# Climate Model Quality Assurance through Consistency Testing and Error Source Identification

by

**Daniel J. Milroy**

B.A., University of Chicago, 2006

M.S., University of Colorado Boulder, 2015

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2018

This thesis entitled:
Climate Model Quality Assurance through Consistency Testing and Error Source Identification
written by Daniel J. Milroy
has been approved for the Department of Computer Science

Professor Elizabeth R. Jessup

Allison H. Baker

Dorit M. Hammerling

Thomas Hauser

Abtin Rahimian

Date

Milroy, Daniel J. (Ph.D., Computer Science)

Climate Model Quality Assurance through Consistency Testing and Error Source Identification

Thesis directed by Professor Elizabeth R. Jessup

The Community Earth System Model (CESM$^{\text{TM}}$) is a global climate model whose simulations have significant impact on social policy. The CESM is large and complex, consisting of about 1.5 million lines of code and several coupled component models. Quality assurance is necessary for the continued development and improvement of the CESM. To address this need, the CESM Ensemble Consistency Test (CESM-ECT) was developed as a statistical test for consistency between experimental outputs and an accepted ensemble. The CESM-ECT provides rapid feedback to model developers, scientists, and end users without expert knowledge of climate science. In this work, we investigate the properties and composition of the CESM-ECT ensemble, resulting in an improved test. We expand the CESM-ECT by creating an "ultra-fast" test at the ninth CESM time step and demonstrate that the test can detect statistical inconsistency in multiple CESM component models. Equipped with refined tests, we focus on locating sources of statistical inconsistency. Our approach uses graph analysis to find relevant segments of CESM code, and we propose an iterative instrumentation method for converging on the sources of inconsistency. We discuss the modification and performance optimization of a central atmospheric microphysics package with the assistance of our newly developed tools. We conclude by advancing a strategy to realize our iterative instrumentation process, which will identify specific code lines leading to statistical inconsistency. The framework created in this thesis will enable full-featured quality assurance for the CESM through consistency testing and error source identification.

## Dedication

To the memory of my uncle, Maurice Blackmon, whose faith in me formed a foundation for my graduate studies. To my aunt, Marilyn Hughes Blackmon, whose continuous support and shared curiosity has helped me build a career. To my mother, without which I could not be.

## Acknowledgements

I am deeply grateful to Allison Baker for patiently helping me organize and refine my ideas and clarify my writing. I am indebted to Dorit Hammerling for accommodating instruction in statistics, and for incisive guidance in interpreting and representing data. They devoted a tremendous amount of time to my education; I hope my work may prove their time well spent. I wish to thank Elizabeth Jessup and Thomas Hauser for making my assistantship possible, and for their counsel during the PhD program.

I extend my gratitude to the scientists and engineers at NCAR who welcomed my presence and generously provided insight into Earth System Modeling and scientific software development: John Dennis, Brian Dobbins, Kevin Paul, and Haiying Xu. Finally, special thanks are due my coauthors Youngsung Kim and Sheri Mickelson, who made valuable contributions to my work.

## Contents

# Tables

## Figures

**Figure**

# Chapter 1

# Introduction

Earth System Models (ESMs) simulate the global environment through the mathematical modeling of its component ecosystems (e.g., the oceans, atmosphere, land, sea ice, river systems) and their interactions. Improving the accuracy of ESM predictions through better understanding of physical earth systems and their interactions in turn promotes deeper insight into the physical systems themselves. Advancing this positive feedback is increasingly important as global climate change accelerates. Understanding anthropogenic forcings (e.g. deforestation and $CO_2$ emissions) and their impacts on natural processes through simulation will become more essential to assess and prepare for future climate scenarios. Indeed, such simulations already form an integral part of international evaluations of climate change, such as the Intergovernmental Panel on Climate Change (IPCC, 2013). Given the relevance of ESMs to society and their influence on policy, it is of utmost importance to ensure confidence in the integrity of their output.

ESMs present a difficult task for typical verification and validation strategies due to their complex and lengthy code and the varied machine environments where they are run (e.g., Clune et al., 2011, Easterbrook et al., 2009). Quantifying difference in output is a deceptively hard task. A standard, straightforward test is bit-for-bit (BFB) equivalence; however, a BFB test places excessive constraints on ESM software development. For example, cost saving modifications such as code optimization and algorithmic improvement often change floating-point values, yielding non-BFB results. Furthermore, different compilers (or the same compiler with different options) transform identical source code into different instructions. Changes that result in non-BFB output can represent the same climate state and allow the same scientific conclusions to be drawn. Baker et al., 2015 replace BFB equivalence by statistical distinguishability from an ensemble for the Community Earth System Model (CESM$^{\text{TM}}$; Hurrell et al., 2013), an open source ESM developed principally at the National Center for Atmospheric Research (NCAR). The test developed in Baker et al., 2015 is known as the CESM Ensemble Consistency Test (CESM-ECT), which partly through the efforts described in this work has been expanded into a suite of testing tools.

CESM-ECT was quickly incorporated into the CESM software development cycle and employed by software engineers and end users. The early success of the test justified research into its improvement, as some limitations were encountered. The first version of the CESM-ECT ensemble was generated from a single source of variability in the form of perturbations to the initial atmospheric temperature field and used a size of 151 ensemble members of 12-month simulations. Investigating the impact of including other forms of variability (such as that introduced by different compilers) into the ensemble, as well as studying the relationship between its size and classification power were clear research opportunities. Although Baker et al., 2015 studied the test's ability to classify many experimental modifications (such as different compilers and parameter changes) with known or expected outcomes, source code optimizations and other minor code modifications that were expected to be consistent were not studied. In addition, the first version of CESM-ECT was met with skepticism due to its use of relatively short, 12-month simulations (as compared to the multi-century runs that were standard at the time). By proving itself useful, the test based on an ensemble of 12-month runs dispelled doubt. However, considerable cost could be saved by testing consistency even earlier in the model runtime. These limitations provided natural opportunities for study that influenced the beginnings of this work. Finally, the most significant opportunity afforded by the creation of the CESM-ECT is the extension of statistical consistency testing to error source identification, where the causes of inconsistency can be traced back to their roots in the model code or execution environment. In other words, the CESM-ECT indicates the existence of an issue with the CESM simulation and then presents a probable cause. Indeed, the overarching goal of this work is to expand CESM-ECT into a full spectrum test and analysis framework for climate scientists and software engineers. In this introductory chapter, we provide relevant background information for this thesis including details of the CESM, early climate model verification and validation methods, and specifics of the CESM-ECT. We conclude this chapter with a brief overview of the following chapters.

## 1.1 The CESM

The Community Earth System Model (CESM) was created with the goal of simulating the evolution of the Earth's climate. While there is no single definition of an Earth System Model (ESM); at a minimum, the model must have the "ability of terrestrial ecosystems and the ocean to remove carbon dioxide from the atmosphere" (Hurrell et al., 2013), which amounts to necessitating a carbon cycle component (Flato, 2011). In fact, the CESM contains many physical modeling capabilities: "interactive carbon-nitrogen cycling, global dynamic vegetation and land use change due to human activity, a marine ecosystem-biogeochemical module, and new chemical and physical processes to study both the direct and indirect effects of aerosols on climate," as well as capability to model the atmosphere up to the thermosphere (Hurrell et al., 2013). As a modeling application, the CESM connects its various component models (whole atmosphere, atmospheric chemistry, land, sea ice, ocean, biogeochemistry, river, wave, and land ice) in a hub-and-spoke architecture via a central coupler (Craig et al., 2012). The encapsulation of components allows each submodel to use a different grid resolution and enables a vast number of possible configurations. The CESM offers hundreds of standard component sets with common experimental configurations, and permits component set configuration customization. Moreover, parameters and initial condition perturbations can be set per model, and different dynamical core solvers (finite volume and spectral element) can be chosen. The coupling frequency can be tuned, and subcycling (e.g., ratio of physics to dynamics time steps) can be used. Data output frequencies as well as output formats (instantaneous and time-averaged values) can be specified per model.

The CESM is one of many ESMs participating in the Coupled Model Intercomparison Project (CMIP phase 6), which informs the Intergovernmental Panel on Climate Change (IPCC) assessments (Eyring et al., 2016). CESM is one of the most expansive and encompassing ESMs contributing to the IPCC, and contains over one and a half million lines of Fortran source code. A code base of this size presents a substantial challenge for verification and Quality Assurance (QA) strategies. Furthermore, the CESM is a very popular ESM and is used by scientists around the world. The community aspect of the model enhances its popularity, and allows distributed model development through numerous contributors. While community development can certainly be a positive, it introduces complexity in the form of different coding practices, base software packages (e.g., compilers and numerical libraries) and runtime environments. Providing QA for a model of such flexibility and disparate uses presents a unique challenge that has not been addressed comprehensively by previous methods.

## 1.2 Legacy Verification Methods for ESMs

Easterbrook et al., 2009 provide a detailed overview of identifying incorrect implementations of physical phenomena and algorithms in ESMs via "Verification and Validation" strategies, focusing on the BFB test. BFB tests performed over short simulation time spans can be indicators of reproducibility over longer time intervals and signal errors in formula implementation (Easterbrook et al., 2009). Clune et al., 2011 describe the challenges of assessing the software quality of ESMs, highlighting the extraordinary difficulty of the undertaking. Even a CESM code base that correctly represents the evolution of the global climate inherits the model's high cyclomatic complexity (the number of independent paths through a subprogram; McCabe, 1976) which translates to a very high risk of containing bugs (Méndez et al., 2014). Many elements of the time evolution of a climate are highly nonlinear, and changing or substituting modules that represent different physical processes can cause an otherwise identical set of simulations to diverge from or converge to a valid true climate. Any change in CESM code, hardware, compiler version, or the supporting software stack can alter the simulation output at least to the magnitude of truncation or round-off errors. The challenge is to distinguish non-BFB results that are consistent and a consequence of a valid change from non-BFB outputs that result from error.

PerGro was a highly successful model validation test developed in Rosinski et al., 1997 for the Community Climate Model version 2 (CCM2; a predecessor of CESM) that did not presuppose BFB results. The intention of PerGro was to compare the time evolution of the Root Mean Square (RMS) atmospheric temperature difference between a perturbed and unperturbed simulation on one machine (control) to the difference between the time evolution after porting CCM2 to another system and one control run. The differences were examined for a model time interval of two days to include important phases of the simulation (such as dynamics and parameterizations). Rosinski et al., 1997 note that port validation based on initial model behavior is challenging because: "[v]alidation of these ports after a few time steps can be difficult because of the growth of differences that are introduced at the magnitude of machine rounding [...] [t]hese include, but are not limited to, a different computational order as parsed by compilers; differences in the intrinsic library functions (e.g., sin, sqrt, alog); and differences in the internal floating point representation of the respective machines." Further, the RMS temperature difference grows extremely rapidly in the first few time steps, which is driven by "dynamical processes and physical parameterizations" (Rosinski et al., 1997).

PerGro was used successfully for validating ports, environment changes, and model changes to several descendants of CCM2. However, when the atmospheric model component, the Community Atmosphere Model (CAM), was upgraded from CAM4 to CAM5, new parameterizations in CAM5 introduced such rapid growth of temperature perturbations that PerGro was unable to distinguish between perturbation growth and differences resulting from the change to be validated. Due in part to the inclusion of CAM5 in CESM, porting and validation became a costly process that involved running a simulation of 400 model years. The output data were then compared to an identical 400 year run on a trusted machine, and the results were run through a diagnostics package and given to a senior scientist for approval, which was a time-consuming and largely subjective process. To address the need for greater objectivity in model quality assurance at a reduced cost, Baker et al., 2015 developed an ensemble-based test that considers the behavior of all CAM variables (not merely temperature) and relationships between them after a single model year.

## 1.3 Ensemble-Based Consistency Testing

The CESM-Ensemble Consistency Test (ECT) is a suite of tools that determines statistical consistency by analyzing 12-month output from two different component models within CESM: CAM, and the Parallel Ocean Program (POP), with ensemble consistency testing tools referred to as CAM-ECT (created in Baker et al., 2015) and POP-ECT (created in Baker et al., 2016), respectively. While both tests employ ensemble methods, the underlying algorithms are different which reflects the varied requirements for characterizing variability in the respective component models. We focus on CAM-ECT as it is central to the investigations in this work.

Statistical consistency in the context of CESM-ECT signifies that experimental runs are statistically indistinguishable from the ensemble. Ensemble methods are common in climate science, but they are typically used for prediction. Conceptually, the CESM-ECT ensemble embodies the internal variability of the climate model, and the test determines whether a set of new runs falls within the quantified climate variability. Ensuring that the ensemble represents the right amount of variability is essential, as this in turn sets a threshold for the acceptable variability of the experimental runs. The test provides an objective measure of difference without resorting to excessively strict metrics like BFB results. Moreover, the output of the test does not require expert knowledge of climate science and returns a user-friendly *Pass* or *Fail*. The test's rapid feedback and ease of use significantly accelerate the software development cycle. In addition, the CESM-ECT software suite is modular and written in Python, so it is easy to extend and portable.

The CAM-ECT ensemble consists of 151 one-year (annually averaged) simulation outputs which differ by $\mathcal{O}(10^{-14})$K perturbations to the initial atmospheric temperature field. From a cost (and time) perspective, this is a tremendous improvement over the previous method of running a single 400 year simulation. Each single year simulation can be run independently, and the ease of use allows the ensembles to be generated by software engineers. The area-weighted global means of CAM variables are standardized by subtracting the mean and scaling by the standard deviation, and projected into a lower dimensional space by principal component analysis (PCA). Since many CAM variables are linearly dependent, PCA is used to characterize ensemble variability as it decorrelates linearly dependent variables. As an additional benefit, PCA allows the test to pick up changes in relationships between variables, since the Principal Components (PCs) are linear combinations of variables. After being subject to some modification, a set of experimental outputs (three by default) is standardized by the means and standard deviations of the ensemble variables, and projected into the PC space. The PC scores of the experimental set are compared to those of the ensemble and marked as failures if they exceed a desired threshold (see Baker et al., 2015 for default confidence intervals and other parameters). If more than a specified number of experimental PCs fail, the experimental set is considered statistically inconsistent with the ensemble.

POP-ECT was developed to determine statistical consistency for the POP component model of CESM (Baker et al., 2016). In POP output data, the spatial variation and temporal scales are much different than in CAM outputs, and there are many fewer variables. Hence the test does not involve PCA or spatial averages, and performs comparisons at each grid location. We focus on CAM-ECT rather than POP-ECT because the rapid propagation of perturbations in the model atmosphere allows for quick detection of changes.

## 1.4 Thesis Overview

The overarching goal of this work is to provide climate scientists with the means and tools to ensure consistent simulation output in modern computing environments when BFB equivalence is impractical (or even impossible). Our particular focus is on extending the robustness and utility of the test for statistical distinguishability, or consistency, introduced in Baker et al., 2015. Because the CESM-ECT evaluates whether a set of experimental outputs is consistent with a collection of trusted runs (the ensemble), the ensemble composition is crucial as the variability of the ensemble defines the permissible spread. Therefore, in Chapter 2, we study ensemble size and sources of variability to improve the test's ability to correctly classify experimental outputs expected to be indistinguishable from the ensemble. Further, to accelerate QA testing and improve the detection of localized inconsistencies, we develop an addition to the CESM-ECT suite in Chapter 3, which evaluates consistency based on only nine model time steps. The "ultra-fast" ECT is significantly faster than the original test, and reduces the CPU time by a factor of over 70. The substantial cost savings allow us to pursue our goal of identifying the reason why a given set of outputs is found to be statistically inconsistent, in order to provide useful feedback to CESM developers. In particular, we aim to trace differences in CESM output variables back to causative code sections; we describe our progress toward this goal in Chapter 4. In Chapter 5 we describe an application of our tools to assist in re-engineering a large CAM microphysics kernel to run in single precision. We discuss future work and make concluding remarks Chapter 6.

# Chapter 2

# Analyzing Ensemble Composition for Consistency Testing

The content of this chapter is contained in the refereed conference proceedings of ICCS (Milroy et al., 2016). The included text is verbatim with the exception of hyperlinked references to sections, which have different enumeration by necessity. All figures appear here unaltered.

## 2.1 Introduction

Modeling the Earth's climate is a formidable task. Earth System Models (ESMs) can be millions of lines of code and represent decades of software development time and climate research. The complexity of ESMs challenges standard verification and validation strategies (e.g., Clune et al., 2011, Easterbrook et al., 2009). Ensuring software quality is difficult due to the variety of platforms on which ESMs run, the vast number of parameters and configurations, and the ongoing state of development (e.g., Pipitone et al., 2012). Therefore, while achieving bit-for-bit (BFB) identical results may be desirable in general (e.g., Stodden et al., 2013) and may facilitate error detection, achieving BFB climate simulation results is difficult (if not impossible) and impedes efforts to improve performance. In particular, a change in ESM code, hardware, compiler version, or the supporting software stack can alter the simulation output at least as much as round-off errors. Further, small perturbations to initial conditions can produce non-BFB results, despite being representations of the same climate. A method to determine whether the same mean climate is represented by non-BFB results allows for the use of more aggressive code optimizations and heterogeneous execution environments.

We focus on output data from the Community Earth System Model (CESM) Hurrell et al., 2013, an open source ESM that is well used by the global climate research community and principally developed at the National Center for Atmospheric Research (NCAR). Motivated by the need for a simple and objective tool for CESM users and developers to determine whether non-BFB CESM outputs represented the same climate state, Baker, Hammerling, et al. recently developed the CESM ensemble consistency test (CESM-ECT) Baker et al., 2015. The idea behind CESM-ECT is to determine objective statistical consistency by comparing a new non-BFB CESM output (e.g. from a new machine) to an ensemble of simulation outputs from the original or “accepted” configuration (e.g. a trusted machine, software stack, etc.). CESM-ECT issues a pass for the newly generated output only if it is statistically indistinguishable from the ensemble’s distribution. The selection of a representative or “accepted” ensemble (and the variability it characterizes) is critical to CESM-ECT’s determination of whether new simulations pass. In Baker et al., 2015, ensemble variability is created by roundoff-level perturbations to the initial temperature field. However, as the goal in Baker et al., 2015 is the introduction of the ensemble consistency testing methodology, the important question of the ensemble composition is not addressed.

Our goal in this chapter is to ensure that the CESM-ECT ensemble composition is adequate for characterizing the variability of a consistent climate. Specifically, we investigate whether the ensemble variability induced by initial temperature perturbations is sufficient to capture legitimate minimal code modifications, such as mathematically equivalent reformulations or an alternative CESM-supported compiler. Note that while initial temperature perturbations are a typical way for climate scientists to gauge model variability, the effects of more general code or compiler modifications (i.e. not climate-specific) have hardly been studied. Perhaps the most relevant work is in He et al., 2001, where global summation orders are modified with noticeable impact on climate simulation results. However, the aim in He et al., 2001 is to improve or even achieve reproducibility (via alternative algorithms or increased precision). In this study, we improve upon the work in Baker et al., 2015 and make three principal contributions: we demonstrate the measurable effect of minimal code changes on CESM simulation output; we demonstrate that the variability induced by perturbations to initial temperature conditions in CESM does not sufficiently capture that induced by minimal code alterations and compiler changes; and we propose an alternative ensemble composition for CAM-ECT that improves the tool's accuracy and broadens its applicability.

This chapter is organized as follows. In Sect. 2.2, we provide background information on CAM-ECT and describe our experimental setup and tools. In Sect. 2.3, we present a series of code modifications that represent plausible, mathematically identical and stable alternatives to the original. Experimental results are given in Sect. 2.4, and an alternative composition is proposed in Sect. 2.5. We demonstrate the utility of the new ensemble composition in Sect. 2.6.

## 2.2 Preliminaries

The first CESM-ECT component (CAM-ECT) developed in Baker et al., 2015 focuses on data from CAM due to its relatively short time scales for propagation of perturbations. CAM output data containing annual averages at each grid point for the atmosphere variables are written in time slices to NetCDF history files in single-precision floating point format. The CAM-ECT ensemble consists of CAM output data for 151 simulations of 1-year in length created on a trusted machine with a trusted version of the CESM software stack. We generate the CESM results in this chapter on a 1° global grid using the CAM5 model version described in Kay et al., 2015. Simulations are run with 900 MPI tasks and two OpenMP threads per task on the Yellowstone machine at NCAR. The iDataPlex cluster is composed of 4,536 dx360 M4 compute nodes, featuring two Xeon E5-2670 Sandy Bridge CPUs and 32 GB memory per node, and FDR InfiniBand interconnects. The default compiler on Yellowstone for our CESM version is Intel 13.1.2 with -O2 optimization; GNU 4.8.0 and PGI 13.0 are also CESM-supported compilers for this version and are used throughout this study.

To thoroughly assess the appropriateness of the ensemble composition, we perform more exhaustive testing than was done in Baker et al., 2015, where most case studies involved the minimum $N_{test} = 3$ simulations runs for pyCECT (yielding a single pass or fail result). For our experiments we run at least 30 total simulations ($N_{tot} = 30$) and obtain pyCECT results equal to the number of ways $N_{test}$ simulations can be chosen from all $N_{tot}$ simulations (i.e., the binomial coefficient: $\binom{N_{tot}}{N_{test}}$). For example, $N_{tot} = 30$ and $N_{test} = 3$ yields 4060 possible combinations (and 4060 pyCECT results), which allows us to make a comprehensive comparison between CAM-ECT's false positive rate and the number of failures out of 4060. If an experiment's failure rate is approximately equal to the false positive rate then we say the experiment is statistically consistent with the ensemble. Testing all combinations in this manner would be prohibitively expensive with pyCECT, which was designed for a single test. Thus we developed a computationally efficient script (Ensemble Exhaustive Test: EET) to perform all $\binom{N_{tot}}{3}$ tests, rendering exhaustive testing both feasible and fast. Indeed, computing all 4060 results for $N_{tot} = 30$ takes less than one second, and 562,475 results for $N_{tot} = 151$ takes less than two seconds.

## 2.3 Code modifications

In this section we define the "minimal" code changes that should produce the same climate when evaluated by CAM-ECT. These minimal changes affect few lines of code and are mathematically equivalent and stable. Code changes potentially have a large impact because of the nonlinear chaotic climate model, but provided we have avoided numerically unstable code and catastrophic cancellation (such as described in Bailey, 2008) they should still produce the same climate. The five Fortran 90 code change experiments presented here all result in a difference in single precision output. They are illustrative of the complete set of CAM code modifications we performed, which is not shown in its entirety for the sake of brevity. For each experiment we ran 30 simulations, differing by a perturbation to the initial CAM temperature field. We examine two categories of modifications: those representing different coding styles and those with minor changes for optimization. Note that these code modifications were all done manually (not compiler-induced).

### 2.3.1 Modifications representing different coding styles

The following code modifications are mathematically equivalent formulations which could arise from two software engineers solving the same problem in different ways. These examples are from subroutines in the semi-implicit primitive equation module (*prim_si_mod.F90*) in CAM.

***Combine*** **(C)** is a single line code change to the *preq_omega_ps* subroutine:

*Original*:

```
ckk = 0.5d0/p(i,j,1)
term = divdp(i,j,1)
omega_p(i,j,1) = vgrad_p(i,j,1)/p(i,j,1)
omega_p(i,j,1) = omega_p(i,j,1) - ckk*term
```

*Modified*:

```
ckk = 0.5d0/p(i,j,1)
term = divdp(i,j,1)
omega_p(i,j,1) = (vgrad_p(i,j,1) - 0.5d0*divdp(i,j,1))/p(i,j,1)
```

Note that the difference in single and double precision output is not due to a catastrophic cancellation of `vgrad_p(i,j,1)` and `0.5d0*divdp(i,j,1)`; this difference is not present in the original code block.

***Expand*** **(E)** is a modification to the *preq_hydrostatic* subroutine. We expand the calculation of the variable `phi`:

*Original*:

```
phi(i,j,1) = phis(i,j) + phii(i,j,2) + Rgas*T_v(i,j,1)*hkk
```

*Modified*:

```
tt_real = Rgas*T_v(i,j,1)
phi(i,j,1) = tt_real*hkk + phis(i,j) + phii(i,j,2)
```

### 2.3.2 Modifications representing optimization strategies

The code changes in this subsection target improving the performance of existing code via rearranging the mathematical expressions.

***Division-to-multiplication* (DM)**: The original version of the *euler_step* subroutine of the primitive trace advection module (*prim_advection_mod.F90*) includes an operation that divides by a spherical mass matrix `spheremp`. The modification to this kernel consists of declaring a temporary variable (`tmpsphere`) defined as the inverse of `spheremp`, and substituting a multiplication for the more expensive division operation.

*Original*:

```
do k = 1 , nlev
  . . .
  do q = 1 , qsize
    qtens_biharmonic(:,:,k,q,ie) = &
    -rhs_viss*dt*nu_q*dp0*Qtens_biharmonic(:,:,k,q,ie) / elem(ie)%spheremp
        (:,:)
```

*Modified*:

```
tmpsphere(:,:) = 1.D0/elem(ie)%spheremp(:,:)
do k = 1 , nlev
  . . .
  do q = 1 , qsize
    qtens_biharmonic(:,:,k,q,ie) = &
    -rhs_viss*dt*nu_q*dp0*Qtens_biharmonic(:,:,k,q,ie) * tmpsphere(:,:)
```

***Unpack-order* (UO)** changes the order that an MPI receive buffer is unpacked in the *edgeVunpack* subroutine of *edge_mod.F90*. Changing the order of buffer unpacking has implications for performance, as traversing the buffer sub-optimally can prevent cache prefetching.

