areas of expressed interest. Since ACME is very heterogeneous in terms of coding experience and style, we are not expecting anything like a single curriculum. Rather, links to multiple series of videos or tutorials that address the key areas will be assembled. An abundance of remarkable content can be found online. As our audience consists of professionals with higher degrees, they are capable of setting their time and their own pace for learning. We feel this on-demand delivery is more appropriate for most content, but may require time at ACME all-hands meetings for tutorials if we find there is a broad need on a single topic.

Here are a few examples of training topics that may be in the curriculum:

- *How to use git: github forks, cherry-picking bug fixes to apply to multiple release branches, use of git “rerere”.*
- *What is Test Driven Development?*
- *How to do pair programming and when should you use it?*
- *How to use code coverage tools to improve your testing?*
- *What is cyclomatic complexity and is it a metric for you to use?*
- *What is Kanban, and how you can use it to improve your productivity?*
- *What are C++ templates and how can they reduce your maintained code base?*
- *How can IDEs like Eclipse improve your development cycle?*

The process of delivering educational content to improve the productivity of scientific software development will involve experimentation, feedback, and improvement. What we learn, and how successful this effort is, will be important data for the rest of the DOE science offices.

We expect that there will be many in ACME who are willing to take some of their time to improve their tools, processes, and skills in software development. However, we also expect that more incentive will be needed to reach the full audience. After we have conducted the initial discovery stage and assembled content, we will ask the ACME Council to help incentivize this professional development activity. One idea would be to require 4 hours of software engineering education each year to maintain write privileges to the ACME git repository.

We hope this small investment in education will have a broad and lasting impact on the culture of ACME software development. The main tasks for the educational component are:

1. Conduct the discovery phase and first iteration of educational program.
2. Institute ACME-wide educational program with targeted materials.

3. Work to make continuous software process improvement a part of ACME developer culture, and exchange ideas across DOE.

See Section 7 for more details on the timeline for deliverables and tasks. This work is led by Michael Heroux (SNL) and has a total staffing level of 0.3 FTEs per year.

7 Project Timeline, Deliverables, and Tasks

See full proposal for details

Full Proposal also includes sections on Project Management, Project Staffing, Data Management Plan, and Software Productivity and Sustainability Plan

8 Abbreviations and Code Names

ACME	Accelerated Climate Model for Energy (both code and project)
ACME v0	Version 0 of ACME model, 2014
ACME v1	Version 1.0 of ACME model, 2016
ACME v2	Version 2.0 of ACME model, 2018
ACME-A	ACME atmosphere model
ACME-SM	Accelerated Climate Model for Energy Software Modernization (this project)
Aeras	LDRD project at SNL on Next Generation Atmosphere Dycore
Albany	Open Source Finite Element code based on Trilinos, includes FELIX
ALCF	Argonne Leadership Computing Facility
Amanzi	Subsurface flow code supported by DOE
ANL	Argonne National Laboratory
API	Application programming interface
ARM	Atmospheric Radiation Measurement
ASC	Advanced Simulation and Computing program out of DOE's NNSA
ASCI	Advanced Strategic Computing Initiative
ASCR	Advanced Scientific Computing Research, DOE office of Science
ATDM-apps	Advanced Technology Development and Mitigation, part of ASC
BER	Biological and Environmental Research division of DOE Office of Science
BG/Q	IBM Blue Gene Q Supercomputer
CAM	Community Atmosphere Model
CAM-SE	Community Atmosphere Model Spectral Element
CAM4	Version 4 of CAM
CAM5	Version 5 of CAM
CASL-Vera	Nuclear Reactor code; Consortium for Advanced Simulation of Light Water Reactors
CDash	Software for displaying testing results on a web-based dashboard
CESM	Community Earth System Model
CIME	Common Infrastructure for Modeling the Earth
CLDICE	Cloud Ice
CLDLIQ	Cloud Liquid
CLUBB	Cloud Layers Unified by Binormals
CMake	Software tool for configuration and build of software
CMDV	Climate Model Development and Validation
CMDV-SE	CMDV Software Engineering: the funding source this proposal is aimed at
Confluence	Collaboration tool used by ACME for discussions and documentation
CouPE	Coupled Physics Environment
CPU	Central Processing Unit
CRM	Cloud-resolving model
CUBIT	Mesh generation tool out of SNL
CUDA	Programming model and libraries for running on GPUs
DCMIP	Dynamical Core Model Intercomparison Project
DOE	Department of Energy
ECHAM	Global Climate Model developed by the Max Planck Institute
ECP	Exascale Computing Program
EOF	Empirical Orthogonal Function
ESMF	Earth System Modeling Framework

F90	FORTRAN 90
FASTMath	Project on applied mathematics algorithms, tools, and software for HPC applications
FELIX	Finite Element for Land Ice eXperiments
FlashTest	Unit testing tool from ASC Flash Center
FLOPS	Floating-point Operations per Second
GASS	Global Atmospheric System Studies
GCM	Global Climate Model
GEV	Generalized Extreme Value
git	Open source version control software tool
github	Commercial web service for hosting git repositories
GPU	Graphics processing unit
GSMaP	Global Segment Map
HDF5	Hierarchical data format
HOMME	High-Order Method Modeling Environment atmosphere dynamics code
HTML	HyperText Markup Language
IDEAS	Interoperable Design of Extreme-scale Application Software project
KiD	Kinematic Driver
Kokkos	Open source library for performance portable code implementations
KS	Kolmogorov-Smirnov
LANDFRAC	Land fraction
LANL	Los Alamos National Laboratory
LBNL	Lawrence Berkeley National Laboratory
LCF	Leadership Computing Facilities, DOE
LDRD	Laboratory Directed Research and Development
LIVVkit	Land Ice Verification and Validation software
LLNL	Lawrence Livermore National Laboratory
MAM	Modal Aerosol Module
MAM3	MAM version with 3 modes
MAM4	MAM version with 4 modes
MCT	Modeling Coupling Toolkit
MG-1	Morrison-Gottelman microphysics
MG2	Morrison-Gottelman microphysics model, version 2
MMD	Maximum Mean Discrepancy
MOAB	Mesh Oriented datABase
MPAS	Model Predictions Across Scales
MPI	Message Passing Interface
NCAR	National Center for Atmospheric Research
NE30	Mesh resolution for atmosphere model, with 5400 elements
NERSC	National Energy Research Scientific Computing Center
NNSA	National Nuclear Security Administration, part of DOE
NUMICE	Cloud ice number concentrations
NUMLIQ	Cloud liquid number concentrations

ODE	Ordinary Differential Equations p.27
OPEN_ACC	Programming model for NVIDIA GPUs
OpenMP	Programming model for parallelism on a compute node.
ORNL	Oak Ridge National Laboratory
OS	Operating System
PDE	Partial Differential Equations
PETSc	Suite of scalable solution algorithms, from ANL
pFUnit	Unit test harness Fortran codes
Phi	Intel Phi processors
PISCEES	Predicting Ice Sheet and Climate Evolution at Extreme Scales
PNNL	Pacific Northwest National Laboratory
PS	Surface Pressure
RKHS	Reproducing Kernel Hilbert Space
RMS	Root mean square
RPI	Rensselaer Polytechnic Institute
RRTMG	Code for computing radiation
SciDAC	Scientific Discovery through Advanced Computing, DOE program
SCM	Single Column Model
SE	Spectral element
Sierra	Multiphysics finite element code from SNL, for ASC applications
SIGMA	Scalable Interfaces for Geometry and Mesh-based Applications
SNL	Sandia National Laboratory
Tempest	Remapping software for climate applications
Trilinos	Suite of scalable computational science libraries
UV-CDAT	Ultrасcale Visualization - Climate Data Analysis Tools
UWM	University of Wisconsin at Madison
WRF	Weather Research and Forecasting model
XML	eXtensible Markup Language
Zoltan	Open source software for load balancing and data ordering