*Original*:

```
do k=1,vlyr
  do i=1,np
    v(i,1,k)  = v(i,1,k)+edge%buf(kptr+k,is+i) !South
    v(np,i,k) = v(np,i,k)+edge%buf(kptr+k,ie+i) !East
```

```
      v(i,np,k) = v(i,np,k)+edge%buf(kptr+k,in+i) !North
      v(1,i,k)  = v(1,i,k)+edge%buf(kptr+k,iw+i) !West
    end do
  end do
```

*Modified*:

```
  do k=1,vlyr
    do i=1,np !South
      v(i,1,k) = v(i,1,k)+edge%buf(kptr+k,is+i)
    end do
    do i=1,np !West
      v(1,i,k) = v(1,i k)+edge%buf(kptr+k,iw+i)
    end do
    do i=1,np !East
      v(np,i,k) = v(np,i,k)+edge%buf(kptr+k,ie+i)
    end do
    do i=1,np !North
      v(i ,np,k) = v(i,np,k)+edge%buf(kptr+k,in+i)
    end do
  end do
```

***Precision*** **(P)** is a performance-oriented modification to the water vapor saturation module (*wv_sat_methods.F90*) which tests whether recasting a subroutine to perform single-precision floating-point arithmetic results in a consistent climate. From a performance perspective this could be extremely advantageous and could present an opportunity for co-processor acceleration due to superior single-precision computation speed. We modify the elemental function that computes saturation vapor pressure by substituting `r4` for `r8` and casting to single-precision in the original:

*Modified*:

```
  elemental function GoffGratch_svp_water_r4(t) result(es)
  real(r8), intent(in) :: t  ! Temperature in Kelvin
```

```
real(r4) :: es, t4, tboil4   ! SVP in Pa
t4 = real(t)
tboil4 = real(tboil)
es = 10._r4**(-7.90298_r4*(tboil4/t4-1._r4)+ &
    5.02808_r4*log10(tboil4/t4)- &
    1.3816e-7_r4*(10._r4**(11.344_r4*(1._r4-t4/tboil4))-1._r4)+ &
    8.1328e-3_r4*(10._r4**(-3.49149_r4*(tboil4/t4-1._r4))-1._r4)+ &
    log10(1013.246_r4))*100._r4
```

## 2.4 Ensemble consistency testing results

In this section we test whether the ensemble distribution suggested in Baker et al., 2015 contains enough variability to capture our code modifications and optimizations. We do not address the causes of test result differences between changes at this time. We also examine the response of CAM-ECT to inter-compiler testing, thus testing the equivalence of code modifications to compilers as sources of variability. We begin with three size 151 ensembles generated by perturbing the initial temperature field on Yellowstone with the CESM-supported compilers Intel, GNU, and PGI (e.g. Sect. 4.4 of Baker et al., 2015). Note that the Intel ensemble is the 151 member set generated on Yellowstone and the suggested default for CAM-ECT in Baker et al., 2015.

### 2.4.1 Code modification results

Recall that we ran 30 simulations for each code modification and the failure rates were determined with the exhaustive-testing tool EET. If these ensembles possessed enough variability, we would expect the failure rates to be nearly 0.5%, as the modification experiments should not be climate-changing. Fig. 2.1 shows that the code modification experiments' EET failure rates against the Intel, GNU, and PGI compiler CAM-ECT ensembles are about an order of magnitude higher than the selected 0.5% false positive rate. Furthermore, their failure rates vary across the code changes and between the three ensembles; this instability is an indication of the deficiency of

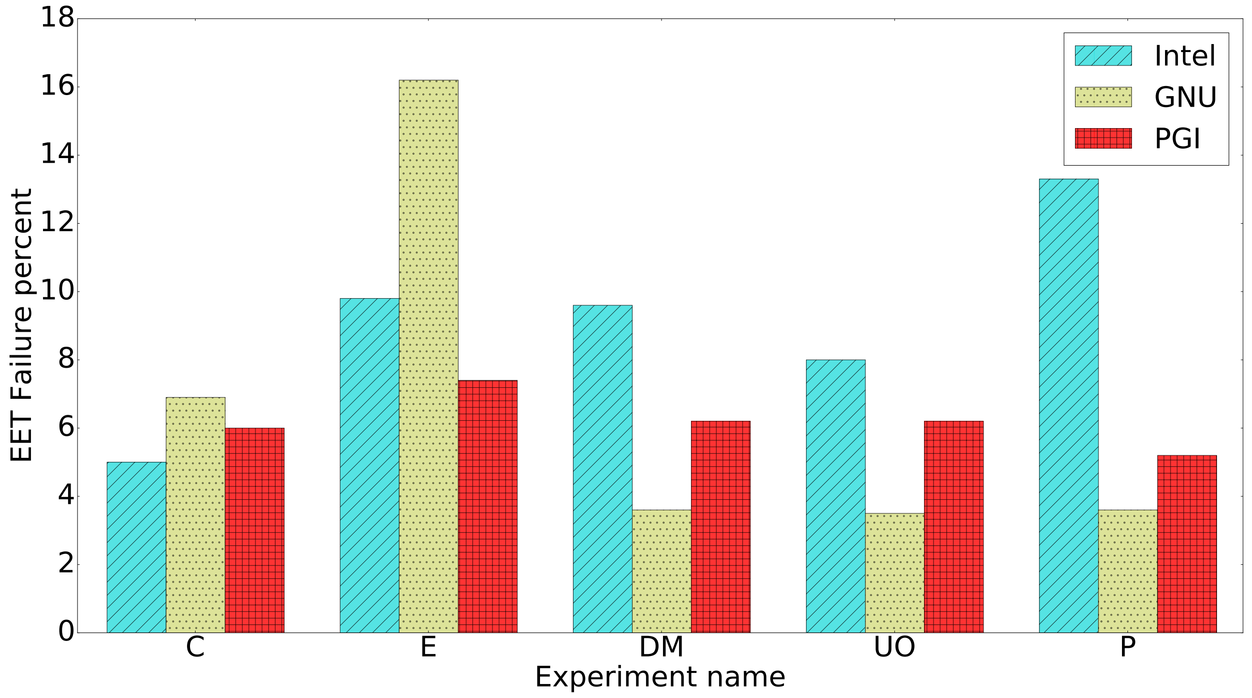


Figure 2.1: Exhaustive failure percentages for code modifications from Sect. 2.3 against original size 151 ensembles from Baker et al., 2015.

variability in each of the ensembles. Ideally the failure rates would be equal across compilers and test cases, and should achieve the 0.5% false-positive rate. Note that **DM**, **UO**, and **P** exhibit a similar failure pattern, possibly suggesting that the Intel, PGI, and GNU compiler ensembles contain increasing variability, respectively. It is also possible that these three experiments' variability more closely match that of the GNU ensemble than that of Intel or PGI, thus explaining the lower failure rates against the GNU ensemble.

### 2.4.2 Compiler effects

We expect compiler effects to be akin to code modifications, as they occur across the code at each time step (as opposed to an initial perturbation). Therefore, as a first step to understanding the compiler effects on Yellowstone, we perform exhaustive consistency testing on the simulations composing each ensemble, which is essentially a "self-test" that is intended as a first-order assessment of CAM-ECT. Tests performed on members against ensembles generated from the same members (i.e. Intel simulations tested against the Intel ensemble) should pass with error rates approximately equal to our false positive rate (0.5%). Empowered by EET, we test the Intel, GNU, and PGI simulations used in Fig. 2.1 against the ensembles composed of them– a total of 562,475 pyCECT evaluations. The results are presented in Fig. 2.2a. Because the Intel, GNU, and PGI compilers on Yellowstone are all CESM-supported configurations, they should pass. Although the failure rates for the self-tests are low, the cross-compiler tests exhibit failure rates well above the specified false positive rate. This issue is not observed in Baker et al., 2015, as only one random selection of three runs from each of the PGI and GNU sets is tested against the Intel ensemble, and with the single sample both tests pass.

The limitation of this self-testing is that the files used to generate the ensembles (and thus principal components) are used in the test itself. Therefore, for a more rigorous test, we perform experiments where the ensemble members and experimental sets are disjoint by randomly excluding 30 simulations from the 181 simulations for each Yellowstone compiler (Intel, GNU, and PGI). We randomly select three sets of 30 simulations per compiler to exclude from the 181, and we run these excluded simulations against the nine ensembles formed by excluding the three sets from each compiler, resulting in 81 tests. Fig. 2.2b depicts the tests composed of disjoint ensemble and experimental sets averaged by compiler and designated Intel-rand, GNU-rand, and PGI-rand. For example, the Intel-rand experiment tested against the Intel-rand ensemble (leftmost bar in Fig. 2.2b) represents the average of nine EET tests for the three experimental Intel sets (30 simulation experimental sets: Intel-rand1, Intel-rand2, and Intel-rand3) against the three Intel ensembles (151 simulation ensembles: Intel-rand1, Intel-rand2, and Intel-rand3). Note that the suffix on each experiment and ensemble (e.g. "rand1") designates the simulations randomly excluded from the ensemble. Concretely, this means that the union of the Intel-rand1 experimental set with the Intel-rand1 ensemble set yields the full 181 member Intel simulation set. The high failure rates present in Fig. 2.2b are evidence that 151 member ensembles with a single compiler are variationally deficient. Notice that experiments in both plots of Fig. 2.2 manifest failure rates comparable to

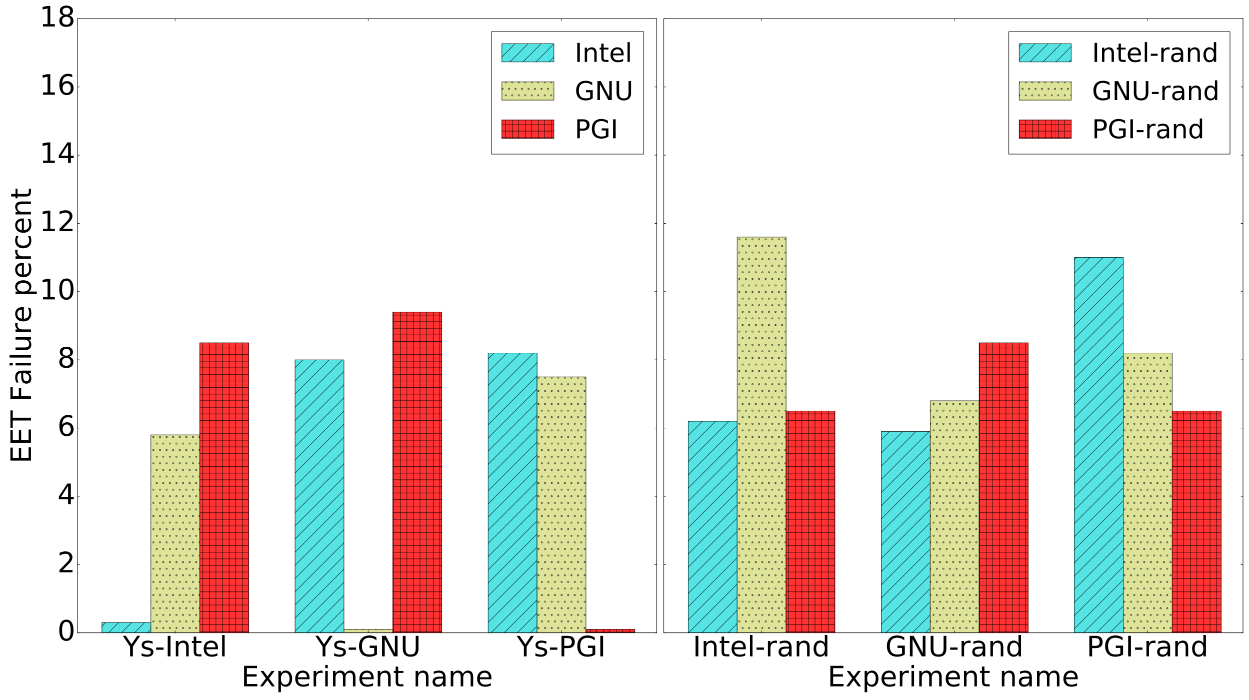


Figure 2.2: EET failure percentage grouped by experiment. Colors and hatching indicate ensemble used in comparison. **2a** (left) shows the so-called "self-tests" of the designated ensembles against themselves; "Ys" abbreviates Yellowstone. **2b** (right) depicts disjoint experiments, e.g. the GNU-rand experiment tested against the Intel-rand ensemble is the average of nine EET tests of the three experimental GNU sets against the three Intel-rand ensembles.

those of the code modification experiments in Fig. 2.1. Now we examine the effect of pooling the compiler ensembles together in an effort to increase the ensemble's variability. Our goal is to align the failure rates of non-climate changing experiments like the Intel experimental set and the code modifications with the specified CAM-ECT false positive rate.

## 2.5 CAM-ECT ensemble composition

Results from the previous section indicate that the default size 151 Intel, GNU, and PGI single-compiler ensembles do not contain sufficient variability. We now increase ensemble variability by using results from multiple compilers in a single ensemble and exhaustively test the code modification experiments against the new combined-compiler ensembles.

We create three new ensembles from subsets of the size 151 "rand" ensembles from Sect. 2.4.2.

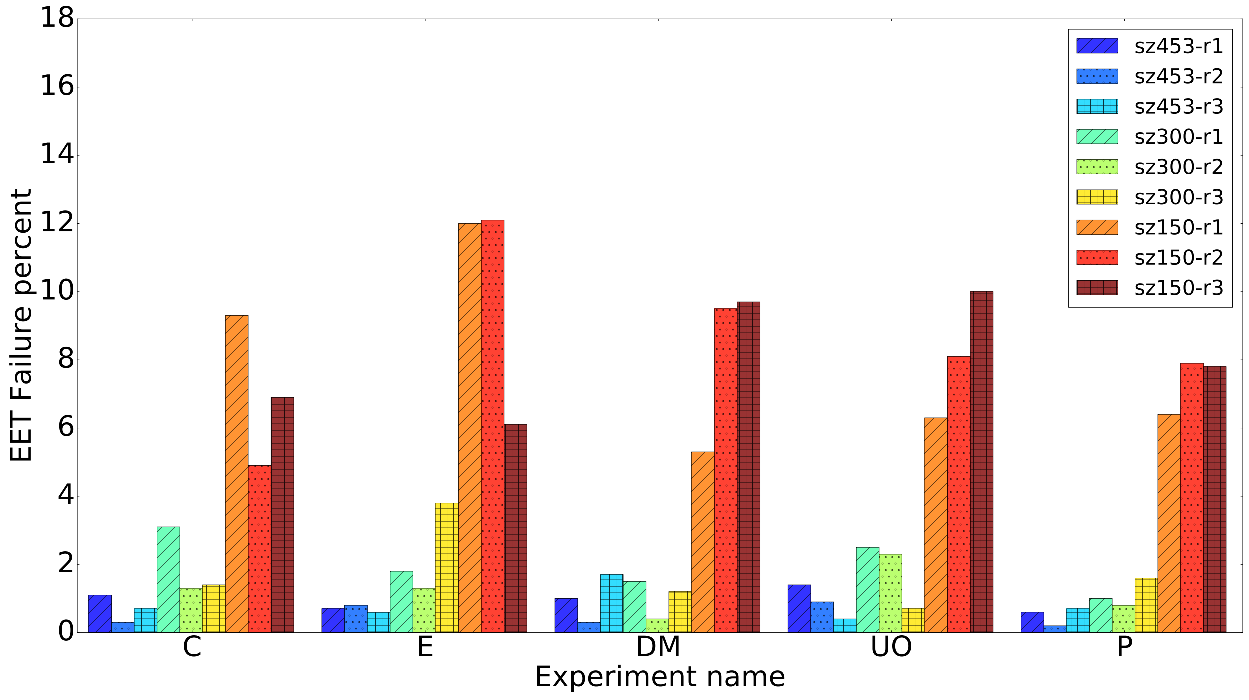


Figure 2.3: EET failure percentage grouped by code modification experiment Sect. 2.3. Colors and hatching indicate ensemble used in comparison. For example, sz300-r1 is 100 Intel-r1, 100 GNU-r1, and 100 PGI-r1 combined. The failure rates of these experiments against the sz453 ensembles are close to 0.5%.

First we create combined-compiler ensembles of size 150 by making three random selections of 50 simulations from each ensemble such that the corresponding CAM initial temperature perturbations form a disjoint cover of the 150 (zero perturbation was excluded) perturbations. The three new ensembles are labeled sz150-r1, sz150-r2, and sz150-r3 to designate the randomly excluded set. We also look at the effect of larger aggregate ensembles and construct three size 453 ensembles by combining the 151 rand ensembles (from Sect. 2.3) from each compiler. Three size 300 ensembles are similarly constructed. Fig. 2.3 shows the results of EET testing of the code modifications against the nine new aggregate ensembles. The "-r*" suffix designates the random set used to construct the ensemble (e.g. sz453-r3 is 151 Intel-r3, 151 GNU-r3, and 151 PGI-r3 together).

Since the failure rates for the size 453 ensembles are consistent and approximately equal to our 0.5% false positive rate, this suggests that these ensembles provide adequate variability. Note that the size 150 aggregate ensembles clearly contain insufficient variability and classification power, but the size 300 ensembles perform nearly as well as the size 453. Further refining the constituents and recommended ensemble size for CAM-ECT is a subject of current study.

## 2.6 Applying the new ensemble

The results from CESM-supported machine testing in Baker et al., 2015 with CAM-ECT show that Argonne National Laboratory's Mira (49,152 node Blue Gene/Q cluster with PowerPC A2 CPUs running at 1.6GHz) and the National Center for Supercomputing Applications' Blue Waters (26,868 node Cray XE/XK hybrid with AMD 6276 Interlagos CPUs) machines fail more than expected as compared to other CESM-supported machines. We now re-examine the Mira and Blue Waters results in the context of the new compiler-aggregate ensembles with CAM-ECT to determine whether there is truly a machine issue or whether the initial CAM-ECT ensemble did not contain sufficient variability. For comparison we also include results from the NERSC Edison machine (Cray XC30: 5576 compute nodes with 12-core Xeon E5-2695v2 Ivy Bridge CPUs), which is representative of most CESM-supported machines in Baker et al., 2015 that pass CAM-ECT. We ran EET on sets of 30 experiments from Mira, Blue Waters, and Edison against the new size 453 aggregate ensembles, and the failure rates averaged 11.9%, 25.4%, and 0.7% respectively. Since Mira and Blue Waters exhibit high failure rates, the question is whether the failures indicate that the the ensemble distribution is still too narrow or whether the failures are evidence of an error in the supercomputers' software or hardware. In particular, because of an upcoming CESM experiment on Mira, an investigation into the validity of its high failure rate was of utmost importance.

CESM-ECT is a coarse-grained testing method, and pyCECT simply returns sets of failing principal components. To relate failing principal components in CESM-ECT to sections of code and perhaps hardware, we first needed to understand which CAM variables were problematic. We performed a systematic elimination of variables, which consisted of removing a CAM variable, updating the PCA and determining a new distribution, and running EET to establish the failure rate. Based on the new failure rates, we concluded that six CAM variables merited further inspection.

We repeated pyCECT testing on the Mira experiment with these six variables removed, and observed nearly five times lower failure rates. With input from climate scientists, we found that four of the six variables were featured prominently in the Morrison-Gettelman microphysics kernel (MG1). Next, the open-source KGEN tool Kim et al., 2016 was used to extract the MG1 kernel from CAM and build it as a stand-alone executable. A subset of MG1 variables with larger normalized Root Mean Square (RMS) errors was found on Mira, and these variables' values were output and compared with those executed on Yellowstone. Given the code lines that compute these variables, we hypothesized that Fused Multiply-Add (FMA) instructions caused the RMS error values, and the instructions were disabled via compiler switch (-qfloat=nomaf). A repeat of the KGEN RMS error testing confirmed that the values were then consistent with those produced on Yellowstone. Disabling FMA for the entire CESM code yielded a 0.7% EET failure rate, which is on par with our false positive rate. This investigative process took significant effort, requiring the cooperation of many climate scientists and software engineers for several months. This demonstrates the necessity and utility of coupling CESM-ECT's coarse-grained testing capability with automatic fine-grained error identification, and adding such capability is work in progress.

## 2.7 Conclusions

In this paper, we introduce minimal and legitimate code modifications into CESM to test whether the CAM-ECT ensembles from Baker et al., 2015 possess sufficient variability to classify these code modifications as passes. We conclude that the ensembles do not, as evidenced by the high failure rates in comparison with the CAM-ECT's false positive rate of 0.5%. To address the limited variability, we propose a new ensemble size (453) and composition that includes simulations from multiple compilers. Finally, equipped with this improved ensemble, we are able to identify the source of Mira's high CAM-ECT failure rates and correct it by disabling FMA. The improved CAM-ECT ensemble facilitates optimization and utilization of new hardware and software technologies. This supports the CESM development cycle, whereby new modules and optimization strategies are tested for integration into the model. Future areas of research include a more thorough study of ensemble size and its effects, including a more comprehensive test of random samples to anti-alias sample size and variability, and the addition of automated fine-grained error identification to CESM-ECT.

# Chapter 3

# Early time step statistical consistency testing

The content of this chapter is contained in the refereed journal GMD (Milroy et al., 2018). The included text is verbatim with the exception of hyperlinked references to sections, which have different enumeration by necessity. All figures appear here unaltered.

## 3.1 Introduction

Requiring bit-for-bit (BFB) identical output for quality assurance of climate codes is restrictive. The codes are complex and constantly evolving, necessitating an objective method for assuring quality without BFB equivalence. Once the BFB requirement is relaxed, evaluating the possible ways data sets can be distinct while still representing similar states is nontrivial. Baker et al., 2015 address this challenge by considering statistical distinguishability from an ensemble for the Community Earth System Model (CESM; Hurrell et al., 2013), an open source Earth System Model (ESM) developed principally at the National Center for Atmospheric Research (NCAR). Baker et al., 2015 developed the CESM ensemble consistency test (CESM-ECT) to address the need for a simple method of determining whether non-BFB CESM outputs are statistically consistent with the expected output. Substituting statistical indistinguishability for BFB equivalence allows for more aggressive code optimizations, implementation of more efficient algorithms, and execution on heterogeneous computational environments.

CESM-ECT is a suite of tools which measures statistical consistency by focusing on 12-month output from two different component models within CESM: the Community Atmospheric Model

(CAM), and the Parallel Ocean Program (POP), with ensemble consistency testing tools referred to as CAM-ECT and POP-ECT, respectively. The key idea of CESM-ECT is to compare new non-BFB CESM outputs (e.g., from a recently built machine or modified code) to an ensemble of simulation outputs from an "accepted" configuration (e.g., a trusted machine and software and hardware configuration), quantifying their differences by an objective statistical metric. CESM-ECT returns a pass for the new output if it is statistically indistinguishable from the distribution of the ensemble, and a fail if the results are distinct. The selection of an "accepted" ensemble is integral to CESM-ECT's pass or fail determination for test simulations. The question of the ensemble composition and size for CAM-ECT is addressed in Milroy et al., 2016, which concludes that ensembles created by aggregating sources of variability from different compilers improve the classification power and accuracy of the test. At this time, CESM-ECT is used by CESM software engineers and scientists for both port verification and quality assurance for code modification and updates. In light of the success of CAM-ECT, the question arose as to whether the test could also be performed using a time period shorter than 1 year, and in particular, just a small number of time steps.

The effects of rounding, truncation and initial condition perturbation on chaotic dynamical systems is a well-studied area of research with foundations in climate science. The growth of initial condition perturbations on CAM has been investigated since Rosinski et al., 1997, whose work resulted in the PerGro test. This test examined the rate of divergence of CAM variables at initial time steps between simulations with different initial conditions. The rates were used to compare the behavior of CAM under modification to that of an established version of the model in the context of the growth of machine roundoff error. With the advent of CAM5, PerGro became less useful for classifying model behavior, as the new parameterizations in the model resulted in much more rapid spread. Accordingly, it was commonly held that using a small number of time steps was an untenable strategy due to the belief that the model's initial variability (that far exceeded machine roundoff) was too great to measure statistical difference effectively. Indeed, even the prospect of using runs of 1 simulation year for CESM-ECT was met with initial skepticism. Note that prior to the CESM-ECT approach, CESM verification was a subjective process predicated on climate scientists' expertise in analyzing multi-century simulation output. The success of CAM-ECT's technique of using properties of yearly CAM means (Baker et al., 2015) translated to significant cost savings for verifying the model.

Motivated by the success and cost improvement of CAM-ECT, we were curious as to whether its general technique could be applied after a few initial time steps, in analogy with Rosinski et al., 1997. This strategy would represent potential further cost savings by reducing the length of the ensemble and test simulations. We were not dissuaded by the fact that the rapid growth of roundoff order perturbations in CAM5 negatively impacted PerGro's ability to detect changes due to its comparison with machine epsilon. In fact, we show that examination of ensemble variability after several time steps permits accurate pass and fail determinations and complements CAM-ECT in terms of identifying potential problems. In this paper we present an ensemble-based consistency test that evaluates statistical distinguishability at nine time steps, hereafter designated the Ultra-Fast CAM Ensemble Consistency Test (UF-CAM-ECT).

A notable difference between CAM-ECT and UF-CAM-ECT is the type of data considered. CAM-ECT spatially averages the yearly mean output to make the ensemble more robust (effectively a double average). Therefore, a limitation of CAM-ECT is that if a bug only produces a small-scale effect, then the overall climate may not be altered in an average sense at 12-months, and the change may go undetected. In this case a longer simulation time may be needed for the bug to impact the average climate. An example of this issue is the modification of the dynamics hyperviscosity parameter (NU) in Baker et al., 2015, which was not detected by CAM-ECT. In contrast, UF-CAM-ECT takes the spatial means of instantaneous values very early in the model run, which can facilitate the detection of smaller-scale modifications. In terms of simulation length for UF-CAM-ECT, we were aware that we would need to satisfy two constraints in choosing an adequate number of initial time steps: some variables can suffer excessive spread while others remain relatively constant, complicating pass/fail determinations. Balancing the run time and ensemble variability (hence test classification power) also alters the types of statistical differences the test can detect;

exploring the complementarity between CAM-ECT and UF-CAM-ECT is a focus of our work.

In particular, we make four contributions in this chapter: we demonstrate that adequate ensemble variability can be achieved at the ninth CESM time step in spite of the heterogeneous spread among the variables considered; we evaluate UF-CAM-ECT with experiments from Baker et al., 2015, code modifications from Sect. 2.3 and several new CAM tests; we propose an effective ensemble size; and we demonstrate that changes to the Community Land Model (CLM) can be detected by both UF-CAM-ECT and CAM-ECT.

In Sect. 3.2, we quantify CESM divergence by time step. In Sect. 3.3, we detail the UF-CAM-ECT method. We demonstrate the results of our investigation into the appropriate ensemble size in Sect. 3.4. We present experimental results in Sect. 3.5, provide guidance for the tools' usage in Sect. 3.6, and conclude our discussion of UF-CAM-ECT with Sect. 3.7.

## 3.2 Motivation: CAM divergence in initial time steps

Applying an ensemble consistency test at nine time steps is sensible only if there is an adequate amount of ensemble variability to correctly evaluate new runs as has been shown for the 1-year runs. This issue is key: with too much spread a bug cannot be detected, and without enough spread the test can be too restrictive in its pass and fail determinations. Many studies consider the effects of initial condition perturbations to ensemble members on the predictability of an ESM, and the references in Kay et al., 2015 contain several examples. In particular, Deser et al., 2012 study uncertainty arising from climate model internal variability using an ensemble method, and an earlier work (Branstator et al., 2010) considers the predictability and forecast range of a climate model by examining the separate effects and timescales of initial conditions and forcings. Branstator et al., 2010 also study ensemble global means and spread, as well as undertaking an entropy analysis of leading Empirical Orthogonal Functions (comparable to PCA). These studies are primarily concerned with model variability and predictability at the timescale of several years or more. However, we note that concurrent to our work, a new method that considers 1 s time steps has been developed in Wan et al., 2017. Their focus is on comparing the numerical error in time integration between a new run and control runs.