9 Literature Cited

References

- Ahrens, J., B. Geveci, C. Law, C. Hansen, and C. Johnson, 2005: Paraview: An end-user tool for large-data visualization. *Citeseer*, 717–731 pp.
- Alexeev, Y., S. Mickelson, S. Leyffer, R. Jacob, and A. Craig, 2014: The heuristic static load-balancing algorithm applied to the community earth system model. *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, 1581–1590, doi:10.1109/IPDPSW.2014.177.
- Archibald, R. K., K. J. Evans, and A. Salinger, 2015: Accelerating Time Integration for Climate Modeling Using GPUs. *Proc. Comp. Sci*, **51**, 2046–2055.
- Baker, A. H., and Coauthors, 2015: A new ensemble-based consistency test for the community earth system model (pycect v1.0). *Geoscientific Model Development*, **8** (9), 2829–2840, doi:10.5194/gmd-8-2829-2015, URL <http://www.geosci-model-dev.net/8/2829/2015/>.
- Beljaars, A., P. Bechtold, M. Köhler, J. J. Morcrette, A. Tompkins, P. Viterbo, and N. Wedi, 2004: The numerics of physical parameterization. *Proc. ECMWF Workshop on Recent Developments in numerical methods for atmosphere and ocean modelling*, European Centre For Medium-Range Weather Forecasts, 113–135, 6–10 September 2004, Shinfield Park, Reading, U.K.
- Binkowski, F. S., and U. Shankar, 1995: The regional particulate matter model: 1. model description and preliminary results. *Journal of Geophysical Research: Atmospheres*, **100** (D12), 26 191–26 209, doi:10.1029/95JD02093, URL <http://dx.doi.org/10.1029/95JD02093>.
- Bogenschutz, P. A., A. Gettelman, H. Morrison, V. E. Larson, C. Craig, and D. P. Schanen, 2013: Higher-Order Turbulence Closure and Its Impact on Climate Simulations in the Community Atmosphere Model. *Journal of Climate*, **26** (23), 9655–9676, doi:10.1175/JCLI-D-13-00075.1, URL <http://journals.ametsoc.org/doi/abs/10.1175/JCLI-D-13-00075.1>.
- Bony, S., and J.-L. Dufresne, 2005: Marine boundary layer clouds at the heart of cloud feedback uncertainties in climate models. *J. Geophys. Res.*, **32** (20), L20 806, doi: 10.1029/2005GL023 851.
- Childs, H., and Coauthors, 2012: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, 357–372.
- Coles, S., 2001: *An Introduction to Statistical Modeling of Extreme Values*. Springer, London, UK.
- Craig, P., R. Jacob, B. Kauffman, T. Bettge, J. Larson, E. Ong, C. Ding, and H. He, 2005: Cpl6: The new extensible high-performance parallel coupler for the community climate system model. *Int. J. High Perf. Comp. App.*, **19** (3), 309–327.
- Craig, P., M. Vertenstein, and R. Jacob, 2012: A new flexible coupler for earth system modeling developed for ccsm4 and cesm1. *Int. J. High Perf. Comp. App.*, **26** (1), 31–42.
- Demeshko, I., O. Guba, R. P. Pawlowski, A. G. Salinger, W. F. Spitz, I. K. Tezaur, and M. A. Heroux, 2016: Towards performance-portability of the albany finite element analysis code using the kokkos library. *submitted*.
- Dennis, J., A. Fournier, W. Spitz, A. St.-Cyr, M. Taylor, S. Thomas, and H. Tufo, 2005: High resolution mesh convergence properties and parallel efficiency of a spectral element dynamical core. *Internat. J. High Perf. Comput. Appl.*, **19**, 225–235.

- Dennis, J., and Coauthors, 2012: CAM-SE: A scalable spectral element dynamical core for the Community Atmosphere Model. *Internat. J. High Perf. Comput. Appl.*, **26**, 74–89.
- Devine, K., E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, 2002: Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, **4** (2), 90–97.
- Edwards, H. C., C. R. Trott, and D. Sunderland, 2014: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Distributed Computing*, **74**, 3202–3216.
- Evans, K., D. Gardner, M. A. Taylor, R. Archibald, P. Worley, and C. Woodward, 2016: Accuracy analysis of a fully implicit solver within a hydrostatic spectral-element atmosphere model. *Mon. Wea. Rev.*, **in Prep.**
- Evans, K., D. Rouson, A. Salinger, M. Taylor, J. B. W. III, and W. Weijer, 2009: A scalable and adaptable solution framework for components of the Community Climate System Model. *Lecture Notes in Comp. Sci.*, **5545**, 332–341, doi:10.1007/978-3-642-01973-9.
- Flato, G., and Coauthors, 2013: *Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, chap. Evaluation of Climate Models. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, (Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex, and P.M. Midgley (eds.)).
- Fournier, A., M. A. Taylor, and J. J. Tribbia, 2010: The spectral element atmosphere model (SEAM): High-resolution parallel computation and localized resolution of regional dynamics. *Mon. Wea. Rev.*, **132**, 726–748.
- Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, 1988: Solving problems on concurrent computers. *Prentice-Hall, Englewood Cliffs, New Jersey*, **19**, 88.
- Gettelman, A., and H. Morrison, 2015: Advanced Two-Moment Bulk Microphysics for Global Models. Part I: Off-Line Tests and Comparison with Other Schemes. *Journal of Climate*, **28** (3), 1268–1287, doi: 10.1175/JCLI-D-14-00102.1, URL <http://journals.ametsoc.org/doi/abs/10.1175/JCLI-D-14-00102.1>.
- Gettelman, A., H. Morrison, and S. J. Ghan, 2008: A new two-moment bulk stratiform cloud microphysics scheme in the Community Atmosphere Model, version 3 (CAM3). Part II: Single-column and global results. *Journal of Climate*, **21**, 3660–3679, doi:10.1175/2008JCLI2116.1.
- Gettelman, A., H. Morrison, S. Santos, P. Bogenschutz, and P. M. Caldwell, 2015: Advanced Two-Moment Bulk Microphysics for Global Models. Part II: Global Model Solutions and Aerosol-Cloud Interactions. *Journal of Climate*, **28** (3), 1288–1307, URL <http://journals.ametsoc.org/doi/abs/10.1175/JCLI-D-14-00103.1>.
- Ghan, S., X. Liu, R. Easter, P. J. Rasch, J. Yoon, and B. Eaton, 2012: Toward a minimal representation of aerosols in climate models: Comparative decomposition of aerosol direct, semi-direct and indirect radiative forcing. *J. Climate*, doi:10.1175/JCLI-D-11-00650.1.
- Golaz, J.-C., V. E. Larson, and W. R. Cotton, 2002: A PDF-based model for boundary layer clouds. Part I: Method and model description. *Journal of the atmospheric sciences*, **59** (24), 3540–3551, URL [http://journals.ametsoc.org/doi/abs/10.1175/1520-0469\(2002\)059%3C3540:APBMFB%3E2.0.CO%3B2](http://journals.ametsoc.org/doi/abs/10.1175/1520-0469(2002)059%3C3540:APBMFB%3E2.0.CO%3B2).
- Gretton, A., K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, 2006: A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 513–520.