As mentioned previously, we were curious about the behavior of CESM in its initial time steps in terms of whether we would be able to determine statistical distinguishability. Fig. 3.1 represents our initial inquiry into this behavior. To generate the data, we ran two simulations of 11 time steps each: one with no initial condition perturbation and one with a perturbation of $\mathcal{O}(10^{-14})$ to the initial atmospheric temperature. The vertical axis labels designate CAM variables, while the horizontal axis specifies the CESM time step. The color of each step represents the number of significant figures in common between the perturbed and unperturbed simulations' area weighted global means: a small number of figures in common (darker red) indicates a large difference. Black tiles specify time steps where the variable's value is not computed due to model sub-cycling (Hurrell et al., 2013). White tiles indicate between 10 and 17 significant figures in common (i.e., a small magnitude of difference). Most CAM variables exhibit a difference from the unperturbed simulation at the initial time step (0), and nearly all have diverged to only a few figures in common by step 10. Fig. 3.1 demonstrates sensitive dependence on initial conditions in CAM and suggests that choosing a small number of time steps may provide sufficient variability to determine statistical distinguishability resulting from significant changes. We further examine the ninth time step as it is the last step on the plot where sub-cycled variables are calculated. Of the total 134 CAM variables output by default in our version of CESM (see Sect. 3.3), 117 are utilized by CAM-ECT, as 17 are either redundant or have zero variance. In all following analyses, the first nine sub-cycled variables distinguished by red labels (AODDUST1, AODDUST3, AODVIS, BURDENBC, BURDENDUST, BURDENPOM, BURDENSEASALT, BURDENSO4, and BURDENSOA) are discarded, as they take constant values through time step 45. Thus we use 108 variables from Fig. 3.1 in the UF-CAM-ECT ensemble.

Next we examine the time series of each CAM variable by looking at the first 45 CESM time steps ($t_0$ through $t_{45}$) for 30 simulations to select a time step when all variables experience sufficient divergence from the values of the reference unperturbed simulation. To illustrate ensemble variability at initial time steps, Fig. 3.2 depicts the time evolution of three representative CAM variables from $t_0$ to $t_{45}$. Most CAM variables' behavior is analogous to one of the rows in this figure. The data set was generated by running 31 simulations: one with no initial atmospheric temperature perturbation, and 30 with different $\mathcal{O}(10^{-14})$K perturbations. The vertical axis labels the difference between the unperturbed simulation and the perturbed simulations' area weighted global means, divided by the unperturbed simulation's area weighted global mean value for the indicated variable at each time step. The right column visualizes the distributions of the data in the left column. Each box plot represents the values in the left column at nine time step intervals from 9 to 45 (inclusive). In most cases, the variables attain measurable but well-contained spread in approximately the first nine time steps. From the standpoint of CAM-ECT, Fig. 3.2 suggests that an ensemble created at the ninth time step will likely contain sufficient variability to categorize experimental sets correctly. Using additional time steps is unlikely to be beneficial in terms of UF-CAM-ECT sensitivity or classification accuracy, and choosing a smaller number of time steps is advantageous from the standpoint of capturing the state of test cases before feedback mechanisms take place (e.g. Sect. 3.5.3.3). Note that we do not claim that 9 time steps is optimal in terms of computational cost, but the difference in run time between 9 and 45 time steps is negligible in comparison to the cost of CAM-ECT 12-month simulations (and the majority of time for such short runs is initialization and I/O). We further discuss ensemble generation and size in Sect. 3.4 with an investigation of the properties of ensembles created from the ninth time step and compare their pass/fail determinations of experimental simulations with that of CAM-ECT in Sect. 3.5.

## 3.3 UF-CAM-ECT approach

UF-CAM-ECT employs the same essential test method as CAM-ECT described in Baker et al., 2015, but with a CESM simulation length of nine time steps (which is approximately 5

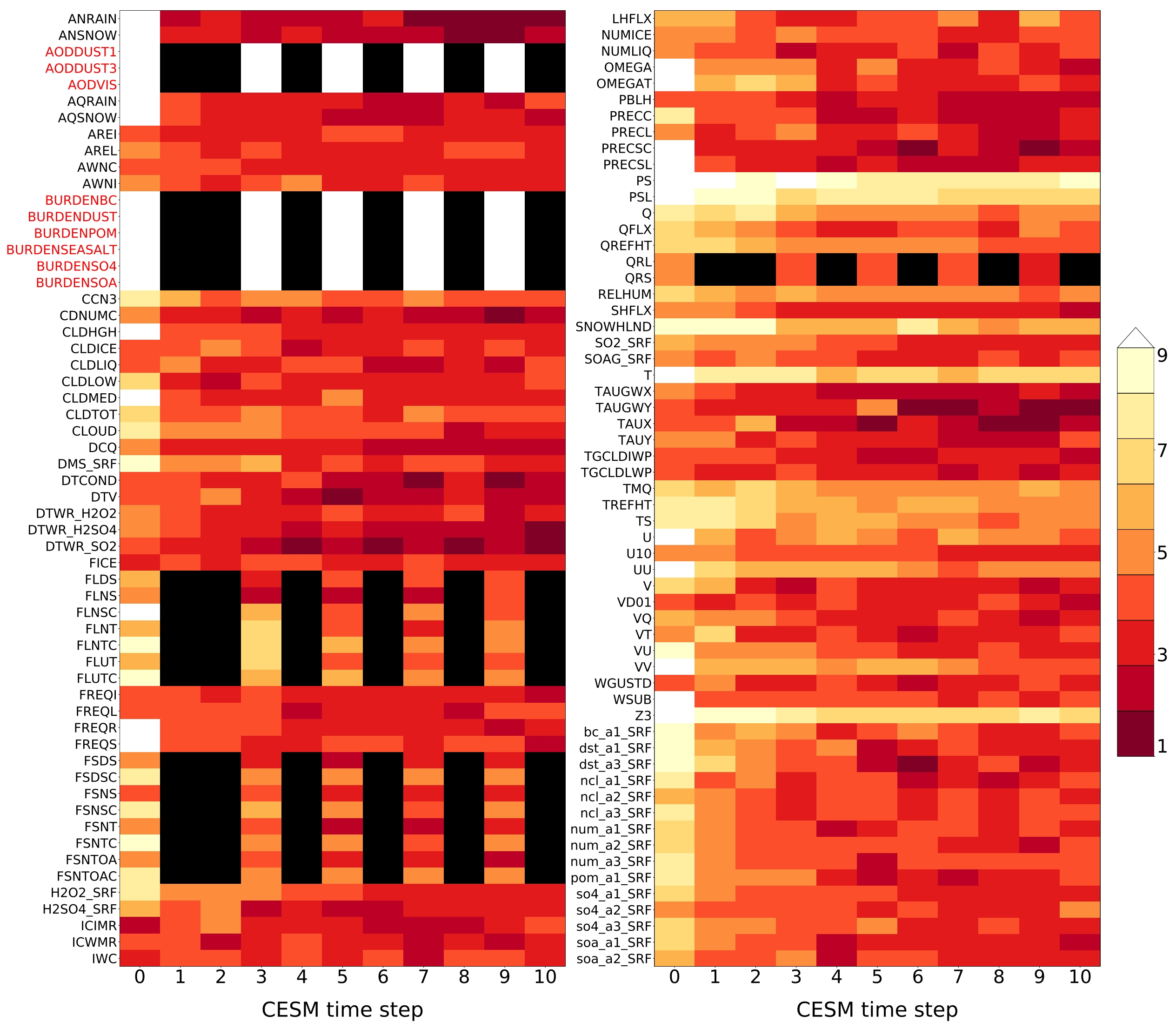


Figure 3.1: Representation of effects of initial CAM temperature perturbation over 11 time steps (including $t = 0$). CAM variables are listed on the vertical axis, and the horizontal axis records the simulation time step. The color bar designates equality of the corresponding variables between the unperturbed and perturbed simulations' area weighted global means after being rounded to $n$ significant digits ($n$ is the color) at each time step. Time steps where the corresponding variable was not computed (subcycled variables) are colored black. White indicates equality of greater than nine significant digits (i.e. 10-17). Red variable names are not used by UF-CAM-ECT.

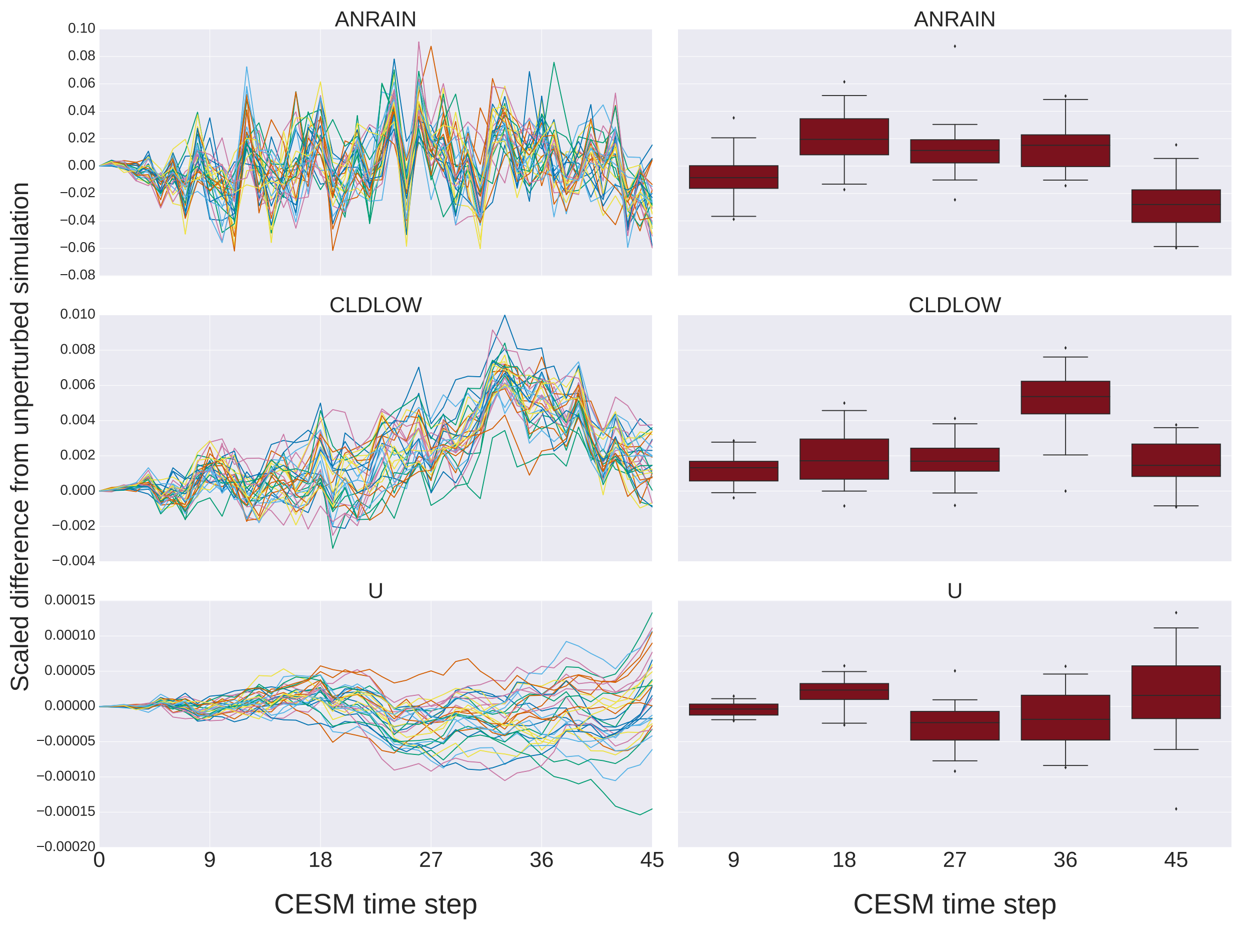


Figure 3.2: The vertical axis labels the difference between the unperturbed simulation and the perturbed simulations' area weighted global means, divided by the unperturbed simulation's area weighted global mean for the indicated variable at each time step. The horizontal axis is the CESM time step with intervals chosen as multiples of nine. The left column plots are time series representations of the values three CAM variables chosen as representatives of the entire set. (Variables behave similarly to one of these three.) The right column plots are statistical representations of the 30 values plotted at each vertical grid line of the corresponding left column. More directly, each box plot depicts the distribution of values of each variable for each time step from 9 to 45 in multiples of 9.

simulation hours) using the default CAM time step of 1800 seconds (30 minutes). By considering a specific time step, we are using instantaneous values in contrast to CAM-ECT, which uses yearly average values. UF-CAM-ECT inputs are spatially averaged, so averaged once, whereas CAM-ECT inputs are averaged across the 12 simulation months and spatially averaged, so averaged twice. As a consequence of using instantaneous values, UF-CAM-ECT is more sensitive to localized phenomena (see Sect. 3.5.3.3). By virtue of the small number of modifications required to transform CAM-ECT into UF-CAM-ECT, we consider the ECT framework to have surprisingly broad applicability. Substituting instantaneous values for yearly averages permits the discernment of different features and modifications- see Sects. 3.5 and 3.5.2 for evidence of this assertion.

As in Baker et al., 2015 and Chapter 2, we run CESM simulations on a $1^\circ$ global grid using the CAM5 model version described in Kay et al., 2015, and despite the rapid growth in perturbations in CAM5 with the default time step of 1800 seconds, we can still characterize its variability. We run simulations with 900 MPI processes and two OpenMP threads per MPI process (unless otherwise noted) on the Yellowstone machine at NCAR. Yellowstone is composed of 4,536 compute nodes, with two Xeon Sandy Bridge CPUs and 32 GB memory per node. The default compiler on Yellowstone for our CESM version is Intel 13.1.2 with -O2 optimization. We also use the CESM-supported compilers GNU 4.8.0 and PGI 13.0 in this study. With 900 MPI processes and two OpenMP threads per process, a simulation of nine time steps on Yellowstone is a factor of approximately 70 cheaper in terms of CPU time than a 12-month CESM simulation.

Either single- or double-precision output is suitable for UF-CAM-ECT. While CAM can be instructed to write its history files in single- or double-precision floating-point form, its default is single-precision which was used for CAM-ECT in Baker et al., 2015 and Sect. 2. Similarly, UF-CAM-ECT takes single-precision output by default. However, we chose to generate double-precision output to facilitate the study represented by Fig. 3.1; it would have been impossible to perform a significance test of up to 17 digits otherwise. In the case of new runs written in double-precision, both CAM-ECT and UF-CAM-ECT compare ensemble values promoted to double-precision with the unmodified new outputs. We determined that the effects of using double- or single-precision outputs for ensemble generation and the evaluation of new runs did not impact statistical distinguishability.

## 3.4 UF-CAM-ECT ensemble size

In this section we consider the properties of the UF-CAM-ECT ensemble, particularly focusing on ensemble size. Given the use of instantaneous values at nine time steps in UF-CAM-ECT, our expectation was that the size of the ensemble would differ from that of CAM-ECT. We considered it plausible that a larger number would be required to make proper pass and fail determinations. The ensemble should contain enough variability that UF-CAM-ECT classifies experiments expected to be statistically indistinguishable as consistent with the ensemble. Furthermore, for experiments that significantly alter the climate, UF-CAM-ECT should classify them as statistically distinct from the ensemble. Accordingly, the ensemble itself is key, and examining its size allows us to quantify the variability it contains as the number of ensemble members increases.

Our sets of experimental simulations (new runs) typically consist of 30 members, but by default pyCECT was written to do a single test on three runs. Performing the full set of possible CAM-ECT tests from a given set of experimental simulations allows us to make robust failure determinations as opposed to a single binary pass/fail test. In this work we utilize the Ensemble Exhaustive Test (EET) tool described in Sect. 2.2 to calculate an overall failure rate for sets of new runs larger than the pyCECT default. A failure rate provides more detail on the statistical difference between the ensemble and experimental set. To calculate the failure rate, EET efficiently performs all possible tests which are equal in number to the ways $N_{test}$ simulations can be chosen from all $N_{tot}$ simulations (i.e., the binomial coefficient: $\binom{N_{tot}}{N_{test}}$). For this work most experiments consist of 30 simulations, so $N_{tot} = 30$ and $N_{test} = 3$ yields 4,060 possible combinations. With this tool we can make a comparison between the exhaustive test failure rate and the single-test CAM-ECT false positive rate calibrated to be 0.5%.

To determine a desirable UF-CAM-ECT ensemble size, we gauge whether ensembles of varying sizes contain sufficient variability by excluding sets of ensemble simulations and performing exhaustive testing against ensembles formed from the remaining elements. Since the test sets and ensemble members are generated by the same type of initial condition perturbation, the test sets should pass should pass. We begin with a set of 801 CESM simulations of nine time steps, differing by unique perturbations to the initial atmospheric temperature field in $\{[-9.99 \times 10^{-14}, 0), (0, 9.99 \times 10^{-14}]\}$ K. The motivation for generating a large number of outputs was our expectation that ensembles created from instantaneous values would contain less variability. Moreover, since the runs are comparatively cheap, it was easy to run many simulations for testing purposes. In the following description, all draws are made without replacement. We first randomly select a subset from the 801 simulations and compute the PC loadings. From the remaining simulations, we choose 30 at random and run EET against this experimental set. For each ensemble size, we make 100 random draws to form an ensemble, and for each ensemble we make 100 random draws of experimental sets. This results in 10,000 EET results per ensemble size. For example, to test the variability of the size 350 ensemble, we choose 350 simulations at random from our set of 801 to form an ensemble. From the remaining 451 simulations, we randomly choose 30 and exhaustively test them against the generated ensemble with EET (4,060 individual tests). This is repeated 99 times for the ensemble. Then 99 more ensembles are created in the same way, yielding 10,000 tests for size 350. As such, we tested sizes from 100 through 750, and include a plot of the results in Fig. 3.3. Since all 801 simulations are created by the same type of perturbation, we expect EET to issue a pass for each experimental set against each ensemble. This study is essentially a resampling method without replacement used jointly with cross validation to ascertain the minimum ensemble size for stable PC calculations and pass/fail determinations. With greater ensemble size the distribution of EET failure rates should narrow, reflecting the increased stability of calculated PC loadings that accompanies larger sample sizes. The EET failure rates will never be uniformly zero due to the statistical nature of the test. The chosen false positive rate of 0.5% is reflected by the red horizontal line in Fig. 3.3. We define an adequate ensemble size as one whose median is less than 0.5% and whose interquartile range (IQR) is narrow. The IQR is defined as the difference between the upper and lower quartiles of a distribution. For the remainder of this work we use the size 350 ensemble shown in Fig. 3.3, as it is the smallest ensemble that meets our criteria of median below 0.5% and narrow IQR. The larger ensembles represent diminishing returns at greater computational expense. Note that the relationship between model time step number and the ensemble size necessary to optimize test accuracy is complex. In Sect. 2.5 we conclude that ensembles of size 300 or 453 are necessary for accurate CAM-ECT test results, which bounds the 350 chosen for UF-CAM-ECT above and below. Minimizing the cost of ensemble generation and test evaluation is not a main consideration of this study, as UF-CAM-ECT is already a sizable improvement over CAM-ECT.

## 3.5 Results

The UF-CAM-ECT must have properties comparable or complementary to CAM-ECT including high classification accuracy. In particular, its response to modifications known to produce statistically distinguishable output should be a fail, and to changes not expected to result in statistically distinguishable output, a pass. We verify its effectiveness by performing the same tests as before with CAM-ECT: CAM namelist alterations and compiler changes from Baker et al., 2015 as well as code modifications from 2.3. We further explore UF-CAM-ECT properties with experiments from CLM and several new CAM experiments. In the following sections, UF-CAM-ECT experiments consist of 30 runs due to their low cost, allowing us to do exhaustive testing. For CAM-ECT, we only run EET (which is far more expensive due to the need for more than three 12-month runs) in Sect. 3.5.3, where the expected experiment outcomes are less certain. The UF-CAM-ECT ensemble selected for testing is size 350 (see Fig. 3.3), and the CAM-ECT ensemble is size 300, comprised of 100 simulations built by Intel, GNU, and PGI compilers (the smallest size recommended in Sect. 2.5).

### 3.5.1 Matching expectation and result: where UF and CAM-ECT agree

UF-CAM-ECT should return a pass when run against experiments expected to be statistically indistinguishable from the ensemble. A comparison between the EET failures of UF-CAM-ECT

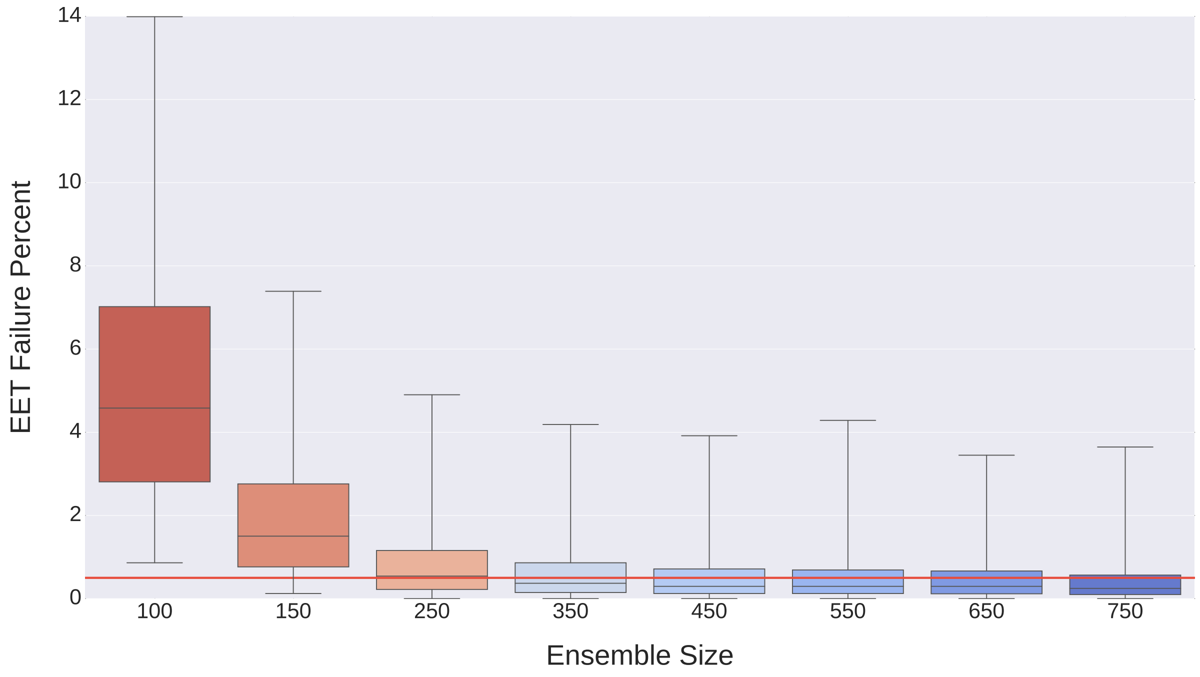


Figure 3.3: Box plot of EET failure rate distributions as a function of ensemble size. The distributions are generated by randomly selecting a number of simulations (ensemble size) from a set of 801 simulations to compute PC loadings. From the remaining set, 30 simulations are chosen at random. These simulations are projected into the PC space of the ensemble and evaluated via EET. For each ensemble size, 100 ensembles are created and 100 experimental sets are selected and evaluated. Thus each distribution contains 10,000 EET results (40,600,000 total tests per distribution). The red horizontal line indicates the chosen false positive rate of 0.5%.

and the single-test CAM-ECT for several types of experiments that should all pass is presented in the upper section of Table 3.1. The first type of examples for this "should pass" category includes building CESM with a different compiler or a different value-safe optimization order (e.g., with no optimization: -O0), or running CESM without OpenMP threading. These tests are labeled *INTEL-15*, *PGI*, *GNU*, *NO-OPT*, and *NO-THRD* (see Appendix A for further details).

A second type of should pass examples includes the important category of port verification to other (i.e., not Yellowstone) CESM-supported machines. In Sect. 2.6 we determined that running CESM with fused multiply–add (FMA) CPU instructions enabled resulted in statistically distinguishable output on the Argonne National Laboratory Mira supercomputer. With the instructions disabled, the machine passed CAM-ECT. We list results from the machines in Table 3.1 with FMA enabled and disabled on *SUMMIT* (note that the *SUMMIT* results can also be found in Anderson et al., 2017), and with xCORE-AVX2 (a set of optimizations that activates FMA) enabled and disabled on *CHEYENNE*.

- ***EDISON*** Cray XC30 with Xeon Ivy Bridge CPUs at the National Energy Research Scientific Computing Center (NERSC; Intel compiler, no FMA capability)
- ***CHEYENNE*** SGI ICE XA cluster with Xeon Broadwell CPUs at NCAR (Intel compiler, FMA capable)
- ***SUMMIT*** Dell C6320 cluster with Xeon Haswell CPUs at the University of Colorado, Boulder for the Rocky Mountain Advanced Computing Consortium (RMACC; Intel compiler, FMA capable)

Finally, a third type of should pass experiments is the minimal code modifications from Sect. 2.3 that were developed to test the variability and classification power of CESM-ECT. These code modifications that should pass UF-CAM-ECT include the following: *Combine* (C), *Expand* (E), *Division-to-Multiplication* (DM), *Unpack-Order* (UO), and *Precision* (P; see Sect. 2.3 for full descriptions). Note that all EET failure rates for the types of experiments that should pass (in the upper section of Table 3.1) are close to zero for UF-CAM-ECT, indicating full agreement between CAM-ECT and UF-CAM-ECT.

Next we further exercise UF-CAM-ECT by performing tests that are expected to fail, which are presented in the lower section of Table 3.1. We perform the following CAM namelist experiments from Baker et al., 2015: *DUST*, *FACTB*, *FACTIC*, *RH-MIN-LOW*, *RH-MIN-HIGH*, *CLDFRC-DP*, *UW-SH*, *CONV-LND*, *CONV-OCN*, and *NU-P* (see Appendix A for descriptions). For UF-CAM-ECT each "expected to fail" result in Table 3.1 (lower portion) is identically a 100% EET failure: a clear indication of statistical distinctness from the size 350 UF-CAM-ECT ensemble. Therefore, the CAM-ECT and UF-CAM-ECT tests are in agreement for the entire list of examples presented in Table 3.1, which is a testament to the utility of UF-CAM-ECT.

### 3.5.2 CLM modifications

The CLM, the land model component of CESM, was initially developed to study land surface processes and land–atmosphere interactions, and was a product of a merging of a community land model with the NCAR Land Surface Model (Oleson et al., 2010). More recent versions benefit from the integration of far more sophisticated physical processes than in the original code. Specifically, CLM 4.0 integrates models of vegetation phenology, surface albedos, radiative fluxes, soil and snow temperatures, hydrology, photosynthesis, river transport, urban areas, carbon–nitrogen cycles, and dynamic global vegetation, among many others (Oleson et al., 2010). Moreover, the CLM receives state variables from CAM and updates hydrology calculations, outputting the fields back to CAM (Oleson et al., 2010). It is sensible to assume that since information propagates between the land and atmosphere models, in particular between CLM and CAM, CAM-ECT and UF-CAM-ECT should be capable of detecting changes to CLM.