- Gustafson, W. I., P.-L. Ma, H. Xiao, B. Singh, P. J. Rasch, and J. D. Fast, 2013: The separate physics and dynamics experiment (spade) framework for determining resolution awareness: A case study of micro-physics. *Journal of Geophysical Research: Atmospheres*, **118** (16), 9258–9276, doi:10.1002/jgrd.50711, URL <http://dx.doi.org/10.1002/jgrd.50711>.
- Heroux, M., and Coauthors, 2005: An overview of the Trilinos project. *ACM Trans. Math. Soft.*, **31** (3), 397–423.
- Hodges, A., and Coauthors, 2001: ASCI Software Engineering: Goals, Principles and Guidelines. Tech. rep., Department of Energy.
- Hoinka, K. P., 1998: Statistics of the global tropopause pressure. *Mon. Wea. Rev.*, **126**, 3303–3325.
- IDEAS, 2016: The IDEAS Project. <http://ideas-productivity.org>.
- Jacob, R., J. Larson, and E. Ong, 2005: $M \times N$ communication and parallel interpolation in Community Climate System Model Version 3 using the model coupling toolkit. *International Journal of High Performance Computing Applications*, **19** (3), 293–307.
- Jain, R., and T. J. Tautges, 2014: Generating unstructured nuclear reactor core meshes in parallel. *Procedia Engineering*, **82**, 351–363.
- Jenkins, 2016, 2016: Jenkins open source software project. <https://jenkins.io>.
- Knuth, D. E., 1984: Literate programming. *The Computer Journal*, **27** (2), 97–111, doi:10.1093/comjnl/27.2.97.
- Larson, V. E., J.-C. Golaz, and W. R. Cotton, 2002: Small-scale and mesoscale variability in cloudy boundary layers: Joint probability density functions. *Journal of the atmospheric sciences*, **59** (24), 3519–3539, URL [http://journals.ametsoc.org/doi/abs/10.1175/1520-0469\(2002\)059%3C3519:SSAMVI%3E2.0.CO%3B2](http://journals.ametsoc.org/doi/abs/10.1175/1520-0469(2002)059%3C3519:SSAMVI%3E2.0.CO%3B2).
- Lauritzen, P. H., C. Jablonowski, M. A. Taylor, and R. D. Nair, 2010: Rotated versions of the jablonowski steady-state and baroclinic wave test cases: A dynamical core intercomparison. *Journal of Advances in Modeling Earth Systems*, **2**, Art. #15.
- Lebassi-Habtezion, B., and P. Caldwell, 2014: Aerosol specification in single-column cam5. *Geoscientific Model Development Discussions*, **7** (6), 7693–7731.
- Liu, X., P.-L. Ma, H. Wang, S. Tilmes, B. Singh, R. C. Easter, S. J. Ghan, and P. J. Rasch, 2016: Description and evaluation of a new four-mode version of the Modal Aerosol Module (MAM4) within version 5.3 of the Community Atmosphere Model. *Geoscientific Model Development*, **9** (2), 505–522, doi:10.5194/gmd-9-505-2016, URL <http://www.geosci-model-dev.net/9/505/2016/>.
- Liu, X., and Coauthors, 2012: Toward a minimal representation of aerosols in climate models: description and evaluation in the community atmosphere model CAM5. *Geoscientific Model Development*, **5** (3), 709–739, doi:10.5194/gmd-5-709-2012, URL <http://www.geosci-model-dev.net/5/709/2012/gmd-5-709-2012.html>.
- Mahadevan, V. S., E. Merzari, T. Tautges, R. Jain, A. Obabko, M. Smith, and P. Fischer, 2014: High-resolution coupled physics solvers for analysing fine-scale nuclear reactor design problems. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, **372** (2021), 20130381.

- Mahajan, S., K. J. Evans, M. Branstetter, V. Anantharaj, and J. K. Leifeld, 2015: Fidelity of precipitation extremes in high resolution global climate simulations. *Procedia Computer Science*, **51**, 2178–2187.
- Missile Defense Agency, 2008: Department of Defense Documentation of Verification, Validation & Accreditation (VV&A) for Models and Simulations. Tech. rep., Department of Defense.
- Morrison, H., and A. Gettelman, 2008: A new two-moment bulk stratiform cloud microphysics scheme in the NCAR Community Atmosphere model (CAM3), Part I: Description and numerical tests. *Journal of Climate*, **21**, 3642–3659, doi:10.1175/2008JCLI2105.1.
- Neale, R. B., and Coauthors, 2012: Description of the NCAR Community Atmosphere Model: CAM5.0. Tech. Rep. NCAR/TN-486+STR, National Center for Atmospheric Research, Boulder, Colorado, USA. 268 pp., URL = <http://www.cesm.ucar.edu/models/atm-cam>.
- Norman, M., , A. V. J. Larkin, and K. Evans, 2015: A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel, implicit climate dynamics. *J. Comp. Sci.*, **9**, 1–6.
- Norman, M. R., A. L. Gaddis, V. G. Anantharaj, K. J. Evans, S. Mahajan, and P. H. Worley, 2016: Computational benefits of an ensemble-based approach to climate modeling and testing at scale. *in preparation*.
- Park, S., C. S. Bretherton, and P. J. Rasch, 2014: Integrating cloud processes in the Community Atmosphere Model, version 5. *Journal of Climate*, **27**, 6821–6855, doi:10.1175/JCLI-D-14-00087.1.
- Pawlowski, R., E. Phipps, and A. Salinger, 2012a: Automating embedded analysis capabilities using template-based generic programming. *Scientific Programming*, 197–219.
- Pawlowski, R., E. Phipps, A. Salinger, S. Owen, C. Siefert, and M. Staten, 2012b: Applying template-based generic programming to the simulation and analysis of partial differential equations. *Scientific Programming*, 327–345.
- Phipps, E. T., M. D’Elia, H. C. Edwards, M. Hoemmen, J. J. Hu, and S. Rajamanickam, 2015: Embedded ensemble propagation for improving performance, portability and scalability of uncertainty quantification on emerging computational architectures. *CoRR*, **abs/1511.03703**, URL <http://arxiv.org/abs/1511.03703>.
- Randall, D., and Coauthors, 2003: Confronting models with data: The gewex cloud systems study. *Bulletin of the American Meteorological Society*, **84** (4), 455–469, doi:10.1175/BAMS-84-4-455.
- Rasch, P. J., and J. E. Kristjánsson, 1998: A comparison of the CCM3 model climate using diagnosed and predicted condensate parameterizations. *Journal of Climate*, **11**, 1587–1614, doi:10.1175/1520-0442(1998)011\$(\$1587:ACOTCM\$)\$2.0.CO;2.
- Ringler, T., 2015: private communication.
- Ringler, T., M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud, 2013: Maps-o. *Ocean Modelling*, **69** (C), 211–232.
- Rosenbaum, P. R., 2005: An exact distribution-free test comparing two multivariate distributions based on adjacency. *Journal of the Royal Statistical Society: Series B*, **67**(4), 515–530.
- Rosinski, J. M., and D. L. Williamson, 1997: The accumulation of rounding errors and port validation for global atmospheric models. *SIAM J. Sci. Comput.*, **18** (2), 552–564, doi:10.1137/S1064827594275534.
- Salinger, A. G., and Coauthors, 2013: Albany: A component-based partial differential equation code built on trinos. *Sandia National Labs Tech Report SAND2013-8430J*.