For our tests we use CLM version 4.0, which is the default for our CESM version (see Sect. 3.3) and the same version used in all experiments in this work. Our CLM experiments are described as follows:

Table 3.1: CAM-ECT and UF-CAM-ECT return the same result (pass or fail) for the experiments listed. The CAM-ECT column is the result of a single test on three runs. The UF-CAM-ECT column represents EET results from 30 runs. Descriptions of the porting experiments (EDISON, CHEYENNE, and SUMMIT) are listed in Sect 3.5.1. The remaining experiments are described in Appendices A and 2.3.

| **Experiment** | **CAM-ECT** | **UF-CAM-ECT** | |
|---|---|---|---|
| | **Result** | **Result** | **EET failure %** |
| INTEL-15 | Pass | Pass | 0.1% |
| PGI | Pass | Pass | 0.1% |
| GNU | Pass | Pass | 0.0% |
| NO-OPT | Pass | Pass | 0.0% |
| NO-THRD | Pass | Pass | 0.0% |
| EDISON | Pass | Pass | 0.1% |
| CHEYENNE (AVX2 disabled) | Pass | Pass | 2.1% |
| SUMMIT (FMA disabled) | Pass | Pass | 0.0% |
| C | Pass | Pass | 0.7% |
| E | Pass | Pass | 0.0% |
| DM | Pass | Pass | 0.0% |
| UO | Pass | Pass | 0.0% |
| P | Pass | Pass | 0.0% |
| CHEYENNE (AVX2 enabled) | Fail | Fail | 92.2% |
| SUMMIT (FMA enabled) | Fail | Fail | 77.2% |
| DUST | Fail | Fail | 100.0% |
| FACTB | Fail | Fail | 100.0% |
| FACTIC | Fail | Fail | 100.0% |
| RH-MIN-LOW | Fail | Fail | 100.0% |
| RH-MIN-HIGH | Fail | Fail | 100.0% |
| CLDFRC-DP | Fail | Fail | 100.0% |
| UW-SH | Fail | Fail | 100.0% |
| CONV-LND | Fail | Fail | 100.0% |
| CONV-OCN | Fail | Fail | 100.0% |
| NU-P | Fail | Fail | 100.0% |

- ***CLM_INIT*** changes from using the default land initial condition file to using a cold restart.
- ***CO2_PPMV_280*** reduces the $CO_2$ type and concentration from CLM_CO2_TYPE = 'diagnostic' to CLM_CO2_TYPE = 'constant' and CCSM_CO2_PPMV = 280.0.
- ***CLM_VEG*** activates CN mode (carbon–nitrogen cycle coupling).
- ***CLM_URBAN*** disables urban air conditioning/heating and the waste heat associated with these processes so that the internal building temperature floats freely.

See Table 3.2 for the test results of the experiments. The pass and fail results in this table reflect our high confidence in the expected outcome: all test determinations are in agreement, the UF-CAM-ECT passes represent EET failure rates $< 1\%$, and failing UF-CAM-ECT tests are all 100% EET failures. We expected failures for CLM_INIT because the CLM and CAM coupling period is 30 simulation minutes, and such a substantial change to the initial conditions should be detected immediately and persist through 12-months. CLM_CO2_PPMV_280 is also a tremendous change as it effectively resets the atmospheric $CO_2$ concentration to a preindustrial value, and changes which $CO_2$ value the model uses. In particular, for CLM_CO2_TYPE = 'diagnostic' CLM uses the value from the atmosphere (367.0 ppmv), while CLM_CO2_TYPE = 'constant' instructs CLM to use the value specified by CCSM_CO2_PPMV. Therefore both tests detect the large reduction in $CO_2$ concentration, generating failures at the ninth time step and in the 12-month average. CLM_VEG was also expected to fail immediately, given how quickly the CN coupling is expressed. Finally, the passing results of both CAM-ECT and UF-CAM-ECT for CLM_URBAN is unsurprising as the urban fraction is less than 1% of the land surface, and heating and air conditioning only occur over a fraction of this 1% as well.

Our experiments thus far indicate that both CAM-ECT and UF-CAM-ECT will detect errors in CLM, and that a separate CESM-ECT module for CLM (required for POP) is most likely not needed. While this finding may be unsurprising given how tightly CAM and CLM are coupled, it

Table 3.2: These CLM experiments show agreement between CAM-ECT and UF-CAM-ECT as well as with the expected outcome. The CAM-ECT column is the result of a single ECT test on three runs. The UF-CAM-ECT column represents EET failure rates from 30 runs.

| **Experiment** | **CAM-ECT** | **UF-CAM-ECT** | |
|---|---|---|---|
| | **Result** | **Result** | **EET failure %** |
| CLM_INIT | Fail | Fail | 100.0% |
| CLM_CO2_PPMV_280 | Fail | Fail | 100.0% |
| CLM_VEG | Fail | Fail | 100.0% |
| CLM_URBAN | Pass | Pass | 0.1% |

represents a significant broadening of the tools' applicability and utility. Note that while we have not generated CAM-ECT or UF-CAM-ECT ensembles with CN mode activated in CLM (which is a common configuration for land modeling), we have no reason to believe that statistical consistency testing of CN-related CLM code changes would not be equally successful. Consistency testing of active CN mode code changes bears further investigation and will be a subject of future work.

### 3.5.3 UF-CAM-ECT and CAM-ECT disagreement

In this section we test experiments that result in contradictory determinations by UF-CAM-ECT and CAM-ECT. Due to the disagreement, all tests' EET failure percentages are reported for 30 run experimental sets for both UF-CAM-ECT and CAM-ECT. We present the results in Table 3.3. The modifications are described in the following list (note that ***NU*** and ***RAND-MT*** can also be found in Baker et al., 2015 and Milroy, 2015, respectively):

- ***RAND-MT*** substitutes the Mersenne Twister pseudo-random number generator (PRNG) for the default PRNG in radiation modules.
- ***TSTEP_TYPE*** changes the time-stepping method for the spectral element dynamical core from 4 (Kinnmark & Gray Runge–Kutta 4 stage) to 5 (Kinnmark & Gray Runge–Kutta 5 stage).
- ***QSPLIT*** alters how often tracer advection is done in terms of dynamics time steps, the

Table 3.3: These experiments represent disagreement between UF-CAM-ECT and fail CAM-ECT. Shown are the EET failure rates from 30 runs.

| **Experiment** | **CAM-ECT EET failure %** | **UF-CAM-ECT EET failure %** |
|---|---|---|
| RAND-MT | 4.7% | 99.4% |
| TSTEP_TYPE | 2.5% | 100% |
| QSPLIT | 1.8% | 100% |
| CPL_BUG | 41.6% | 0.1% |
| CLM_HYDRO_BASEFLOW | 30.7% | 0.1% |
| NU | 33.0% | 100% |
| CLM_ALBICE_00 | 12.8% | 96.3% |

default is one, and we increase it to nine.

- ***CPL_BUG*** sets albedos to zero above 57 degrees N latitude in the coupler.
- ***CLM_HYDRO_BASEFLOW*** increases the soil hydrology baseflow rate coefficient in CLM from $5.5 \times 10^{-3}$ to 55.
- ***NU*** changes the dynamics hyperviscosity (horizontal diffusion) from $1 \times 10^{15}$ to $9 \times 10^{14}$.
- ***CLM_ALBICE_00*** changes the albedo of bare ice on glaciers (visible and near-infrared albedos for glacier ice) from 0.80,0.55 to 0.00,0.00.

#### 3.5.3.1 Minor setting changes: RAND-MT, TSTEP_TYPE, and QSPLIT

RAND-MT is a test of the response to substituting the CAM default PRNG in the radiation module for a different CESM-supported PRNG (Milroy, 2015). Since the PRNG affects radiation modules which compute cloud properties, it is reasonable to conclude that the change alters the distributions of cloud-related CAM variables (such as cloud covers). Both CAM and its PRNG are deterministic; the variability at nine time steps exhibits different characteristics depending on the PRNG. However, we would not expect (nor would we want) a change to the PRNG to induce statistically distinguishable results over a longer period such as a simulation year, and this expectation is confirmed by CAM-ECT.

TSTEP_TYPE and QSPLIT are changes to attributes of the model dynamics: TSTEP_TYPE alters the time-stepping method in the dynamical core, and QSPLIT modifies the frequency of tracer advection computation relative to the dynamics time step. It is well known that CAM is generally much more sensitive to the physics time step than to the dynamics time step. Time-stepping errors in CAM dynamics do not affect large-scale well-resolved waves in the atmosphere but they do affect small-scale fast waves. While short-term weather should be affected, the model climate is not expected to be affected by time-stepping method or dynamics time-step. However, like the RAND-MT example, UF-CAM-ECT registers the less "smoothed" instantaneous global means as failures for both tests, while CAM-ECT finds the yearly averaged global means to be statistically indistinguishable. Small grid-scale waves are affected by choice of TSTEP_TYPE in short runs, however, the long-term climate is not affected by time-stepping method. The results of these experiments are shown in the top section of Table 3.3, and for experiments of this type, CAM-ECT yields anticipated results. This group of experiments exemplifies the categories of experiments to which UF-CAM-ECT may be sensitive: small-scale or minor changes to initial conditions or settings which are irrelevant in the long term. Therefore, while the UF-CAM-ECT results can be misleading in particular in these cases, they may indicate a larger problem as will be seen in the examples in Sect. 3.5.3.3.

#### 3.5.3.2 Contrived experiments: CPL_BUG and CLM_HYDRO_BASEFLOW

Motivated by experiments which bifurcate the tests' findings, we seek the reverse of the previous three experiments in Sect. 3.5.3.1: examples of a parameter change or code modification that are distinguishable in the yearly global means, but are undetectable in the first time steps. We consulted with climate scientists and CESM software engineers, testing a large number of possible modifications to find some that would pass UF-CAM-ECT and fail CAM-ECT. The results in the center section of Table 3.3 represent a small fraction of the tests performed, as examples that met the condition of UF-CAM-ECT pass and CAM-ECT fail were exceedingly difficult to find. In fact, CPL_BUG and CLM_HYDRO_BASEFLOW were devised specifically for that purpose. That their failure rates are far from 100% is an indication of the challenge of finding an error that is not present at nine time steps, but manifests clearly in the annual average.

CPL_BUG is devised to demonstrate that it is possible to construct an example that does not yield substantial differences in output at nine time steps, but does impact the yearly average. Selectively setting the albedos to zero above 57 degrees N latitude has little effect at nine time steps since this region experiences almost zero solar radiation during the first 5 h of January 1. The nonzero CAM-ECT result is a consequence of using annual averages since for the Northern Hemisphere summer months this region is exposed to nearly constant solar irradiance.

CLM_HYDRO_BASEFLOW is another manufactured example of a change designed to be undetectable at the ninth time step. It is an increase in the exponent of the soil hydrology baseflow rate coefficient, which controls the amount of water drained from the soil. This substantial change (4 orders of magnitude) cannot be detected by UF-CAM-ECT since the differences at nine time steps are confined to deep layers of the soil. However, through the year they propagate to and eventually influence the atmosphere through changes in surface fluxes, which is corroborated by the much higher CAM-ECT failure rate.

#### 3.5.3.3 Small but consequential changes: NU and CLM_ALBICE_00

CAM-ECT results in Baker et al., 2015 for the NU experiment are of particular interest as climate scientists expected this experiment to fail. NU is an extraordinary case, as Baker et al., 2015 acknowledge: “[b]ecause CESM-ECT [currently CAM-ECT] looks at variable annual global means, the ‘pass’ result [for NU] is not entirely surprising as errors in small-scale behavior are unlikely to be detected in a yearly global mean.” The change to NU should be evident only where there are strong field gradients and small-scale precipitation. We applied EET for CAM-ECT with 30 runs and determined the failure rate to be 33.0% against the reference ensemble. In terms of CAM-ECT this experiment was borderline, as the probability that CAM-ECT will classify three

NU runs a “pass” is not much greater than a “fail” outcome. In contrast UF-CAM-ECT is able to detect this difference much more definitively in the instantaneous data at the ninth time step. We would also expect this experiment to fail more definitively for simulations longer than 12-months, once the small but nevertheless consequential change in NU had time to manifest.

CLM_ALBICE_00 affects a very small percent of the land area. Furthermore, of that land area, only regions where the fractional snow cover is $< 1$ and incoming solar radiation is present will be affected by the modification to the bare ice albedo. Therefore, it was expected to pass both CAM-ECT and UF-CAM-ECT, yet the EET failure rate for UF-CAM-ECT was 96.3%. We consider the CLM_ALBICE_00 experiment in greater detail to better understand the differences between UF-CAM-ECT and CAM-ECT. Since the change is small and localized, we need to discover the reason why UF-CAM-ECT detects a statistical difference, particularly given that many northern regions with glaciation receive little or no solar radiation at time step 9 (January 1). To explain this unanticipated result, we created box plots of all 108 CAM variables tested by UF-CAM-ECT to compare the distributions (at nine time steps and at 1 year) of the ensemble versus the 30 CLM_ALBICE_00 simulations. Each plot was generated by subtracting the unperturbed ensemble value from each value, and then rescaling by the unperturbed ensemble value. After analyzing the plots, we isolated four variables (FSDSC: clear-sky downwelling solar flux at surface; FSNSC: clear-sky net solar flux at surface; FSNTC: clear-sky net solar flux at top of model; and FSNTOAC: clear-sky net solar flux at top of atmosphere) that exhibited markedly different behaviors between the ensemble and experimental outputs. Fig. 3.4 displays the results. The left column represents distributions from the ninth time step which demonstrate the distinction between the ensemble and experiment: the top three variables' distributions have no overlap. For the 12-month runs, the ensemble and experiments are much less distinct. It is sensible that the global mean net fluxes are increased by the albedo change, as the incident solar radiation should be a constant, while the zero albedo forces all radiation impinging on exposed ice to be absorbed. The absorption reduces the negative radiation flux, making the net flux more positive. The yearly mean distributions are not altered enough for CAM-ECT to robustly detect a fail (12.8% EET failure rate), which is due to feedback mechanisms having taken hold and leading to spatially heterogeneous effects, which are seen as such in the spatial and temporal 12-month average.

The percentage of grid cells affected by the CLM_ALBICE_00 modification is 0.36% (calculated by counting the number of cells with nonzero FSNS, fractional snow cover (FSNO in CLM) $\in (1, 0)$, and PCT_GLACIER greater than zero in the surface data set). Remarkably, despite such a small area being affected by the CLM_ALBICE_00 change, UF-CAM-ECT flags these simulations as statistically distinguishable. The results of CLM_ALBICE_00 taken together with NU indicate that UF-CAM-ECT demonstrates the ability to detect small-scale events, fulfilling the desired capability of CAM-ECT mentioned as future work in Baker et al., 2015.

## 3.6 Implications and ECT guidelines

In this section we summarize the lessons learned in Sect. 3.5 to provide both clarification and guidance on the use of the complementary tools UF-CAM-ECT and CAM-ECT in practice. Our extensive experiments, a representative subset of which are presented in Sect. 3.5, indicate that UF-CAM-ECT and CAM-ECT typically return the same determination. Indeed, finding counterexamples was non-trivial. Certainly the types of modifications that occur frequently in the CESM development cycle (e.g., compiler upgrades, new CESM-supported platforms, minor code rearrangements for optimization, and initial state changes) are all equally well classified by both UF-CAM-ECT and CAM-ECT. Therefore, in practice we recommend the use of the cheaper UF-CAM-ECT as a first step for port verification, code optimization and compiler flag changes, as well as other frequent CESM quality assurance procedures. Moreover, the low cost of ensemble generation provides researchers and software engineers with the ability to generate ensembles rapidly for evaluation of new physics, chemistry, or other modifications which could affect the climate.

CAM-ECT is used as a second step only when needed for complementary information as follows. First, our experimentation indicates that if UF-CAM-ECT issues a pass, it is very likely that CAM-ECT will also issue a pass. While devising examples where UF-CAM-ECT issues a pass and CAM-ECT issues a fail is conceptually straightforward (e.g. a seasonal or slow-propagating effect),

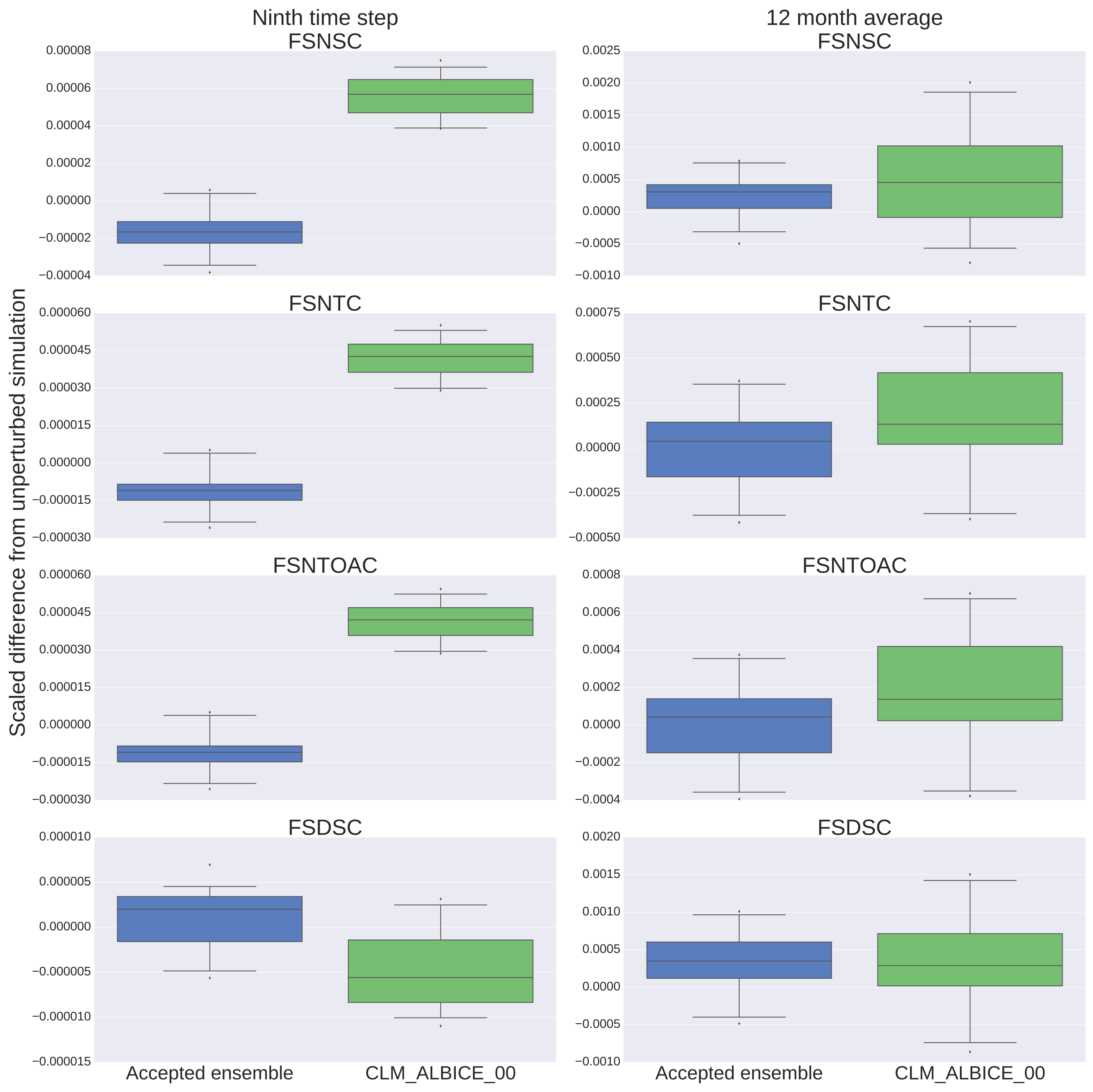


Figure 3.4: Each box plot represents the statistical distribution of the difference between the global mean of each variable and the unperturbed, ensemble global mean, then scaled by the unperturbed, ensemble global mean for both the 30 ensemble members and 30 CLM_ALBICE_00 members. The plots on the left are generated from nine time step simulations, while those on the right are from one simulation year.

in practice none of the changes suggested by climate scientists and software engineers resulted in a discrepancy between CAM-ECT and UF-CAM-ECT. Hence, we constructed the two examples presented in Sect. 3.5.3.2, using changes which were formulated specifically to be undetectable by UF-CAM-ECT, but flagged as statistically distinguishable by CAM-ECT. It appears that if a change propagates so slowly as not to be detected at the ninth time step, its later effects can be smoothed by the annual averaging which includes the initial behavior. Accordingly, the change may go undetected by CAM-ECT when used without EET (e.g., failure rates for CAM-ECT in the lower third of Table 3.3 are well below 100%). A user may choose to run both tests, but in practice applying CAM-ECT as a second step should only be considered when UF-CAM-ECT issues a fail. In particular, because we have shown that UF-CAM-ECT is quite sensitive to small-scale errors or alterations (see CLM_ALBICE_00 in Sect. 3.5.3.3 which impacted less than 1% of land area), by running CAM-ECT when UF-CAM-ECT fails, we can further determine whether the change also impacted statistical consistency during the first year. If CAM-ECT also fails then the UF-CAM-ECT result is confirmed. On the other hand, if CAM-ECT passes, the situation is more nuanced. Either a small-scale change has occurred that is unimportant in the long term for the mean climate (e.g., RAND-MT), or a small-scale change has occurred that will require a longer time scale than 12-months to manifest decisively (e.g., NU). In either case, the user must have an understanding of the characteristics of the modification being tested to reconcile the results at this point. Future work will include investigation of ensembles at longer time scales, which will aid in the overall determination of the relevance of the error.

## 3.7 Conclusions

We developed a new Ultra-Fast CAM Ensemble Consistency test from output at the ninth time step of the CESM. Conceived largely out of curiosity, it proved to possess surprisingly wide applicability in part due to its use of instantaneous values rather than annual means. The short simulation time translated to a cost savings of a factor of approximately 70 over a simulation of 12 months, considerably reducing the expense of ensemble and test run creation. Through methodical testing, we selected a UF-CAM-ECT ensemble size (350) that balances the variability contained in the ensemble (hence its ability to classify new runs) with the cost of generation. We performed extensive experimentation to test which modifications known to produce statistically distinguishable and indistinguishable results would be classified as such by UF-CAM-ECT. These experiments yielded clear pass/fail results that were in agreement between the two tests, allowing us to more confidently prescribe use cases for UF-CAM-ECT. Due to the established feedback mechanisms between the CLM component of CESM and CAM, we extended CESM-ECT testing to CLM. Our determination that both CAM-ECT and UF-CAM-ECT are capable of identifying statistical distinguishability resulting from alterations to CLM indicates that a separate ECT module for CLM is likely unnecessary. By studying experiments where CAM-ECT and UF-CAM-ECT arrived at different findings we concluded that UF-CAM-ECT is capable of detecting small-scale changes, a feature that facilitates root cause analysis for test failures in conjunction with CAM-ECT.

UF-CAM-ECT will be an asset to CESM model developers, software engineers, and climate scientists. The ultra-fast test is cheap and quick, and further testing is not required when a passing result indicating statistical consistency is issued. Ultimately the two tests can be used in concert to provide richer feedback to software engineers, hardware experts, and climate scientists: combining the results from the ninth time step and 12 months enhances understanding of the time scales on which changes become operative and influential. We intend to refine our understanding of both UF-CAM-ECT and CAM-ECT via an upcoming study on decadal simulations. We hope to determine whether the tests are capable of identifying statistical consistency (or lack thereof) of modifications that may take many years to manifest fully. Another potential application of the tests is the detection of hardware or software issues during the initial evaluation and routine operation of a supercomputer.

# Chapter 4

# Identifying the Root Causes of Statistical Inconsistency in the CESM

The content of this chapter is from **D. J. Milroy** et al., arXiv preprint, 2018. The included text is verbatim with the exception of hyperlinked references to sections, which have different enumeration by necessity, and citation format differences. All figures appear here unaltered.

## 4.1 Introduction

This work is prompted by an investigation into output discrepancies between two large supercomputers running the Community Earth System Model (CESM$^{\text{TM}}$). Determining the reason for the statistically distinct model output in more than a million lines of code required equal measures of data analysis, climate science knowledge, experience with the code base, and intuition. The process took the combined expertise of many scientists and engineers and lasted months (Milroy et al., 2016). In this work we make significant progress toward automating root cause analysis for sources of error and discrepancy in CESM.

The CESM is a commonly used application for simulating the Earth system, and its influence extends from science to policy. The model's Fortran code base is modular, which facilitates its evolutionary and community development. The CESM has grown to approximately 1.5 million lines of code, which contain expressions of modern coding techniques together with code written in its earliest versions (decades ago). CESM's size, complexity, and continuous development make finding errors difficult. Furthermore, there are few tools designed for debugging large models written in Fortran. We focus on the CESM in this work, though our debugging methods may be applicable to other large Fortran models or with a different parser, models written in other languages.

The first step to finding sources of inconsistency is to identify abnormal output. A simple test like bit-for-bit equivalence is not useful because legitimate changes or optimizations to the model can result in bitwise differences between outputs. The works Baker et al., 2015; Baker et al., 2016 establish statistical testing for consistency with an ensemble of "accepted" output from the Community Atmospheric Model (CAM) and Parallel Ocean Program (POP) component models of CESM. These Ensemble Consistency Tests (ECTs) quantify the natural climate model variability present in an ensemble of the respective component models' outputs. The ECT can then evaluate new, experimental outputs in the context of the ensemble to determine whether the new outputs are statistically consistent. While this test has been shown to work very well for correctly classifying new outputs, in the case of a failure it provides no information as to the causes. In this work we attempt to develop a path to providing this crucial information on root causes of errors.

This work is organized as follows: in Section 4.2, we overview our strategy and contributions and discuss related work. Section 4.3 describes identifying output variables most affected by inconsistencies. In Section 4.4, we detail transforming approximately 660,000 lines of code into a directed graph (digraph). In Section 4.5 we define our method of iterative convergence to locate sources of discrepancy, and in Section 4.6 we present examples of our method.

## 4.2 Overview and related work

In this section we provide a summary of the methods we develop and describe our principal contributions. We summarize related work on program slicing and runtime sampling.

### 4.2.1 Method and contributions

Each step of our method is motivated by reducing the search space of possible causes of discrepancy. We wish to identify differences between the ensemble and experimental outputs as early as possible, so we examine the model in its first time steps and run consistency testing at time step nine using UF-CAM-ECT (Milroy et al., 2018). Using an early time step is an advantage for

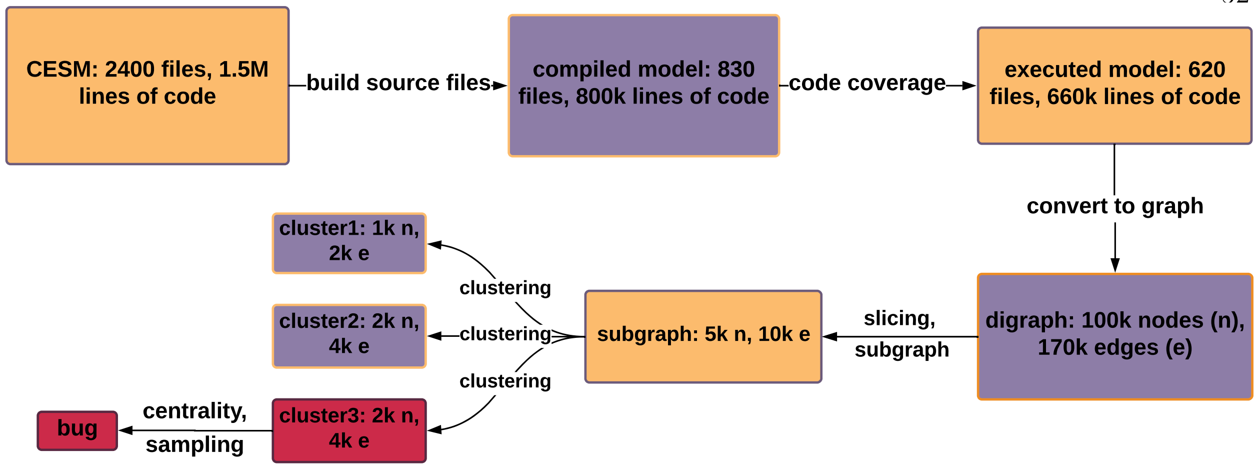


Figure 4.1: Example of process flow for our methods.

several reasons: bugs or discrepancies may not propagate changes through the entire model, climate feedback mechanisms may not yet take effect, and less of the source code is executed. Since CESM can be compiled in numerous configurations we begin by eliminating modules not built into the final executable. We use an existing code coverage tool to discard modules not yet executed by the second time step, and remove subprograms that are unused. Next, we focus on variables written to file that are most affected by the discrepancy, allowing us to disregard locations that compute other variables. These initial steps reduce the potential lines to search from about 1.5 million to 660,000, which is still substantial. From this reduced code base, we construct a digraph of variable dependencies expressed through assignment statements. We then extract from this graph a subgraph that computes the variables identified as affected by the discrepancy. To facilitate parallelism and runtime sampling (among other benefits), we use clustering to partition the subgraph. For each cluster, we rank nodes based on their centrality to determine which code variables to sample at runtime. We plan to further narrow the search space based on value differences between an ensemble and an experimental run, followed by clustering and sampling by centrality to converge iteratively on the sources of discrepancy (currently performed in simulation). Figure 4.1 is a schematic of our process.

We make the following contributions in this work: we create a pipeline to convert the CESM source code into a digraph with extensive metadata that represents variable assignment paths. We develop a hybrid static program slicing approach that efficiently returns large slices. We devise an iterative refinement procedure based on community detection, centrality, and runtime sampling to contract the slice to a useful size. We perform experiments based on CESM output that demonstrate finding the causes of model discrepancy. Finally, we provide evidence that our methods accurately characterize information flow at runtime.

### 4.2.2 Related work

Program slicing is a common technique in debugging and in software development and maintenance that extracts sections of a program that can affect a particular region of code (Weiser, 1981; Weiser, 1984). In a broad sense, program slicing can be divided into two methods: static slicing, which considers all possible executions of a program, and dynamic slicing, which accounts for only one execution given a set of criteria (e.g., Tip, 1994; Silva, 2012). Static slicing is generally less expensive but can return slices that contain too many extraneous statements to be useful (Bent et al., 2001). Dynamic slicing can be far more precise but correspondingly expensive due to the inclusion of algorithms needed to evaluate the satisfiability of sections of the slice (such as SAT or Satisfiability Modulo Theory solvers: Harris et al., 2010). So-called backward slicing considers subsets of code that affect a target location by backward traversal; it can be performed via static or dynamic slicing (Jaffar et al., 2012). We are not aware of any dynamic slicing methods that scale to models consisting of over a million lines of code. We adopt the strategy of hybrid slicing (R. Gupta et al., 1995), which uses dynamic information about program execution to refine static slices. In our case, the dynamic information is provided by a code coverage tool.

Program sampling or instrumentation provides detailed analysis of program states by monitoring variable values at runtime. This type of monitoring can be used to detect divergent values of individual variables but can be extremely expensive (both in space and time) depending on the sampling frequency and the number of variables monitored. Many debuggers and profiling toolkits can perform sampling of large, distributed-memory applications (Allinea MAP and DDT: ARM, 2018, TotalView: RogueWave Software, 2018, and Tau: Shende et al., 2006, to name a few), and tools such as FLiT (Sawaya et al., 2017) and KGen (Kim et al., 2016; Kim et al., 2017) can detect divergent values at runtime. We seek to reduce the search space of CESM to the point that such tools (or those of future design) can identify specific variables that cause model divergence.

## 4.3 Identifying affected output variables

After UF-CAM-ECT returns a failure, we identify CAM output variables that are affected (or most affected) by the cause of the failure. Doing so allows us to make a connection between the model outputs and the code itself. Ideally, we perform a normalized comparison of floating point values at the first model time step, selecting only those variables that exhibit a difference between a single ensemble member and a single experimental run. This approach is the most direct measure of difference, and we recommend using it first due to its simplicity. However, comparing floating point values is seldom useful for narrowing down the number of variables, since in most cases all CAM output variables are different at the model time step zero. For these cases, we instead examine properties of the variables' distributions with two variable selection methods to identify those most affected by the discrepancy.

The first method measures distances between the distribution medians of the ensemble and experimental runs for each variable. To make meaningful distance comparisons across variables, we standardize each variable's distribution by its ensemble mean and standard deviation. Then we identify variables whose interquartile ranges (IQRs) of ensemble and experimental distributions do not overlap. We then rank these variables by descending order of distance between their medians. Although this provides a straightforward ordering of variables, the disadvantage of this approach is that often many variables are identified. Our second method employs logistic regression with regularization via a penalized $L_1$-norm (known as the lasso). We generate a set of experimental runs and use this in conjunction with our ensemble set to identify the variables that best classify the members of each set. We tune the regularization parameter to select about five variables as that yields a subset of CESM and CAM that, in our experiments, contains the known source of statistical inconsistency while still being small. The variables selected by the lasso (and their order) mostly coincide with the order produced by computing the distance between standardized medians. Variable selection for smaller or simpler models may present less of a challenge.

## 4.4 From source code to digraph

Finding lines of code that modify a particular CAM output variable seems a straightforward task: use a text-based search to select code that modifies the variable in question. However, many internal variables may alter values that eventually propagate to the affected output values, and the data dependencies are likely to be complicated. To describe the relationships between CESM variables accurately, we convert each source code file into an Abstract Syntax Tree (AST), which represents code syntax as structural elements of a tree. From the ASTs we create a digraph which represents variable dependencies. Figure 4.2 provides a simple example of the transformation of source code assignments to a digraph.

### 4.4.1 Generating the AST

To construct the AST for CESM, we need to parse the source code. We use the same CESM version as in Kay et al., 2015, and our experimental setup (FC5) consists of a subset of all available component models. Before parsing, we do several preprocessing steps to exclude code that is not executed. Unfortunately, the CESM build system obfuscates which components' Fortran modules are compiled into the specified model. Therefore, we employ KGen (Kim et al., 2016; Kim et al., 2017), a tool to extract and run code kernels as standalone executables, to identify the files compiled into the executable model, reducing the number of modules from approximately 2400 to the nearly 820 used by our experimental setup. KGen also replaces preprocessor directives with their compile-time values, enabling conversion of Fortran code to a Python AST via fparser (based on F2PY: Peterson, 2009). Fparser is the only tool we are aware of to parse Fortran into Python data structures.

We further limit the scope of code considered by examining coverage, which identifies code lines, subprograms, and modules executed in a given application. Since our objective is to identify critical code sections as early as possible in the CESM runtime, we can ignore many subsections of code which are not yet run. To find such code, Intel provides a code coverage tool (Intel, 2017) that writes profiling files that indicate coverage down to individual lines. In our experience, the tool returns accurate evaluations to the level of subprograms, but its behavior at the line-level is inconsistent. Nevertheless, finding entire unused modules and uncalled subprograms is useful and reduces the number of modules and subprograms to be parsed by about 30% and 60%, respectively. We develop software to parse the codecov HTML output, using the output to remove unnecessary modules and comment out unused subprograms.

### 4.4.2 From AST to digraph

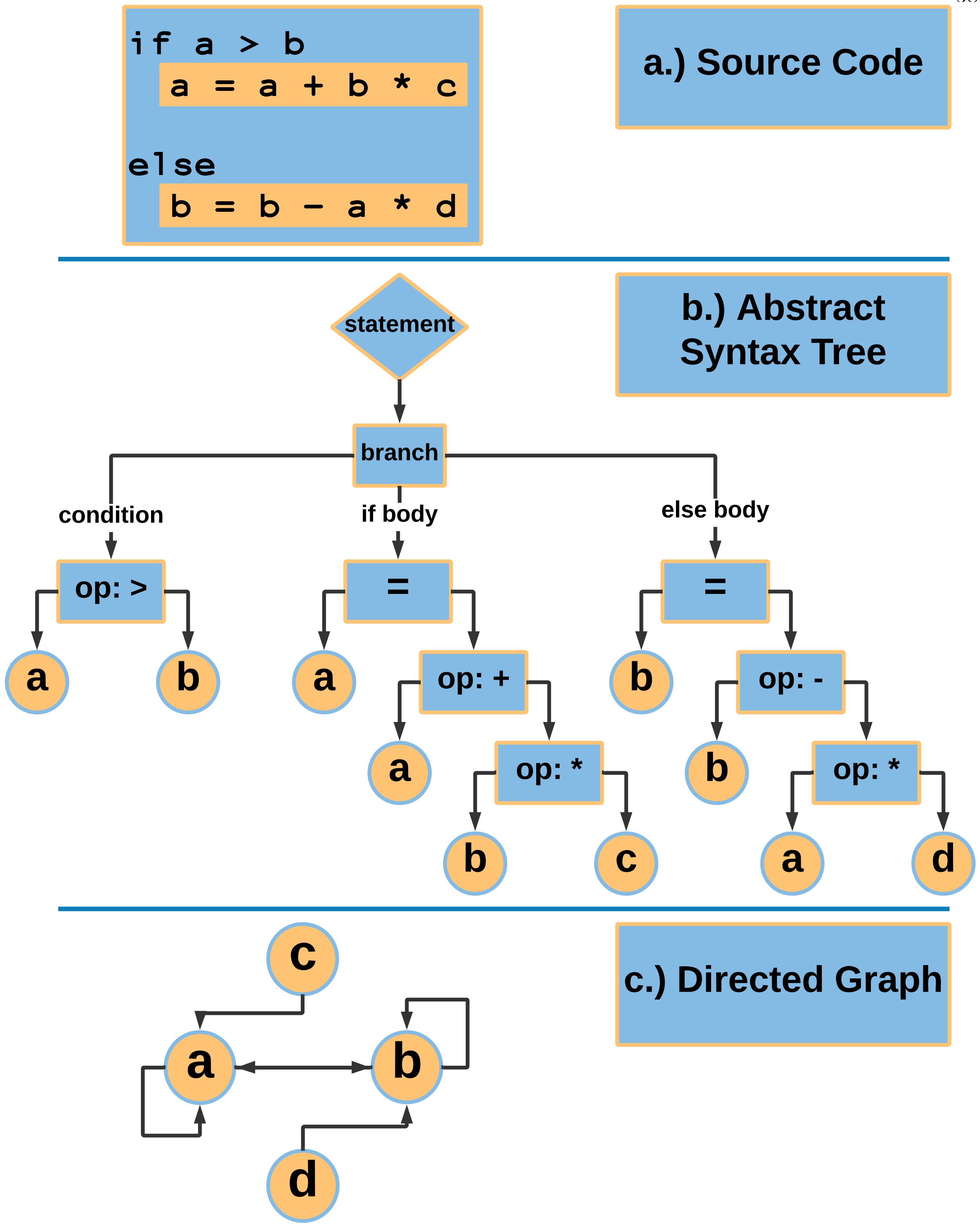
After converting each Fortran module file into an AST, we extract data dependencies to form a digraph. See Figure 4.3 for a visual overview. We need to resolve all assignments, as directed paths of assignments define dependencies between variables. Tracing dependencies between subprograms (similar to interprocedural program slicing: Weiser, 1984) requires processing subroutine and function calls, interfaces, use statements, etc. Assignments without functions or arrays are processed immediately. To allow correct mappings between call and subprogram arguments, parsing statements with calls must be done after all source files are read. Furthermore, Fortran syntax does not always distinguish function calls from arrays, so correct associations must be made after creating a hash table of function names.

Transforming the source code into a digraph presents several challenges. Fparser sometimes fails to convert a Fortran file into an AST due to bugs and statements that exceed fparser's capabilities (e.g., one CESM statement consists of over 3500 characters). In fact, CESM contains thousands of expressions that are highly complex, with deep function and subroutine calls. Because existing Fortran parsing tools are inadequate for CESM, we employ three different parsers for each assignment (some are subjected to multiple passes of these parsers): fparser, KGen helper functions, and our custom string parsing tool based on regular expressions and Python string manipulations.

Processing the ASTs results in a *metagraph* Python class that contains a digraph of internal variables, subprograms, and methods to analyze these structures. CESM internal variables are nodes with metadata, such as location (module, subprogram and line) and "canonical name" (the variable name before being entered into the digraph - which requires unique node names). The digraph component of the metagraph is a NetworkX digraph (Hagberg et al., 2008). NetworkX is a Python graph library that provides an extensive collection of easy to implement graph algorithms and analysis tools.

With static analysis it is not always possible to determine which function a Fortran interface call truly executes at runtime. We adopt the conservative approach of mapping all possible connections. We map the target of use statements to their local names to establish correct local symbols for remote procedures, resolving Fortran renames. If the use statement does not specify an "only list," we map all public variables in the source module to their target module variables. We do not consider chained use statements (i.e., where module A uses B, which uses C), since accurate dependency paths can be created by connecting the statements independently. With these associations defined, we iterate through statements containing subroutine calls and possible functions or arrays. We process subroutine and function calls by treating each argument as a tree, and we successively map outputs of lower levels to corresponding inputs above. Each output gets an edge to the above layer's input, which injects the call's graph structure into the CESM digraph. The top level argument output is connected to the subroutine's corresponding argument in its definition. Discerning functions from arrays is addressed by hash table lookups in the metagraph. Ultimately, the expression's right-hand-side variables and arrays and function (or subroutine argument) outputs are given edges to the left-hand-side.

We adopt a conservative approach for handling composite and complex Fortran data structures. Arrays are considered atomic in that we ignore indices. Pointers are treated as normal variables. Fortran derived types are challenging, as they can be chained into deep composite data structures. We define the indexed element of the derived type as the metagraph canonical name, e.g., `elem(ie)%derived%omega_p` has a canonical name of "omega_p." In effect, we are com-



```
if a > b
   a = a + b * c

else
   b = b - a * d
```

Figure 4.2: Example statement in three forms: a.) source code, b.) source code converted to an AST, and c.) AST assignment statements into a digraph.

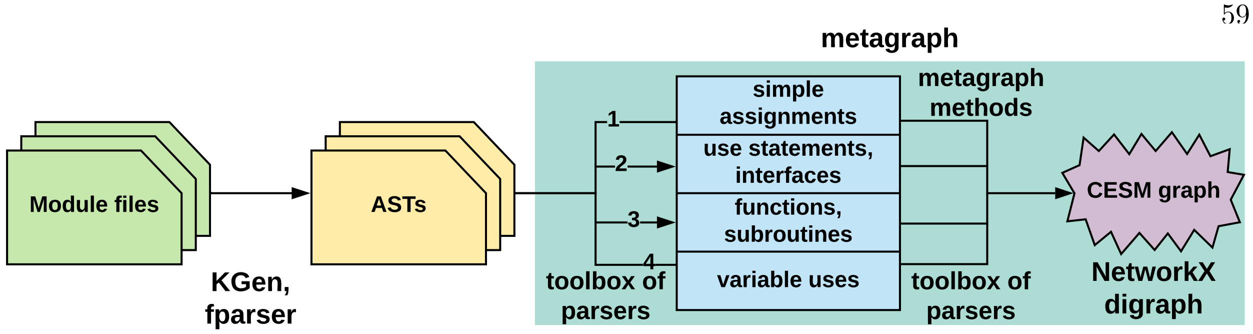


Figure 4.3: Converting Fortran files into a metagraph.

piling the CESM Fortran source code into node relationships in a digraph. Note that our parsing is able to handle all but 10 assignment statements of the 660,000 lines of code in the coverage-filtered source.

## 4.5 Analyzing the CESM graph

We have transformed the CESM code into a digraph that is composed of nodes, which are variables present in assignment expressions, and directed edges that indicate the directionality of the effect of one variable upon another. Now we narrow the scope of our search for bugs by analyzing the graph, usually accomplished by program slicing. Static slicing often produces slices that are too large to locate error sources, and dynamic slicing, while more precise, is too expensive to apply to the CESM graph (about 100,000 nodes and 170,000 edges). Therefore, to make locating internal CESM variables or nodes that influence the values of the affected output variables more tractable, we examine static data dependency paths that terminate on these variables. We mitigate the imprecision of static backward slicing by integrating graph analysis algorithms to refine our slices. In this section, we discuss these methods and propose an iterative subgraph refinement procedure that involves runtime sampling of CESM graph nodes.

### 4.5.1 Tracing affected internal variables in the graph

Since variable relationships in assignment statements are represented as directed edges in the graph, we are interested in directed paths through CESM. These paths ignore control flow such as "if" statements or "do loops," so this approach is akin to backward static slicing. A key difference between our approach and typical program slicing is that nodes in the graph are single variables rather than expressions of multiple variables. Slicing criteria are thus single variables. When used in conjunction with runtime information in the form of code coverage, our method can be considered hybrid slicing (e.g., R. Gupta et al., 1995).

In NetworkX, the fastest way to determine dependencies is by computing shortest paths. In particular, we seek the shortest paths that terminate on output variables. Finding such output variables is a challenge in its own right. Ideally, we would find the locations where I/O calls are made with the output variables as arguments, and find all shortest paths in the graph that end on those calls. Considering these paths does not work well in practice because CESM subprograms that write derived types, e.g., `state%omega` usually take the base derived type (`state`) as an argument, rather than the derived type element (`omega`). This means that there are few paths that terminate on `state%omega` at the call location. We address this problem by searching for paths that terminate on nodes with the canonical name (see Section 4.4) of omega. This approach increases the size of our static slice, but with the attendant advantage that the bug source will very likely be contained in the slice.

CESM I/O statements use temporary variables extensively and include character type variables in the output name argument, so uncovering the exact variable output for a given I/O call must be done with custom instrumentation. Of the nearly 1200 CAM I/O calls which write output variables, many include variables to label the output. To resolve these variables, we instrument the code to print the corresponding string label, permitting a mapping between internal variable names and names written to file. For example, we do not search for paths that end on CAM output flds, but on variables whose canonical names are the internal name `flwds`.

So given a set of output variables that are affected by a certain change, we compute the shortest directed paths that terminate on these variables with Breadth First Search (BFS). After finding these paths, we form the union of the node sets of all such paths. We are interested in the union rather than the intersection as multiple disjoint code sections can be involved in the computation of an affected variable. Such a scenario can arise when conditionals dictate whether I/O calls are executed. Using the union of all shortest paths terminating on the internal canonical names of affected output variables, we induce a subgraph on CESM, which yields the graph containing the causes of discrepancy.

### 4.5.2 Community structure and node centrality

Since CESM and its component models are modular, it is reasonable to conclude that its graph should exhibit clusters corresponding to the modules or related processes. Induced subgraphs of CESM may contain cluster or community structure that can be exploited to improve our search for bug sources, which ends with sampling affected variables. Since sampling can be an expensive process, only a limited number of nodes in the subgraph should be instrumented. By partitioning the subgraph through community detection, we can choose a small number of highly connected nodes in each community to sample and perform the instrumentation of these nodes independently (in parallel). This process can be performed iteratively to reduce the search space.

CAM contains two main processes: physics (sub-grid scale) and dynamics, which taken together feature a set of highly connected modules (the "core"). These CAM modules are involved in the computation of many of the output variables, and bugs are likely to affect multiple output variables. An examination of node connectivity in the core reveals clustering of highly connected nodes in different communities. Although sampling the whole core's most connected nodes may detect floating point differences between ensemble and experimental runs, instrumenting highly connected nodes in each community instead can reduce the distance between instrumented variables and bug locations (reducing the number of iterations needed to refine the search space).

Centrality is a fundamental way to distinguish nodes in a graph. Two simple examples of centrality are degree centrality, which counts the number of edges connected to a given node, and betweenness centrality, which counts the number of BFS or Dijkstra shortest paths (for weighted graphs) that traverse a node (or edge). Graph analysis via centralities proves useful in many diverse areas of research, e.g., Freeman, 1978; Salathé et al., 2010; Shah et al., 2010; Clauset et al.,

2015. A study of the relationship between brain regions' centralities and physical and cognitive function (Heuvel et al., 2013) is particularly relevant to our work. They conclude that such analysis consistently identifies structural hubs (high centrality regions) in the cerebral cortex, and that "high centrality makes hubs susceptible to disconnection and dysfunction."

The Girvan-Newman algorithm (G-N) (Girvan et al., 2002; Newman et al., 2004) is a popular method for identifying communities in undirected graphs. The algorithm is based on edge betweenness centrality, which ranks edges by the number of shortest paths (computed via BFS) that traverse them. The algorithm successively removes the edge with highest centrality in each connected component, which breaks the graph into ever smaller communities. G-N identifies communities via the following steps (Girvan et al., 2002): 1. calculate the betweenness for all edges in the network; 2. remove the edge with the highest betweenness; 3. recalculate betweenness for all edges affected by the removal; 4. repeat from step 2 until no edges remain. In practice each iteration involves removing the edge with the highest betweenness until the number of communities increases (Newman et al., 2004). Note that G-N was formulated to identify communities in undirected graphs. In our case, we convert the directed subgraph into an undirected subgraph for purposes of community detection. This conversion is desirable for our work, as it is equivalent to forming the weakly connected graph of the directed subgraph. Weakly connected graphs are digraphs where any node can be reached from any other node by traversing edges in either direction. Bug locations may be anywhere in the subgraph, so we cannot impose assumptions about whether instrumented nodes are reachable via bug sources in the digraph (even between communities) in either direction. However, in our experiments we know where the bug locations are, so we can simulate how our sampling procedure detects floating point differences between the ensemble and experiment. Given our knowledge of directed paths' connectivity from known bug sources to central nodes, we can deduce whether a difference can be detected. For our method to be useful in situations where bug locations are unknown, we cannot assume such knowledge when we identify communities.

### 4.5.3 Finding important nodes with centrality

Given a modification that alters the values of a set of output variables, we seek locations in CESM that influence their computation. The CESM digraph lacks any information about the nature of connections between variables, so, for example, linear and exponential relationships are expressed identically in the graph. Indeed, the connectivity of the CESM graph is the only information we have to identify important locations in the code for sampling.

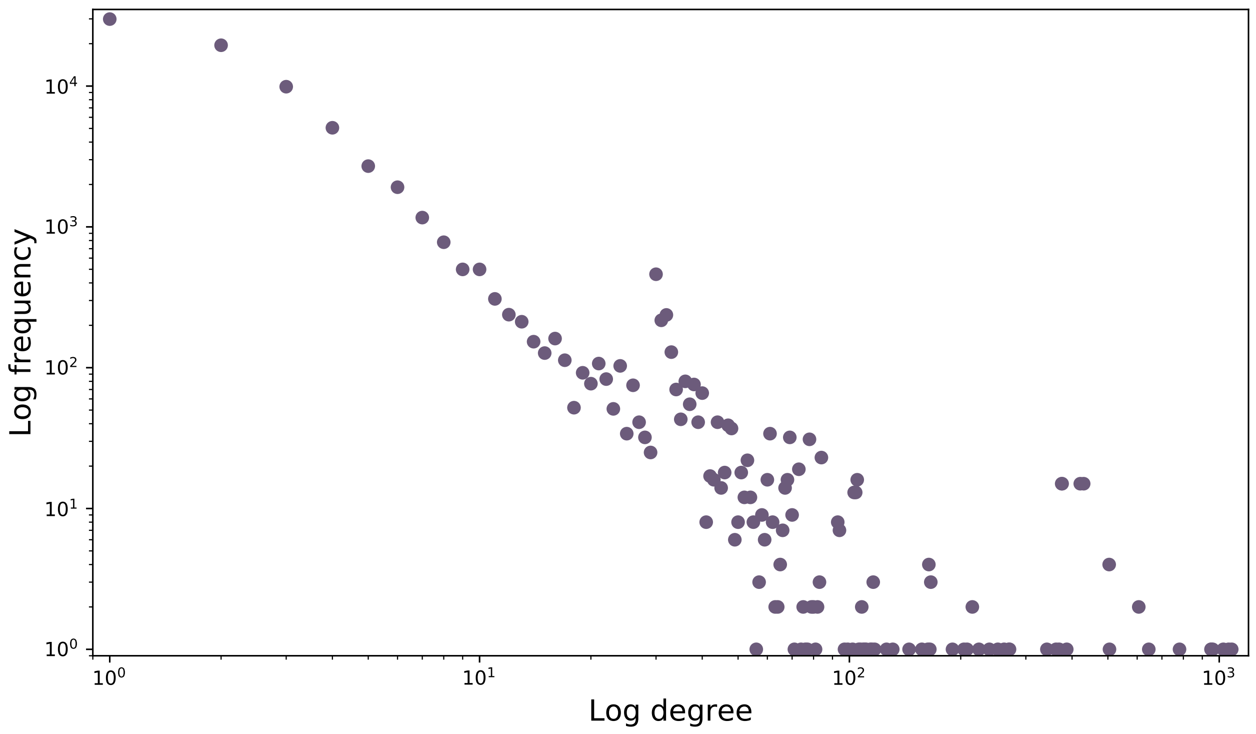


Figure 4.4: Degree distribution of nodes in the CESM digraph.

We use centrality to select nodes whose values are likely to be affected by the causes of statistical distinguishability. We can then sample the variables' runtime values to detect differences between an experimental and a control (or ensemble) run. Eigenvector centrality is a promising choice, as it considers not only the degree of each node, but the degrees of its neighbors and their neighbors, and is related to information flow in a graph. In fact, eigenvector centrality is related to PageRank, which is used to rank web pages in search results (Page et al., 1999). In this work we focus on in-centrality, as we seek nodes which are likely to be affected by the bug sources. From the perspective of sampling, we are looking for information sinks rather than sources. Eigenvector centrality has the disadvantage of favoring hubs (highly connected nodes), which “causes most of the weight of the centrality to concentrate on a small number of nodes in the network” for power law graphs (Martin et al., 2014). The degree distribution of the total CESM graph approximately follows a power law, as can be seen in Figure 4.4. Induced subgraphs of the CESM graph are also plausibly scale-free. A natural question is whether the concentration of centrality on graph hubs has undesirable effects on the ranking of nodes. We found that the application of non-backtracking centrality (based on the Hashimoto matrix: Hashimoto et al., 1989) provides no advantage over standard eigenvector centrality for the CESM graph, its subgraphs, or communities. However, it may prove beneficial for models with graphs that follow a power law that produce more pronounced localization (Martin et al., 2014).