- Salvadori, G., and C. DeMichele, 2013: *Extremes in a changing climate*, chap. Multivariate extreme value methods. Cambridge University Press, Springer, a. AghaKouchak, D. Easterling, K. Hsu, S. Schubert, S. Sorooshian (eds).
- Sargent, A., and J. Fastook, 2010: Manufactured analytical solutions for isothermal full-stokes ice sheet models. *Cryosphere*, **4**, 285–311.
- Shipway, B. J., and A. A. Hill, 2012: Diagnosis of systematic differences between multiple parametrizations of warm rain microphysics using a kinematic framework. *Quarterly Journal of the Royal Meteorological Society*, **138** (669), 2196–2211, doi:10.1002/qj.1913, URL <http://dx.doi.org/10.1002/qj.1913>.
- SIGMA, 2016, 2016: Sigma: Scalable interfaces for geometry and mesh based applications. <http://sigma.mcs.anl.gov>.
- Sjaardema, G., and Coauthors, 1994: Cubit mesh generation environment volume 1: Users manual. Sandia National Laboratories.
- Spotz, W., T. Smith, I. Demeshko, and J. Fike, 2015: Aeras: a next generation global atmosphere model. *Procedia Computer Science*, **51**, 2097–2106, International Conference On Computational Science, ICCS 2015 Computational Science at the Gates of Nature.
- Székely, G. J., and M. L. Rizzo, 2004: Testing for equal distributions in high dimension. *InterStat*, **5**, 1–6.
- Tautges, T. J., and A. Caceres, 2009: Scalable parallel solution coupling for multiphysics reactor simulation. *Journal of Physics: Conference Series*, IOP Publishing, Vol. 180, 012017.
- Tautges, T. J., C. Ernst, C. Stimpson, R. J. Meyers, and K. Merkley, 2004: Moab: a mesh-oriented database. Tech. rep., Sandia National Laboratories.
- Tautges, T. J., J. A. Kraftcheck, N. Bertram, V. Sachdeva, and J. Magerlein, 2012: Mesh interface resolution and ghost exchange in a parallel mesh representation. *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, IEEE, 1670–1679.
- Taylor, M., and A. Fournier, 2010: A compatible and conservative finite element method on unstructured grids. *J. Comput. Phys.*, **229**, 5879–5895, doi:10.1007/978-3-642-01973-9.
- Tezaur, I., M. Perego, A. Salinger, R. Tuminaro, and S. Price, 2015a: Albany/FELIX: A parallel, scalable and robust finite element higher-order Stokes ice sheet solver built for advance analysis. *Geoscientific Model Development*, **8** (4), 1197–1220, URL <http://www.geosci-model-dev.net/8/1197/2015/>.
- Tezaur, I., R. Tuminaro, M. Perego, A. Salinger, and S. Price, 2015b: On the scalability of the Albany/FELIX first-order Stokes approximation ice sheet solver for large-scale simulations of the Greenland and Antarctic ice sheets. *Procedia Computer Science*, **51**, 2026–2035, doi:http://dx.doi.org/10.1016/j.procs.2015.05.467, URL <http://www.sciencedirect.com/science/article/pii/S1877050915012752>, international Conference On Computational Science, ICCS 2015 Computational Science at the Gates of Nature.
- Ullrich, P. A., and M. A. Taylor, 2015: Arbitrary-order conservative and consistent remapping and a theory of linear maps: Part i. *Monthly Weather Review*, **143** (6), 2419–2440, doi:10.1175/MWR-D-14-00343.1.
- Valcke, S., T. Craig, and L. Coquart, 2015: Oasis3-mct user guide. Tech. Rep. TR/CMGC/15/38, CER-FACS/CNRS.
- Valcke, S., and Coauthors, 2012: Coupling technologies for earth system modelling. *Geoscientific Model Development*, **5** (6), 1589–1596.

- Wan, H., P. J. Rasch, M. A. Taylor, and C. Jablonowski, 2015: Short-term time step convergence in a climate model. *Journal of Advances in Modeling Earth Systems*, **7** (1), 215–225, doi:10.1002/2014MS000368, URL <http://dx.doi.org/10.1002/2014MS000368>.
- Wan, H., P. J. Rasch, K. Zhang, J. Kazil, and L. R. Leung, 2013: Numerical issues associated with compensating and competing processes in climate models: an example from ECHAM-HAM. *Geoscientific Model Development*, **6** (3), 861–874, doi:10.5194/gmd-6-861-2013, URL <http://www.geosci-model-dev.net/6/861/2013/>.
- Wan, H., P. J. Rasch, K. Zhang, Y. Qian, H. Yan, and C. Zhao, 2014: Short ensembles: an efficient method for discerning climate-relevant sensitivities in atmospheric general circulation models. *Geosci. Model Dev.*, **7** (2), 1961–1977, doi:10.5194/gmd-7-1961-2014.
- Whitby, E. R., and P. H. McMurry, 1997: Modal aerosol dynamics modeling. *Aerosol Science and Technology*, **27** (6), 673–688, doi:10.1080/02786829708965504, URL <http://dx.doi.org/10.1080/02786829708965504>.
- Williamson, D., 1999: Convergence of atmospheric simulations with increasing horizontal resolution and fixed forcing scales. *Tellus A*, **51** (5), 663–673, URL <http://www.tellusa.net/index.php/tellusa/article/view/14485>.
- Xie, S., and Coauthors, 2010: Clouds and more: Arm climate modeling best estimate data. *Bulletin of the American Meteorological Society*, **91** (1), 13–20, doi:10.1175/2009BAMS2891.1, URL <http://dx.doi.org/10.1175/2009BAMS2891.1>, <http://dx.doi.org/10.1175/2009BAMS2891.1>.
- Yan, M., K. Jordan, D. Kaushik, M. Perrone, V. Sachdeva, T. J. Tautges, and J. Magerlein, 2012: Coupling a basin modeling and a seismic code using MOAB. *Procedia Computer Science*, **9**, 986–993.