### 4.5.4 Iterative refinement procedure

Once communities are detected in the subgraph, we compute each community’s eigenvector in-centralities and choose the top nodes to sample. The number of nodes to sample is dictated by computational resources. Based on whether a value difference can be found between the nodes sampled in the ensemble run and the experimental run, we can iteratively reduce the size of the subgraph to converge on the sources of statistical inconsistency. This iterative approach is similar to a $k$-ary search, which is a generalization of binary search. In binary search, the search space is halved and a single determination is made at each iteration, however for $k$-ary search the space is partitioned into $k$ sections and $k$ evaluations are made at each iteration. In our case $k$ varies by iteration depending on the number of communities identified. The following algorithm summarizes our overall approach:

**Algorithm 4.5.4**

(1) Perform variable selection detailed in Section 4.3

(2) Map the set of affected CAM output variables in step 1 to their internal CAM variables

$\{V_i\}$

(3) For each affected internal variable $V_i$, use BFS to find the set of nodes $\{n_{i_j}\}$ in all shortest paths that terminate on variables with canonical names equal to $V_i$ in the CESM digraph

(4) Form the induced subgraph $G$ via the union of nodes in the paths in step 3

(5) Use G-N to identify the communities $\{C_k\}$ of undirected $G$ (omitting communities smaller than 3 nodes)

(6) Compute the eigenvector in-centrality for each $C_k$ and select $m$ nodes with largest centrality $\{n_{k_l}\}$

(7) Instrument $\{n_{k_l}\}$ for all $k$ in parallel for an ensemble run and an experimental run, noting the set of nodes which take different values $\{d_{k_l}\}$ ($\{d_{k_l}\} \subseteq \{n_{k_l}\}$)

(8) (a) If $\{d_{k_l}\} = \emptyset$ (i.e., no different values are detected), form the induced subgraph on all nodes in $G$ that are not in BFS shortest paths that terminate on $\{n_{k_l}\}$

(b) Else, form the induced subgraph of $G$ generated by nodes in $G$ that belong to BFS shortest paths that terminate on $\{d_{k_l}\}$

(9) Repeat steps 5-8 until the subgraph is small enough for manual analysis or the bug locations are instrumented

There are three issues involved in the process above that merit discussion. First, it is possible that steps 5-8b in algorithm 4.5.4 do not refine the subgraph of the previous iteration i.e., if the subgraph connectivity is such that all nodes are connected to all central nodes that take different values between the ensemble and experimental runs. In this case, we can select a subset of the most central nodes “most affected” by the bugs. The second issue is that it is possible that the bug sources are not contained in any community i.e., if a bug is in an output variable that has only one neighbor. In this case, no different values will be detected in step 7, and the new induced subgraph will still meet the condition in step 8a. The next iterations will not detect differences, and the successive subgraphs will become increasingly disconnected. Eventually G-N will not identify any communities, and the resulting nodes will need to be analyzed. The third issue is an artifact of static slicing: since the paths do not take into account, e.g., conditional branches, some of the paths may not be traversed. We need to develop a method to track edge traversal and remove invalid paths; algorithm 4.5.4 must only remove nodes that actually can influence $\{n_{k_l}\}$ in step 8a.

Unless otherwise noted, we perform only one iteration of G-N in algorithm 4.5.4 step 5. We could use a larger number to further subdivide the induced subgraph in each iteration (possibly enabling more parallelism), but we adopt a conservative approach to avoid clustering the subgraphs far beyond the natural structure present in the code. Note that excessive G-N iterations would not prevent algorithm 4.5.4 from locating bug sources, but it may slow the process.

## 4.6 Experiments

We apply the overall method discussed in Section 4.5.4 to several experiments. For all but one experiment, we introduce a bug into the source code so that the correct location is known. We then verify that our method can be used to identify the bug location in CESM by demonstrating how it would converge on the location given instrumentation. First, we show that our method can correctly identify straightforward single-line bugs before proceeding to more complicated sources of output discrepancy, such as the identification of variables most affected by certain CPU instructions. We make the following choices and assumptions in our experiments (unless otherwise noted): we restrict our subgraphs to nodes in CAM modules, perform a single G-N iteration, choose the top 10 nodes by in-centrality to sample, and assume all paths are traversed at runtime. Our method can iteratively locate bug sources if our restriction to variables in CAM modules is lifted, but the resulting search may require more iterations than restricting variables to CAM. In the figures that follow, subfigures **a** are the outputs of algorithm 4.5.4 step 4, 8a, or 8b, depending on the iteration or whether simulated sampling detects differences. Subfigures **b** color members of each community discovered by step 5, and subfigures **c** represent the output of step 7 for the community containing the discrepancy sources. Note that we use vector graphics for plots to encourage electronic copy readers to zoom in on features in each figure. Each following subsection describes a different experiment.

### 4.6.1 WSUBBUG

We begin our testing with a bug in an isolated CAM output variable: wsub. By isolated we mean disconnected from the CAM core (see Section 4.5.2) and highly localized. Such a bug has minimal effect and scope which is a good sanity check for our method. The bug consists of a plausible typo (transposing `0.20` to `2.00`) in one assignment of `wsub` in `microp_aero.F90`. The variable is written to file in the next line, so this bug affects only the single output variable. This small change produces a UF-CAM-ECT failure. In this case the median-distance method clearly indicates that the wsub variable is distinct; the distance between the experimental and ensemble medians for this variable is more than 1,000 times greater than for the variable ranked second. The induced subgraph contains only 14 internal variables, all of which are related to wsub, with one being the bug itself.

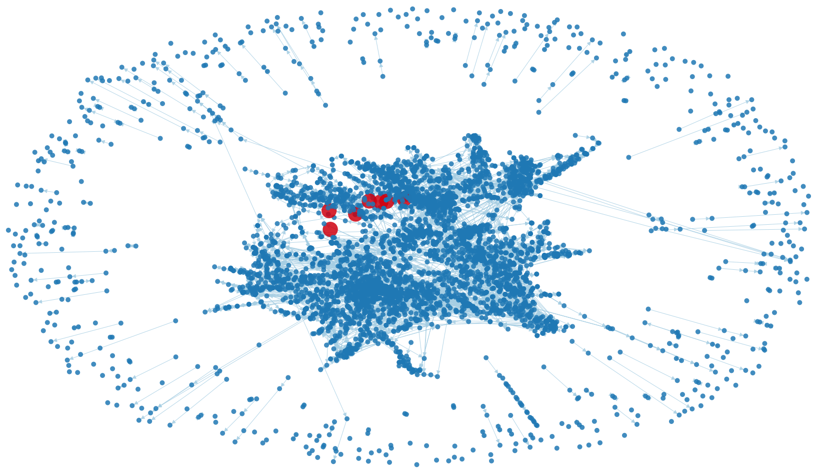
### 4.6.2 RAND-MT

This example, RAND-MT, involves replacing the CESM default pseudo random number generator (PRNG) with the Mersenne Twister. This experiment appears in Milroy et al., 2018 as an example that results in a UF-CAM-ECT failure. The random number generator is used to calculate distributions of cloud-related CAM variables, and this experiment is interesting because it is not a bug and not localized to a single line. We identify the variables immediately influenced or defined by the numbers returned from the PRNG, and consider them to be the bug locations. The lasso variable selection method identifies the five output variables most affected by the PRNG substitution. From these variables, we extract a subgraph of 5,121 nodes and 9,755 edges. Given the size of this induced subgraph, we must use our iterative technique on subgraph communities to reduce the scope of our search. G-N identifies two main communities (blue and green in Fig. 4.5b), in the CAM core. The smaller, green community contains the nodes computed using output from the PRNG. Instrumenting the top 10 most central variables in this community would not detect a difference, as there are no paths from the variables in the bug location to these nodes (see Fig. 4.5c). Creating the induced subgraph of all nodes not in shortest paths terminating on the most central nodes (algorithm 4.5.4 step 8a) admits a dramatic reduction in the search space (Figure 4.6a), which includes disconnected nodes and those with a single neighbor along the perimeter. Instrumenting the most central, orange nodes in Figure 4.6c would indicate a difference as there are multiple paths from the discrepancy sources. This subgraph is small, and the sources are sufficiently near the sampling sites that the cause could be found at this stage.

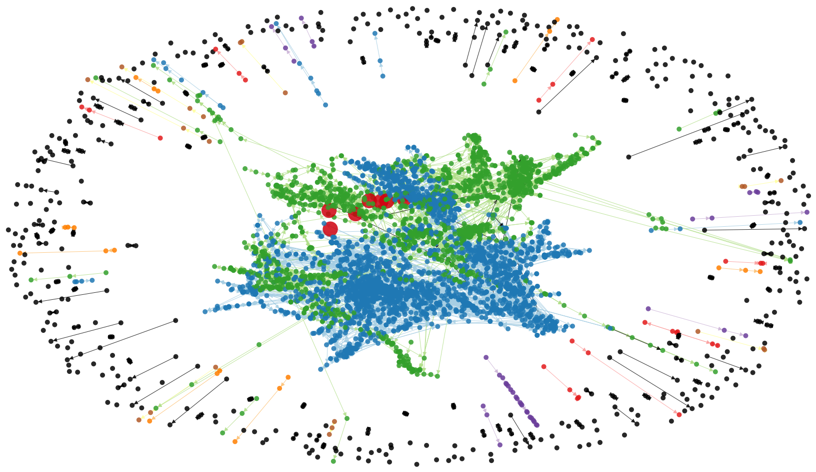
It is noteworthy that the induced subgraph does not contain all the source locations of the statistical distinguishability. The PRNG in CAM is called in two modules: one that computes cloud cover given longwave radiation, and the second with shortwave radiation. The combination of flwds (downwelling longwave flux at surface) and qrl (longwave heating rate) causes the longwave module to be present in the induced subgraph. However, the two variables that are needed to include shortwave radiation in the induced subgraph (fsds and qrs) are not in the set of first five variables returned by lasso.

### 4.6.3 GOFFGRATCH

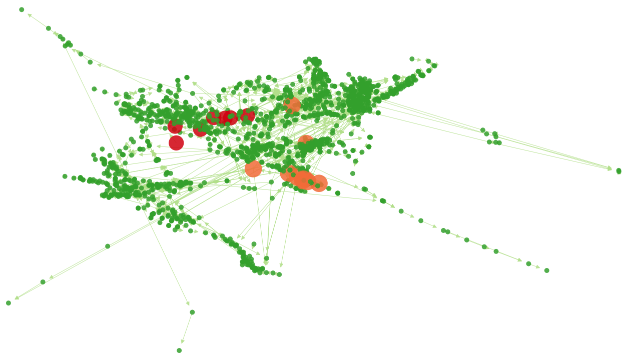
Our third experiment is a modification in the Goff and Gratch Saturation Vapor Pressure elemental function. We change a coefficient of the water boiling temperature from `8.1328e-3` to `8.1828e-3`. This easy to miss typo results in a UF-CAM-ECT failure. The output of the Goff and Gratch function is used extensively in the CAM core, so its effects are not localized. The lasso variable selection method selects 10 variables. Due to experiment-specific conditions, tuning the regularization parameter to select only five variables would require a more sophisticated approach. Inducing a subgraph on locations that compute these variables results in a graph of 5,162 nodes and 9,873 edges (Figure 4.7a). The largest community (blue in Figure 4.7b) contains the nodes affected by the incorrect coefficient. Instrumenting the top 10 most central variables in this community would detect a difference, as there are paths from the variables in the bug



(a)

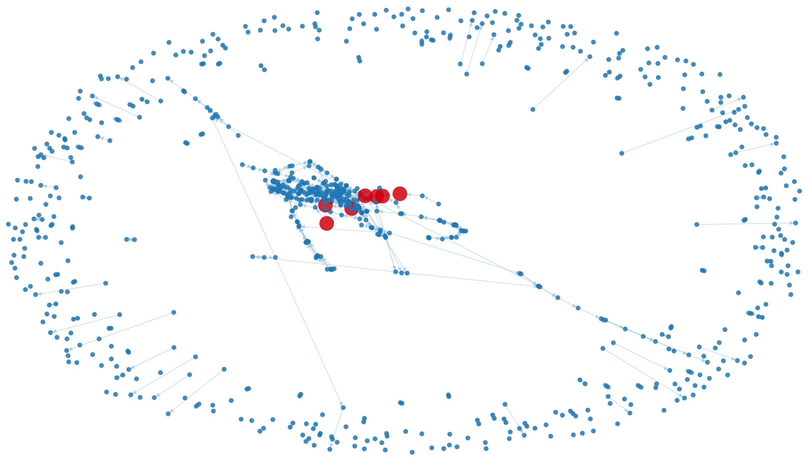


(b)

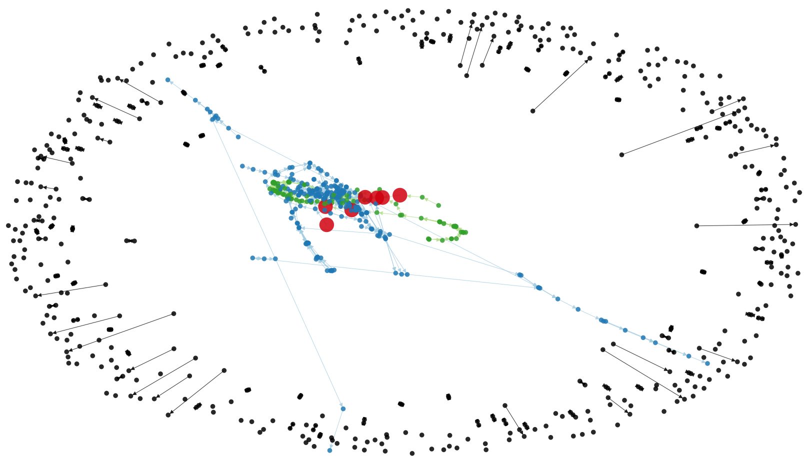


(c)

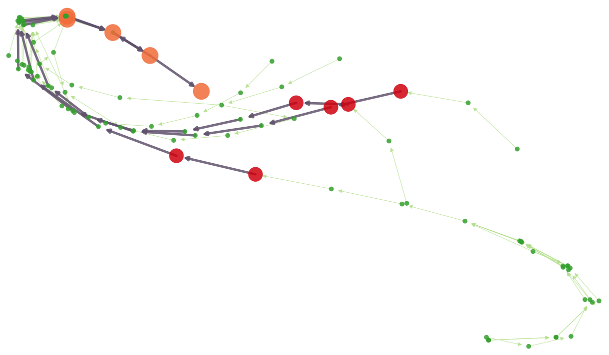
Figure 4.5: RAND-MT first iteration. Variables computed using numbers generated by the Mersenne Twister PRNG are larger red nodes. Larger orange nodes indicate those with the largest eigenvector in-centrality.



(a)



(b)



(c)

Figure 4.6: RAND-MT second iteration. Variables computed using numbers generated by the Mersenne Twister PRNG are larger red nodes. Larger orange nodes indicate those with the largest eigenvector in-centrality. Note the sparsely connected nodes on the perimeter of **a** and **b**: they result from the bug locations not having paths to the most central nodes in the first iteration.

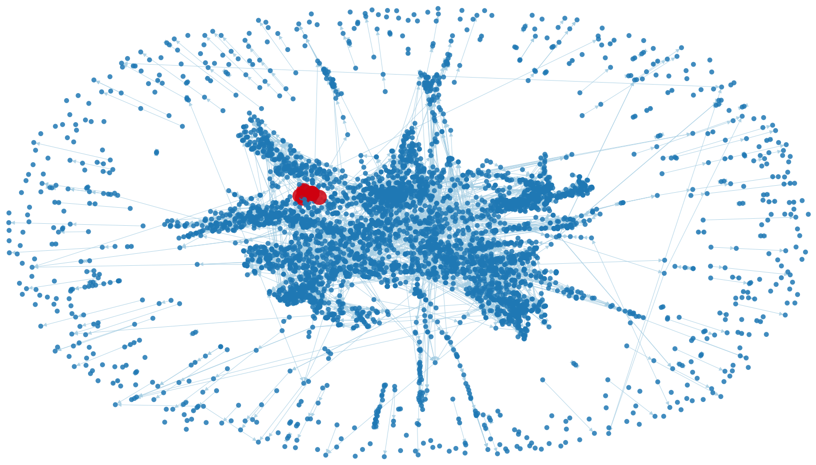
location to these nodes (Figure 4.7c). Creating the induced subgraph of all shortest paths terminating on these central nodes, algorithm 4.5.4 step 8b returns a subgraph that includes part of the green community from the first iteration. Subsequent community detection reveals the remnants of the green community of the first iteration, which are then excluded by sampling. However, in this case, no further simulated iterative refinement can be performed by inducing a subgraph on nodes connected to the instrumented variables, as this subgraph is so highly connected that the induced subgraph equals the community subgraph. In this case, we can rank the differences obtained by sampling and further refine the subgraph based on the nodes with the greatest differences.

### 4.6.4 AVX2

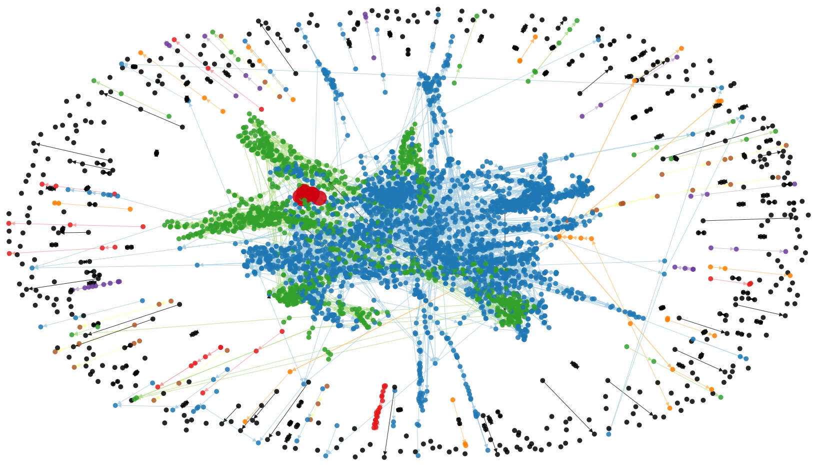
As noted in Section 4.1, this work is motivated by the lengthy, manual investigative process to find the source of statistical distinguishability between CESM outputs generated on the Mira (ALCF, 2018) and on Yellowstone (CISL, 2016) supercomputers described in Milroy et al., 2016. The discrepancy was determined to be caused by FMA instructions used by Mira and sparked interest in developing an automated process.

Using a method that measured each CAM output variable's contribution to the CAM-ECT failure rate, affected variables were identified and located in the Morrison-Gettelman microphysics module (MG1) Milroy et al., 2016. KGen was used to convert this module into a kernel and to find variables which had substantially different RMS values between Yellowstone and Mira. One of these variables was `nctend`, which is modified by a frequently used temporary variable `dum`. `Nctend` also exhibits significantly different values between Yellowstone and Haswell generation (FMA capable) Intel CPUs.

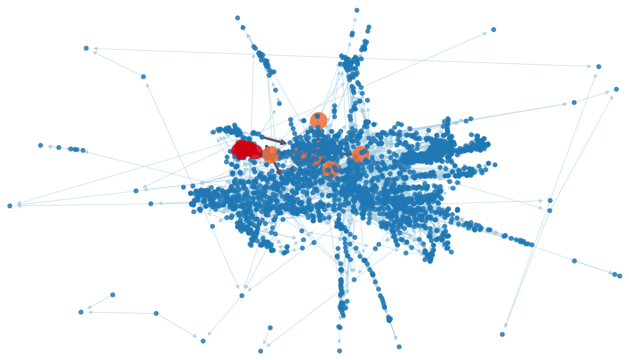
Here we demonstrate that results culminating from months of work by CESM experts could be obtained by our automated method. Because we are unable to use Mira and Yellowstone, we evaluate FMA on Cheyenne (CISL, 2017). Cheyenne contains Intel Broadwell CPUs, which support the Intel AVX2 instruction set, and these instructions include FMA. For our work, we



(a)



(b)



(c)

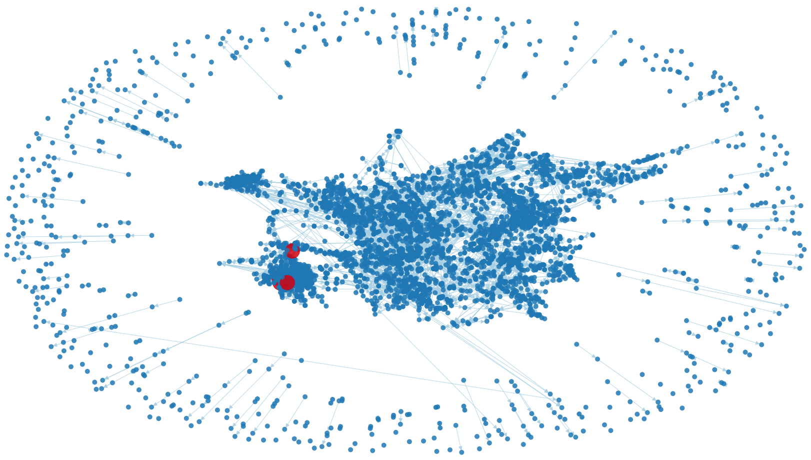
Figure 4.7: GOFFGRATCH, first iteration. The bug locations are indicated as large red nodes, and the top 10 most central variables in the blue (physics) community are indicated by larger orange nodes. Path segments from the bugs to the sampled central nodes are thicker purple edges.

compare an ensemble generated with AVX2 disabled (thus disabling FMA) to an experimental set generated with AVX2 (and FMA) instructions enabled. We verify that enabling AVX2 and FMA causes a UF-CAM-ECT failure (see Table 4.1). Since FMA instructions can be generated from many different lines of source code (distributed sources of discrepancy), we employ KGen to identify a small number of variables affected by AVX2 and FMA to designate as bugs. We extract the Morrison-Gettelman microphysics kernel identified in Milroy et al., 2016 and compare the normalized Root Mean Squared (RMS) values computed by the kernel with AVX2 disabled to the normalized RMS values with AVX2 enabled. KGen flags 42 variables as exhibiting normalized RMS value differences exceeding $10^{-12}$. Here, we determine if our iterative refinement procedure can find some of these variables given CAM outputs most affected by AVX2 instructions.

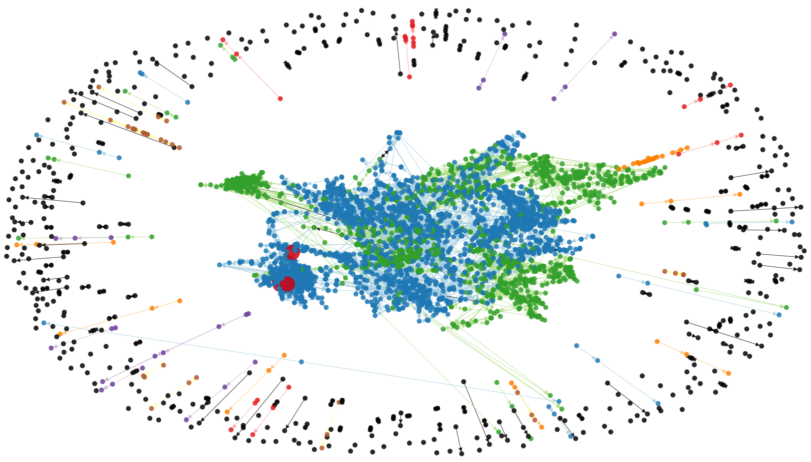
Inducing a subgraph on assignment paths that compute CAM output variables affected by enabling AVX2 instructions (selected by lasso) results in the graph in Figure 4.8a (4,801 nodes and 9,329 edges). Five of the 42 variables identified by KGen are present in this subgraph, all of which are in the blue community of Figure 4.8b. This community contains the CAM core physics processes, of which MG1 forms a central part. The node with the largest eigenvector in-centrality is the temporary, dummy variable `dum` in Figure 4.8c. Four of the five variables with normalized RMS values exceeding our threshold are in the top 15 nodes with the greatest in-centrality. These variables are `nctend`, `qvlat`, `tlat`, and `nitend`. The fifth variable, (`qsout`), is modified by `qniic` (in the top 15 most central nodes) in an assignment statement. All five variables have paths that terminate on all 15 most central nodes. That our iterative refinement procedure would sample and identify the locations of nodes known to be most affected by AVX2 instructions on the first iteration is a testament to the potential utility of our method, particularly in the challenging case where hardware or CPU instructions cause statistical distinguishability.

### 4.6.5 AVX2 in the CESM graph

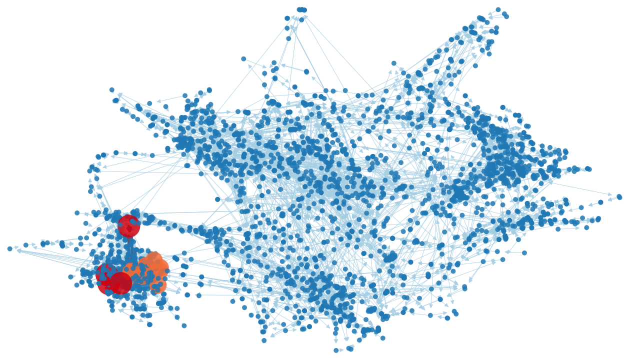
Here, we deviate slightly to discuss how centrality can be used to identify Fortran modules crucial to information flow in the overall CESM graph. While the MG1 module and its constituent



(a)



(b)



(c)

Figure 4.8: AVX2. Variables found to take significantly different normalized RMS values between Broadwell CPUs with AVX2 enabled (FMA enabled) and AVX2 disabled (FMA disabled) are larger red nodes (`nctend`, `qvlat`, `tlat`, `nitend`, and `qsout` in MG1) and also in the top 15 most central nodes. Large orange nodes are remaining nodes in the top 15.

Table 4.1: Selective AVX2 disablement

| Experiment | ECT failure rate |
|---|---|
| AVX2 enabled, all modules | 92% |
| AVX2 disabled, 50 random modules | 85% |
| AVX2 disabled, 50 largest modules | 76% |
| AVX2 disabled, 50 central modules | 13% |
| AVX2 disabled, all modules | 2% |

variables are causes of ECT failure with AVX2 and FMA enabled, these instructions can be generated in many CESM modules. This suggests we compute the (in and out) centrality of the modules themselves (rather than individual variables) to rank them by their potential to propagate FMA-caused differences within CESM. This viewpoint applies to other machine instructions or hardware errors.

To calculate the centrality, we must collapse the graph of variables into modules by considering the graph minor of CESM code formed by the quotient graph of Fortran modules. A graph minor is a subgraph of a graph $G$ obtained by contracting edges of $G$. This graph minor identifies (or collapses) nodes using an equivalence relation, meaning that if two nodes satisfy the equivalence relation, they are replaced by a single node. Edges between equivalent nodes are deleted, and edges between remaining nodes and the new node are preserved. In this case we use the equivalence relation $v_1 \sim v_2 \iff v_1$ and $v_2$ are in the same CESM module (modules become equivalence classes). Applying this equivalence relation to the CESM graph yields a digraph of 564 nodes and 4,263 edges. Selectively disabling AVX2 on the top 50 modules ranked by centrality results in a substantial reduction in the UF-CAM-ECT failure rate in comparison with AVX2 enabled on all modules. Furthermore, this approach exhibits a substantially lower failure rate than disabling AVX2 on 50 modules at random, and even the top 50 modules by lines of code. See Table 4.1 for the failure rates. These results indicate that eigenvector centrality accurately captures the information flow between CESM modules and provides a useful ordering. Selective disablement of instructions such as AVX2 balances optimization with statistical distinguishability and leads to more efficient CPU usage.

## 4.7 Conclusions and future work

The goal of this study is to develop methods that make root cause analysis of CESM possible. To this end, we create a toolkit to convert the CESM source code into a digraph together with metadata that represents variable assignment paths and their properties. We develop an efficient hybrid static program slicing approach based on combining code coverage with BFS. We combine the Girvan-Newman algorithm with eigenvector in-centrality in series to enable runtime sampling of critical nodes. We perform experiments based on CESM output to demonstrate in simulation how our process can find causes of model discrepancy. Finally, we provide evidence that our methods accurately characterize information flow at runtime by successfully finding variables determined to be susceptible to FMA instructions. Creating a method to identify which variables to sample to refine the root cause search space is a significant accomplishment. However, developing and implementing a sampling procedure for the running model is a challenging undertaking that remains to be done. We note that creating a Python interface for LLVM (Lattner et al., 2004) with Flang (PGI, 2017) would allow our parsing to succeed on any compilable Fortran code and that integrating C/C++ capability through Clang (University of Illinois/NCSA, 2007) is also desirable.

## 4.8 Supplementary Material

### 4.8.1 Parsing

Subprogram call arguments can be as deep (in terms of, e.g., function composition) as the stack permits and composed of functions, arrays, strings, derived types, etc. We process such expressions by treating each argument as a tree, and successively map outputs of lower levels to corresponding inputs above. Each output gets an edge to the above layer's input, which injects the call's graph structure into the CESM directed graph. The top level argument output is connected to the subprogram's corresponding argument in its definition. The complexity of discerning functions from arrays is addressed by constant time lookups in the metagraph class function hash table. An additional complexity of assignment statements containing functions is that the expression right-hand-side (RHS) can contain a large (compiler determined) number of functions. Thus processing expressions with many deep functions on the RHS can be expensive. Ultimately, the RHS variables and arrays and function outputs are given edges to the left-hand-side. An example of node-edge mapping within a composite function is provided by the following process, where each function's internal variables will form a path connecting its inputs to its outputs, in order of depth:

$$\begin{aligned}
\omega &= \alpha(b(c,d) * e(f(g + h(i)))) \\
i &\rightarrow input(h) \\
output(h) &\rightarrow input(f) \\
g &\rightarrow input(f) \\
output(f) &\rightarrow input(e) \\
c &\rightarrow input1(b) \\
d &\rightarrow input2(b) \\
output(e) &\rightarrow input(\alpha) \\
output(b) &\rightarrow input(\alpha)
\end{aligned}$$

and the output of the right-hand-side ($\alpha$) gets a directed edge to the left-hand-side ($\omega$):

$$output(\alpha) \rightarrow \omega$$

### 4.8.2 Hashimoto non-backtracking centrality

Scale-free or power law graphs which have degree distributions that are negative exponentials with exponent magnitude greater than 2.5 are identified as causing localization in (Martin et al., 2014). The degree distribution of the total CESM graph approximately follows a power law, as can be seen in Figure 4.9. Induced subgraphs of the CESM graph are also approximately scale-free, consistent with the properties of such graphs (see Figure 4.10 for the GOFFGRATCH experiment subgraph). A natural question is whether the concentration of centrality on graph hubs has undesirable effects on the ranking of nodes. The application of non-backtracking or Hashimoto centrality (Hashimoto et al., 1989) as a substitute for eigenvector centrality for power law graphs is discussed in Martin et al., 2014. We compare the two centralities in Figure 4.11 for the GOFFGRATCH experiment. The Hashimoto non-backtracking centrality indeed distributes the centrality from the hubs to other nodes, but the effect is subtle until approximately the $300^{th}$ ranked node. Also note that the Hashimoto centrality does not provide a rank for all nodes in the subgraph, as can be noted by the sharp drop at the end of its curve in Figure 4.11. This is due to the Hashimoto centrality's use of the line graph of the subgraph's adjacency matrix, which excludes nodes with no neighbors. Although we determine that the non-backtracking centrality provides at best marginal improvement over eigenvector centrality for our graph, we provide a derivation based on that which appears in Martin et al., 2014. Hashimoto centrality may prove beneficial for models with graphs that follow power laws that produce more pronounced localization (Martin et al., 2014).

#### 4.8.2.1 Centrality derivation

This section is a reformulation of the derivation in Martin et al., 2014, which we have reworked in the interest of clarity. Let $G$ be a graph with $n \times n$ adjacency matrix $\mathbf{A}$, set of nodes $V$ and edges $E$. The graph order is the number of nodes: $|V| = n$, while the graph size is the number of edges: $|E| = m$. Note that $m$ is the number of nonzero entries in $\mathbf{A}$ if $G$ is directed, and the number of nonzero entries in the upper or lower triangle of $\mathbf{A}$ if $G$ is undirected.

Let $e \in E$ be represented as $(v_1, v_2)$. If $G$ is directed, $(v_1, v_2) \neq (v_2, v_1)$ and the order represents direction: $v_1 \rightarrow v_2 \coloneqq (v_1, v_2)$. If $G$ is undirected, $(v_1, v_2) = (v_2, v_1)$.

Let $N(i)$ be the set of neighbors of node $i$.

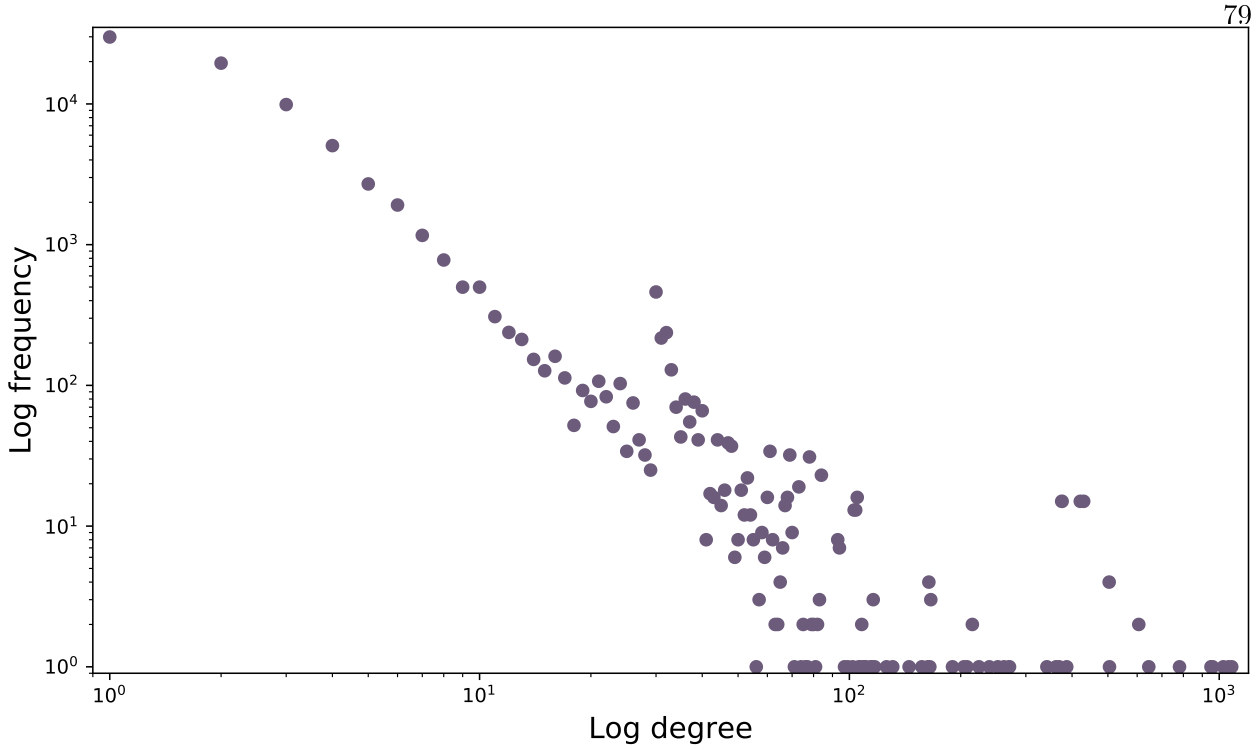


Figure 4.9: Degree distribution of nodes in the CESM digraph.

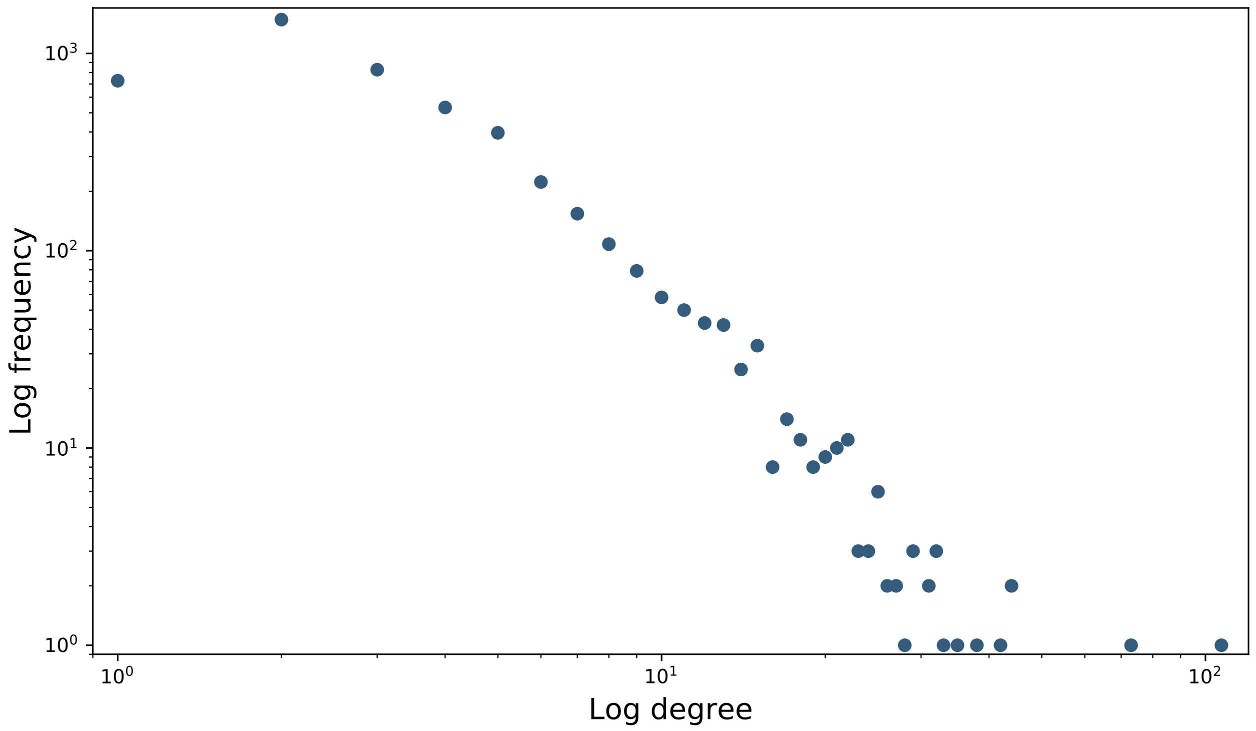


Figure 4.10: Degree distribution of nodes in the GOFFGRATCH digraph.

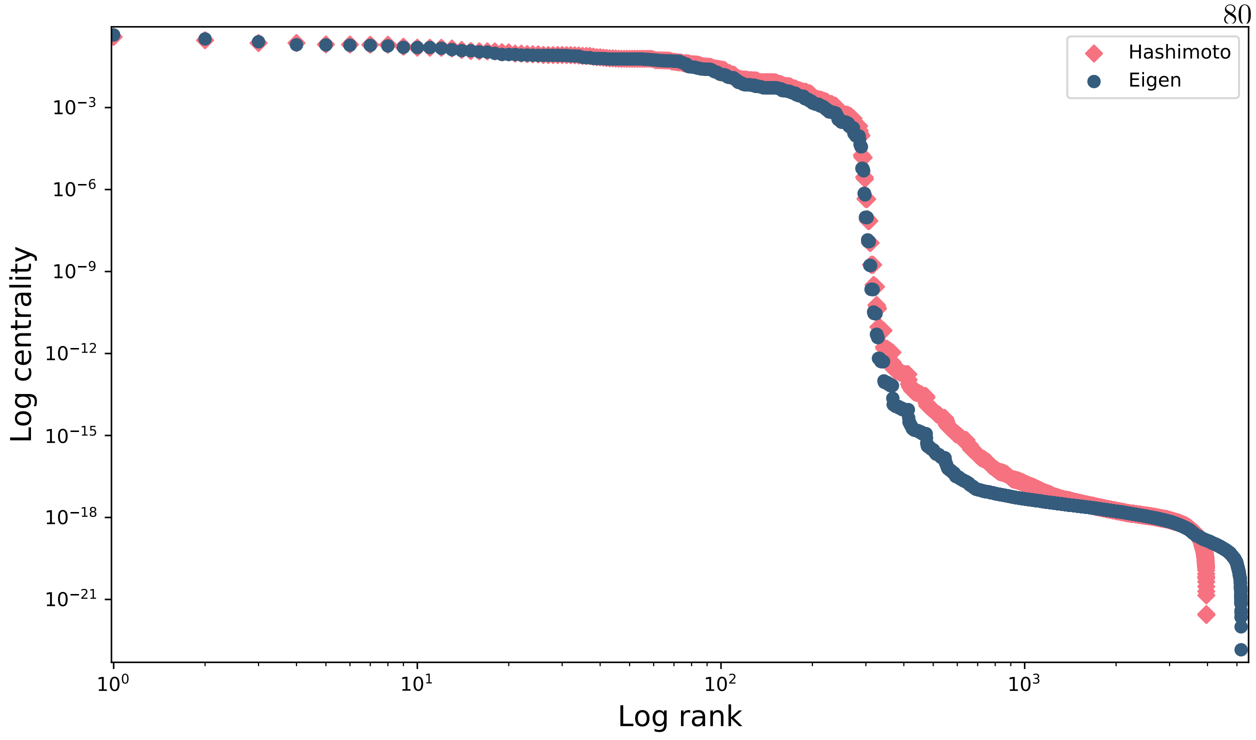


Figure 4.11: Log rank versus log absolute value of centrality for Hashimoto non-backtracking centrality and eigenvector centrality in the GOFFGRATCH experiment subgraph. The absolute value of the centralities is used since the lowest ranked terms are small negative numbers.

The Hashimoto, or non-backtracking matrix (Hashimoto et al., 1989) of graph $G$ is denoted $\mathbf{B}$ and is an adjacency matrix on $E$:

$$\forall (u,v),(w,x) \in E;$$

$$B_{(u,v),(w,x)} \text{ or } B_{(u\to v),(w\to x)} = \delta_{vw}(1-\delta_{ux})$$

Where $\delta$ is the Kronecker delta. For an undirected graph, each $(v_1, v_2) \in E$ becomes two ordered pairs $(v_1, v_2), (v_2, v_1)$. Thus $\mathbf{B}$ is $m \times m$ if $G$ is directed, and $2m \times 2m$ for undirected $G$. $\mathbf{B}$ is closely related to the *line graph* $L(G)$ which is also an adjacency matrix on $E$: $L(G)_{(u\to v),(w\to x)} := \delta_{vw}$

Instead of computing the eigenvector centrality on $\mathbf{A}$, we use $\mathbf{B}$. Let $\lambda$ be the Perron-Frobenius (leading) eigenvalue of $\mathbf{B}$, and $\vec{v}$ be the corresponding eigenvector. Then the out-centrality (corresponding to out-edges) for some $i \in V$ can be derived by starting from the eigenvector equation $\lambda\vec{v} = \mathbf{B}\vec{v}$. To compute the in-centrality used in this work, we can reverse the directed edges of $\mathbf{A}$ (via the transpose $\mathbf{A}^\intercal$).

$$\begin{aligned}
\vec{v}_{(i\to j)} &= \frac{1}{\lambda}\sum_{(k\to l)\in N((i\to j))} B_{(i\to j),(k\to l)}\vec{v}_{(k\to l)} \\
&= \frac{1}{\lambda}\sum_{(k\to l)\in N((i\to j))} \delta_{jk}(1-\delta_{il})\vec{v}_{(k\to l)} \\
&= \frac{1}{\lambda}\sum_{k}^{n}\sum_{l}^{n} A_{kl}\delta_{jk}(1-\delta_{il})\vec{v}_{(k\to l)} \\
&= \frac{1}{\lambda}\sum_{l}^{n} A_{jl}(1-\delta_{il})\vec{v}_{(j\to l)}
\end{aligned}$$

or

$$\vec{v}_{(i\to j)} = \frac{1}{\lambda}\sum_{l\neq i}^{n} A_{jl}\vec{v}_{(j\to l)}$$

then the full non-backtracking centrality of node $i$ is:

$$c_i = \sum_{q\in N(i)} \vec{v}_{(i\to q)}$$

Where we are free to choose a constant to normalize the centrality.

### 4.8.3 Additional experimental results

Note that we refer to the Girvan-Newman algorithm (Girvan et al., 2002; Newman et al., 2004) as G-N.

#### 4.8.3.1 GOFFGRATCH

This supplementary section includes both the first and second iterations of GOFFGRATCH (Figures 4.12 and 4.13, respectively). The second iteration of this experiment does not appear in

Table 4.2: CAM output variables selected by the methods described in Sect. 4.3, and their internal counterparts.

| Experiment | Output variables | Internal variables |
|---|---|---|
| WSUBBUG | wsub | wsub |
| RANDOMBUG | omega | omega |
| GOFFGRATCH | aqsnow, freqs, cldhgh, precsl, ansnow, cldmed, cloud, cldlow, ccn3, cldtot | qsout2, freqs, clhgh, snowl, nsout2, clmed, cld, cllow, ccn, cltot |
| DYN3BUG | vv, omega, z3, uu, omegat | v, omega, z3, u, t |
| RAND-MT | flds, taux, snowhlnd, flns, qrl | flwds, wsx, snowhland, flns, qrl |
| AVX2 | taux, trefht, snowhlnd, ps, u10, shflx | wsx, tref, snowhland, ps, u10, shf |

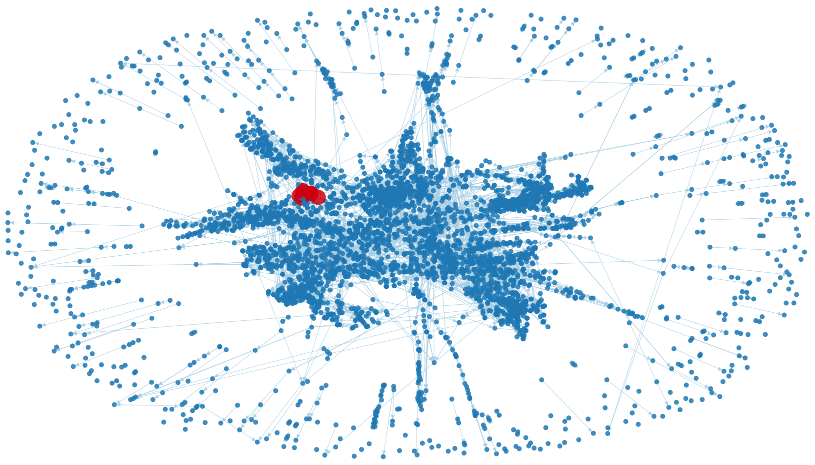
the paper.

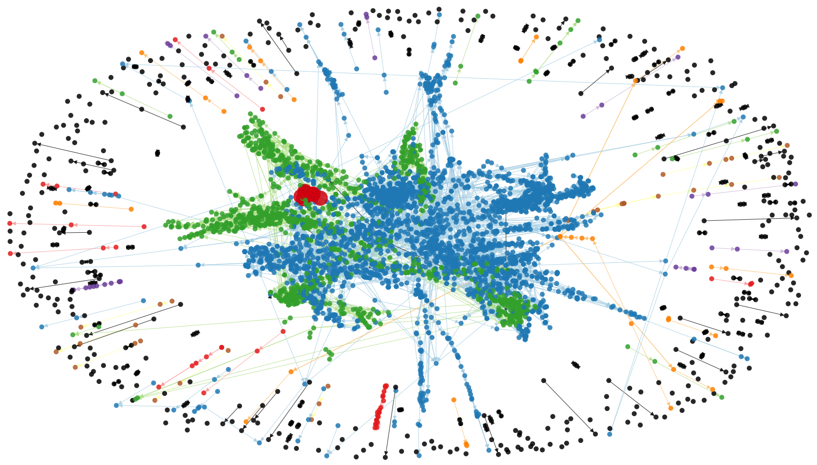
#### 4.8.3.2 RANDOMBUG

We select the module for this bug by randomly choosing a module from the set of CAM modules known to be executed by our simulation in the first time step. We introduce an error in the array index of a variable used to assign the contents of the derived type containing physics state variables (t, u, v, etc.), in particular the state variable omega. As in the previous experiment, this change results in a UF-CAM-ECT failure. Omega is output to file with the value `state%omega`, so we use "omega" as the canonical name for generating the induced subgraph. This experiment is more challenging than WSUBBUG, as omega is computed in other CAM modules, yielding a subgraph of 628 nodes and 295 edges. Applying the G-N algorithm to the remaining nodes identifies several small (fewer than 30 nodes) communities, one of whose most central node is the bug source. See Figure 4.14.

#### 4.8.3.3 DYN3BUG

Another example of a bug consisting of a single line change is located in a dynamics subroutine that computes hydrostatic pressure in the CAM core. The bug particularly affects the five variables listed in Table 4.2. We apply our iterative refinement to the induced subgraph of 6,017 nodes and 11,512 edges (Figure 4.15a), and successfully separate the red dynamics community from the blue


(a)


(b)

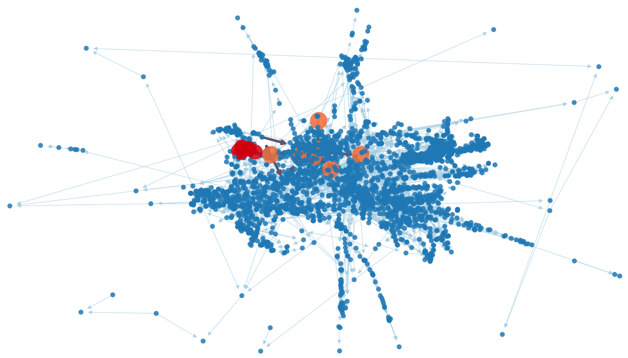
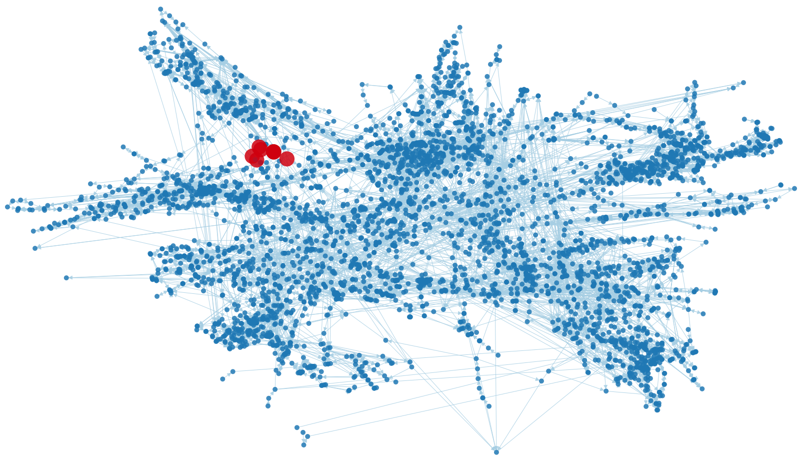
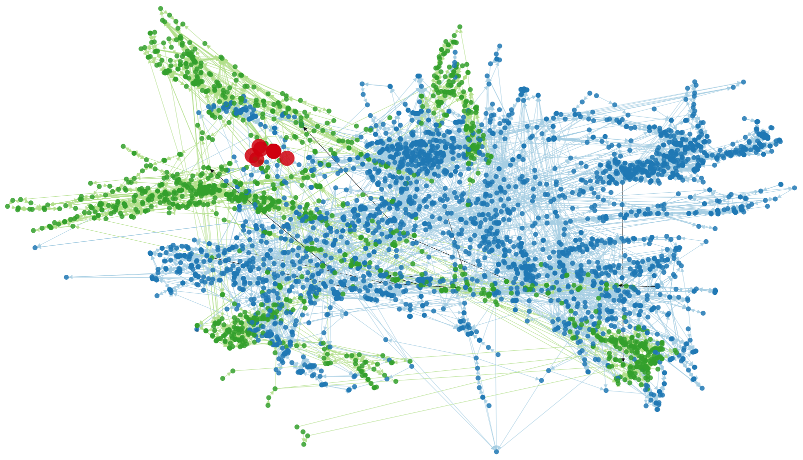

(c)

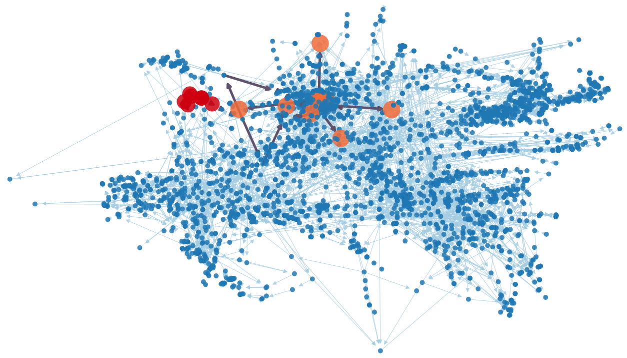
Figure 4.12: GOFFGRATCH, first iteration. The bug locations are indicated as large red nodes, and the top 10 most central variables in the blue (physics) community are indicated by larger orange nodes. Path segments from the bugs to the sampled central nodes are thicker purple edges.



(a)

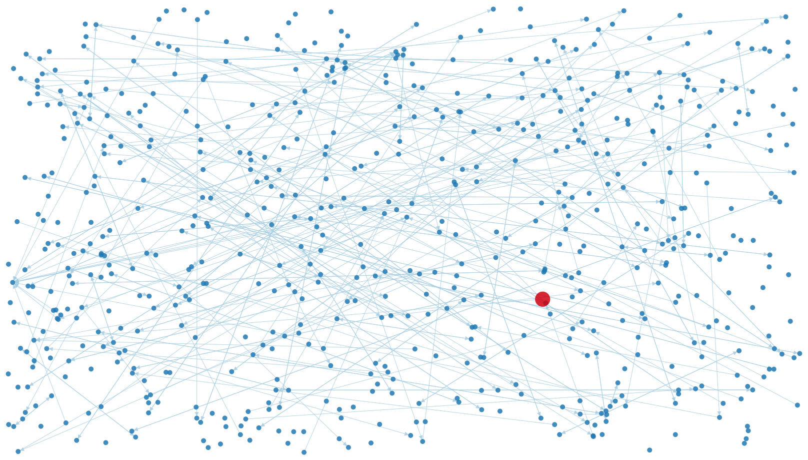


(b)

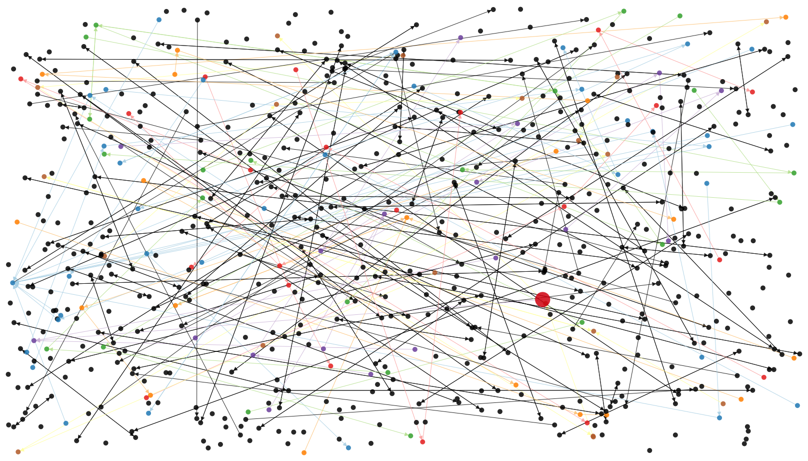


(c)

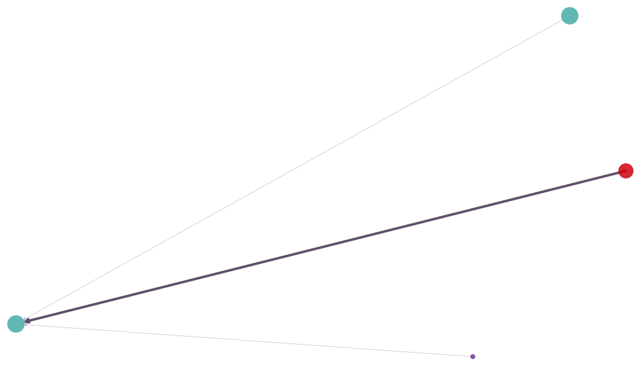
Figure 4.13: GOFFGRATCH, second iteration. The bug locations are indicated as large red nodes, and the top 10 most central variables in the blue community are indicated by larger orange nodes. Path segments from the bugs to the sampled central nodes are thicker purple edges. A few black edges can be seen in **b**, which correspond to edges removed by the G-N algorithm.



(a)



(b)



(c)

Figure 4.14: Randombug, single iteration. The bug location is indicated by a large red node. In **c**, the light blue nodes are the most central of the small community shown, and the purple edge designates the connection from the bug to the instrumented node.

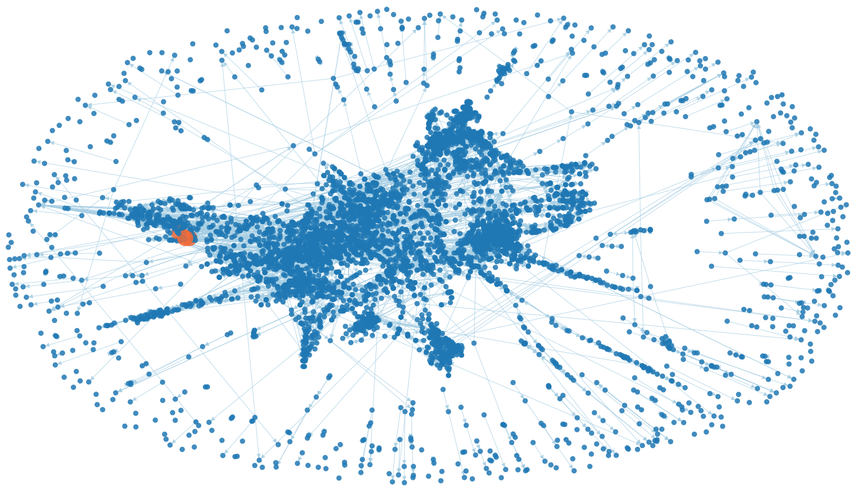
physics community. Instrumenting the light blue, most central nodes in Figure 4.15c would detect a difference in values between ensemble and experimental runs, as at least one instrumented node is reachable from the bugs. Inducing a subgraph on nodes contained in paths terminating on the central nodes connected to the bugs further reduces the size of the subgraph. In this way the subgraph will become small enough to instrument all nodes, or the exact bugs will be sampled.

#### 4.8.3.4 AVX2

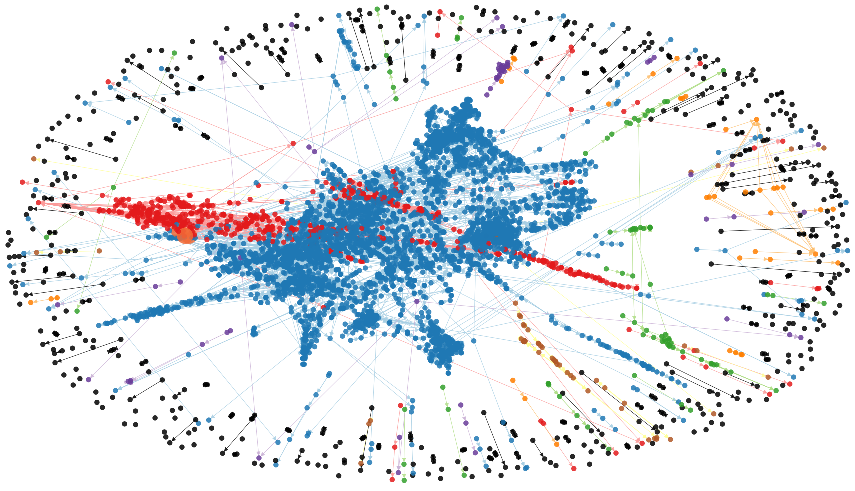
Figure 4.17 is an assertion that restricting our induced subgraph nodes to variables present in CAM is not necessary. This subgraph is created with the same affected variable list as Figure 4.17, but allows nodes outside of CAM (such as in the land model). Although the graph is larger (5,162 nodes and 9,873 edges), it manifests the community structure of the CAM core (orange cluster). Note that these communities are produced from two divisions of the G-N algorithm rather than our default one, as the models' structure is less evident with a single division. This suggests that the same conclusions are reached with this subgraph after a greater number of iterations.

### 4.8.4 Centrality output examples

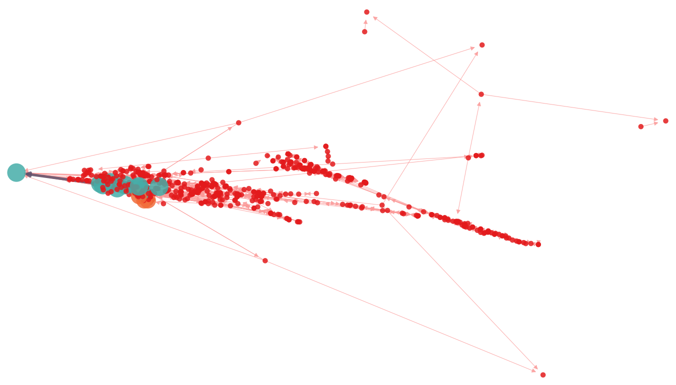
In this section, we provide in-centrality output to corroborate our assertion in paper Section 4.5.2, namely "Although sampling the whole core's most connected nodes may detect floating point differences between ensemble and experimental runs, instrumenting highly connected nodes in each community instead can reduce the distance between instrumented variables and bug locations (reducing the number of iterations needed to refine the search space)." In Appendix B.1, we include the output of the eigenvector in-centrality computed for the AVX2 subgraph. The output lists the top 500 most central nodes in the subgraph as `(node, centrality value)` tuples in descending order of centrality. Note that the node name displayed appends the name of the subprogram containing the variable to the name, which ensures unique node names in the graph. For example, the top node by centrality is `dum__micro_mg_tend`, which corresponds to `dum` in the `micro_mg_tend` subroutine (part of the MG1 microphysics kernel). There are a substantial number



(a)

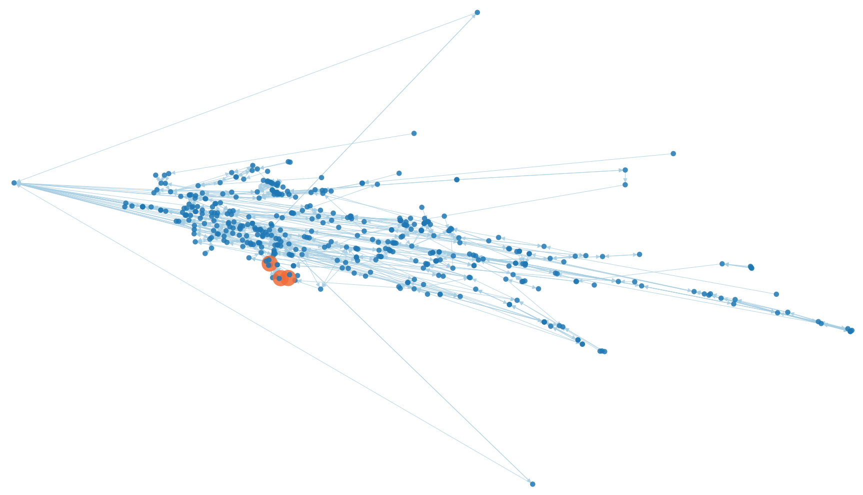


(b)

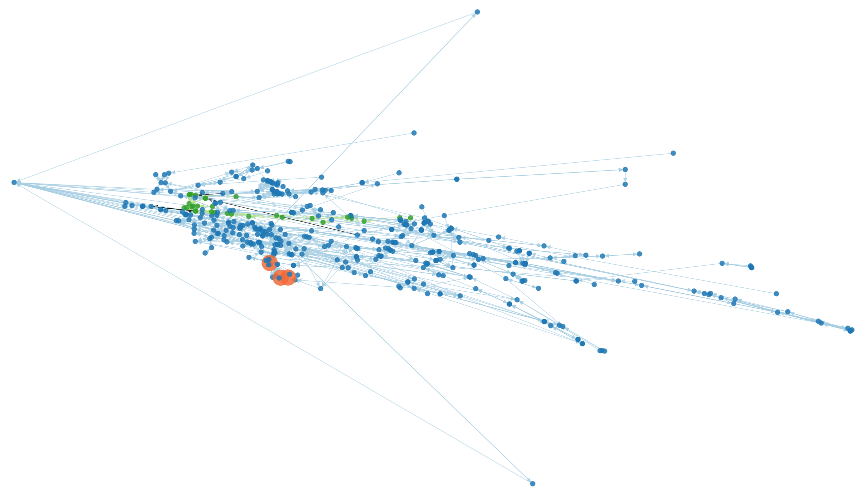


(c)

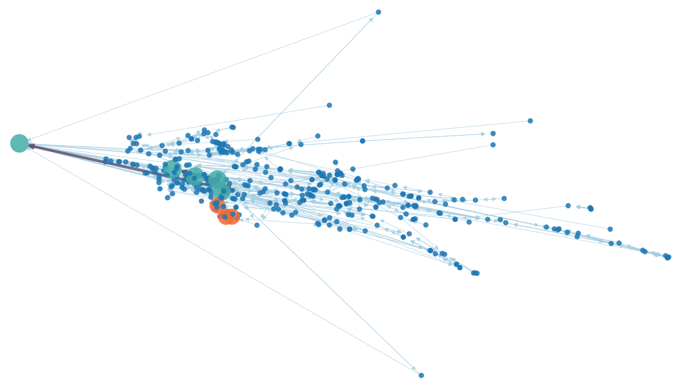
Figure 4.15: DYN3BUG first iteration. In these figures, the large orange nodes are the bugs, and the large, light blue nodes designate the most central nodes to be sampled.



(a)



(b)



(c)

Figure 4.16: DYN3BUG second iteration. In these figures, the large orange nodes are the bugs, and the large, light blue nodes designate the most central nodes to be sampled.

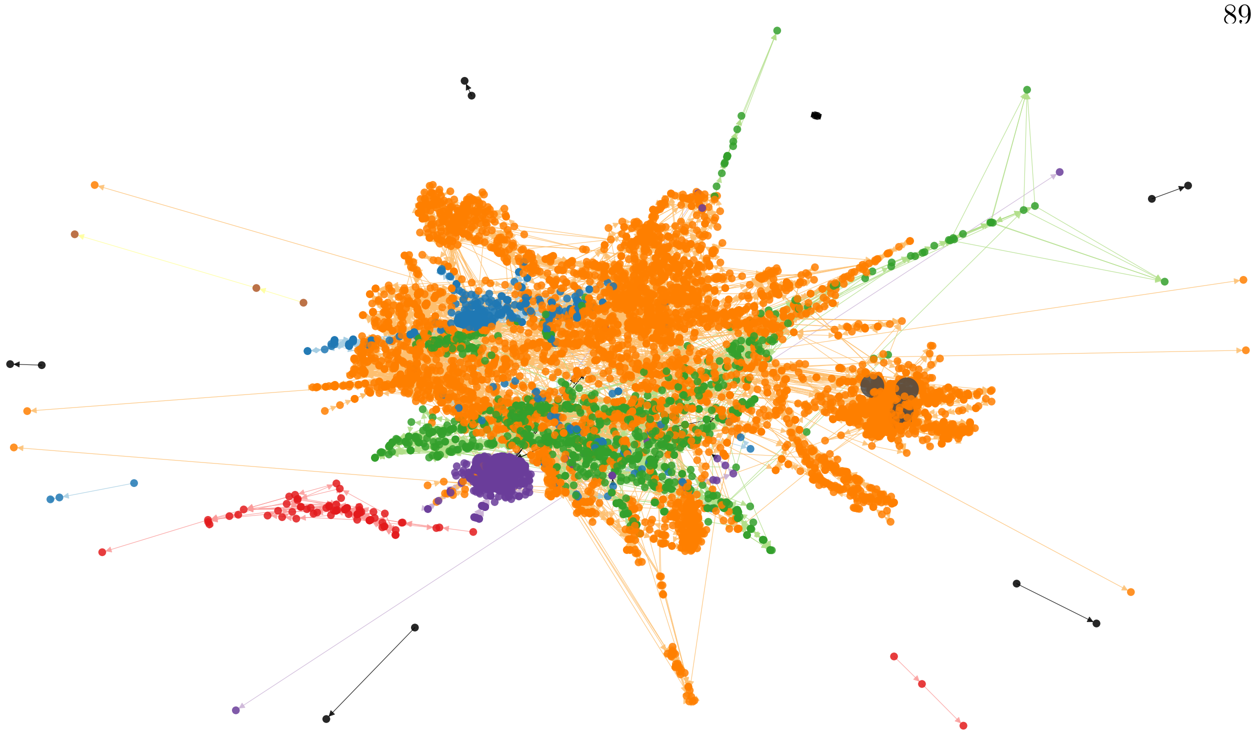


Figure 4.17: Communities generated by the induced subgraph defined by variables affected by the AVX2 experiment. Variable locations are not restricted to CAM. The variables identified by KGen are colored dark gray, and are among the top 15 most central in the orange community.

of nodes that appear like `min_line#__subprogram`. These nodes are Fortran procedures like `min` or `max`, which we introduce into the graph by creating paths from their inputs to themselves, and then to their outputs. This ensures correct dependency paths for these intrinsic procedures. Nearly all of the top 500 most central variables are located in the MG1 kernel. Nodes in MG1 mask the centralities of nodes in other modules, or clusters of modules.

Appendix B.2 lists the tuples in descending order of centrality for the first G-N community of the AVX2 subgraph. Notice that these tuples are very similar to those seen in the centrality output for the entire AVX2 subgraph. However, the second G-N community of the AVX2 subgraph (Appendix B.3), contains nodes not in MG1 or closely associated modules. In fact, the top nodes in the second community are not present in the top 500 nodes in the AVX2 subgraph, or in the first community. This supports our choice of partitioning the subgraph into communities with G-N, since we can search multiple possible connected clusters in parallel. Furthermore, if the sources of output discrepancy for AVX2 were in the second community, our sampled nodes would be closer to the sources. For the AVX2 experiment the sources are the most central node (and several others in the top 15), but for DYN3BUG (Section 4.8.3.3) the bugs are in the second community. This can reduce the number of iterations necessary to converge on the discrepancy sources.

# Chapter 5

# Applying our Tools: Adapting CAM Microphysics for Single Precision Computation

## 5.1 Introduction

Performing computations in single precision (32 bits or 4 bytes) is desirable from a performance standpoint. In comparison to double precision (64 bits), half the memory is needed to store each number, and arithmetic operations can be more than twice as fast (Baboulin et al., 2009). The disadvantage of using single precision is the increased potential for numerical instability, and its vastly smaller dynamic range. However, many algorithms remain stable (such as currently popular deep learning methods) in low precision (even half precision) and have no need for large range (S. Gupta et al., 2015). Adapting sections of CESM to run in single precision has been of interest for a long time, and the tools developed in this thesis can help achieve this goal.

The majority of CESM atmosphere model component, the Community Atmosphere Model (CAM), is devoted to the dynamical core (equations of motion) and physical processes. Physics computations cost approximately 70% of CAM runtime, all of which is currently computed with double-precision floating-point numbers. Many of the physics modules feature sub-grid scale parameterizations which are approximations that should not need the precision of a 16-decimal mantissa of 8-byte floats (Palmer, 2014; Váňa et al., 2017). Computing these physical processes with 4-byte floating-point numbers could translate to a substantial performance increase. The precision experiment from Sections 2.3.2, 3.5.1, and 4.6.3 is effectively a proof of concept test of using single precision for parameterized physics as `goffgratch_svp_water` is used in the CAM microphysics package. That the modified function yields statistically indistinguishable results provides promise that adapting larger sections of code may also produce statistically consistent output.

The Morrison-Gettelman microphysics version 2 (MG2) package performs sub-grid parameterizations of cloud microphysics processes such as "condensation/evaporation, freezing, melting, and sedimentation," aerosol physics, and precipitation for CAM (Morrison et al., 2008). See Figure 5.1 for a visual summary of CAM5 processes, including MG. Microphysics computations require a high degree of approximation, as cloud processes represent a large range of physical and temporal scales (Morrison et al., 2008). Large scale models cannot resolve individual clouds, let alone the physical processes within them. Therefore, models represent small-scale processes with distributions and make predictions of their moments, such as number concentration (Morrison et al., 2008). Parameterizations are expected to require fewer bits of precision to represent physical processes faithfully (Váňa et al., 2017). It is reasonable that converting the package to single precision can represent CAM cloud microphysics with sufficient fidelity to pass CESM-ECT. MG2 represents an appreciable portion of the CAM code, and 6% of the atmosphere model runtime. Moreover, converting MG2 to run in vectorized single precision translates to a speedup of 1.9 (on a specialized vector processor) versus double-precision MG2. UF-CAM-ECT and the graph analysis developed in this thesis used in concert with KGen (Kim et al., 2016; Kim et al., 2017) allow us to make notable performance gains and progress toward achieving statistical indistinguishability for MG2 in single precision. In this chapter, we describe our efforts to adapt MG2 to use 4-byte floats while preserving statistical consistency with our accepted UF-CAM-ECT ensemble.

## 5.2 Experimental Setup

MG2 consists of sections of seven Fortran modules and represents about 8,000 lines of code, including functions and subroutines used by other CAM modules. The subroutine that executes the microphysics processes is named `micro_mg_tend`, which is contained in `micro_mg2_0.F90`, and called in `micro_mg_cam.F90`. The MG2 tendency subroutine (`micro_mg_tend`) contains more than 300 local variables in 2,681 lines of code. Other modules and subprograms initialize the

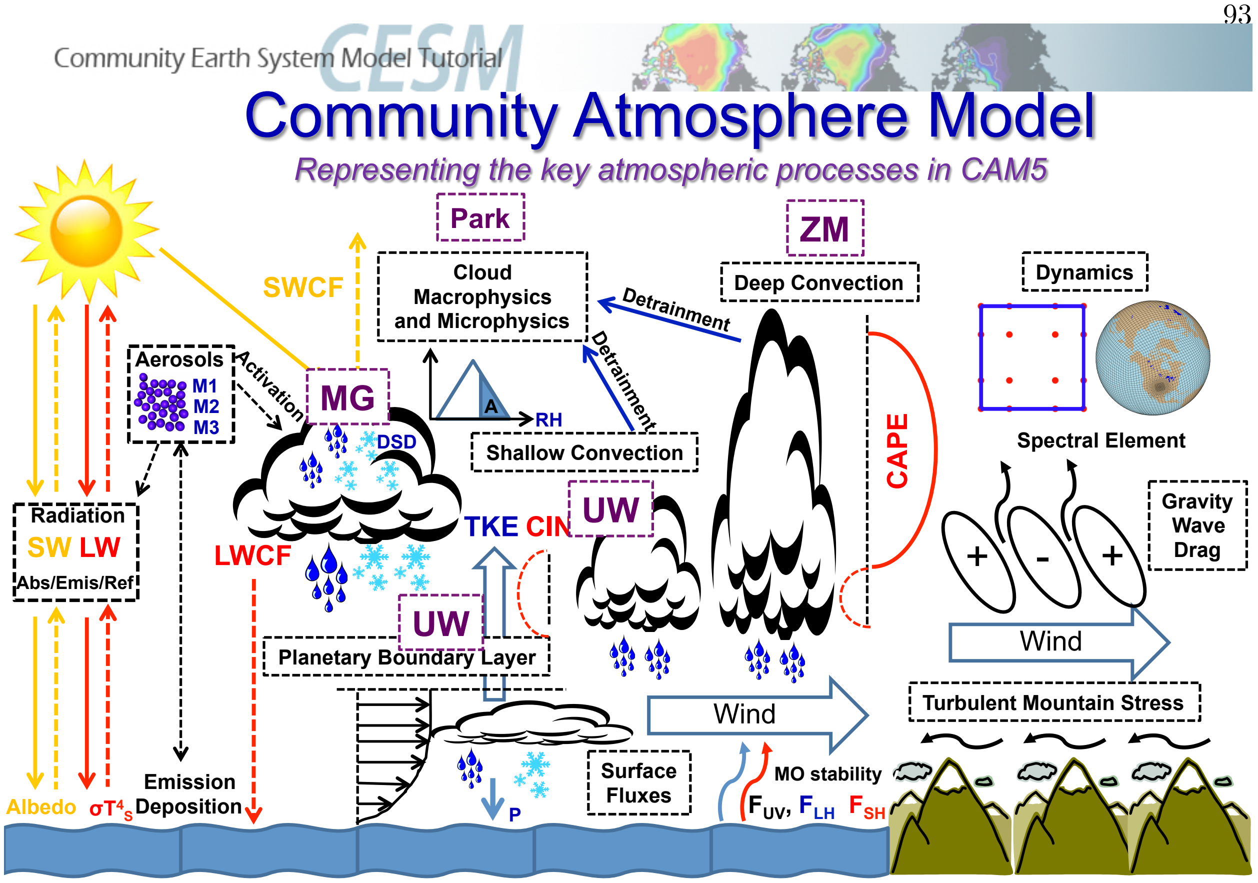


Figure 5.1: Schematic of CAM5 processes and subcomponents. The Morrison-Gettelman microphysics package is indicated by "MG". This figure appears in Neale, 2016 and is used with permission from the author.

package, read data from other parts of the model, set physical parameters and write variables to the CAM output files, and compute auxiliary physics. They also read user-defined modifications to CAM runtime behavior and broadcast them to all MPI ranks. MG2 relies on auxiliary modules that compute water vapor saturation processes (`wv_sat_methods.F90` which contains the Goff and Gratch SVP function mentioned in Sections 2.3.2, 3.5.1, and 4.6.3) and computes expensive transcendental functions like the gamma function and the error function (`shr_spfn_mod.F90`). MG2 also contains very small ($\sim 10^{-20}$) physical constants related to particle masses and mixing ratios which are present in numerous equations.

To expedite the process of identifying runtime errors and individual value discrepancies, we isolate and remove MG2 from the rest of the model through so-called kernel extraction. We use

KGen (Kim et al., 2016; Kim et al., 2017) to extract MG2 as a kernel from CESM and perform comparisons to data generated with MG2 integrated into CAM. KGen provides fine-grained testing of variable values, and compares Normalized Root Mean Square (NRMS) differences between data generated by the kernel when integrated into the model and values from the extracted kernel. KGen considers NRMS differences higher than a user specified threshold as failures.

UF-CAM-ECT facilitates rapid modification prototyping and consistency testing of the whole model, and we use DDT from ARM Forge (ARM, 2018) to trace variables and set watchpoints for divergent values. We proceed with the assumption that highly divergent values encountered in the extracted kernel correspond to statistical inconsistency as determined by UF-CAM-ECT. However, a UF-CAM-ECT *Pass* gives us confidence in slightly different, non-BFB values discovered by KGen.

Our experimental configuration consists of an experimental set of 10 simulations, run on two Cheyenne (CISL, 2017) nodes with 36 MPI processes (and no OpenMP threads) running on each node. We use the 2.0.0 release of CESM2, with CAM6 physics, CLM 5.0 with prognostic crops, sea ice (CICE) version 5 with prescribed ice, data ocean with prescribed ocean, MOSART (MOdel for Scale Adaptive River Transport), CISM2 (ice sheet model) with ice evolution off, and a stub wave component at year 2000 initialization time. We use a 2-degree finite volume CAM grid, with a 1 degree POP displaced pole grid (F2000climo component set and f19_g17_mg17 resolution). To facilitate debugging and tracing with DDT, we use the Single Column Atmosphere Model (SCAM) which runs on a single Cheyenne CPU to avoid the complexity of parallel debugging.

## 5.3 Converting MG2 to Single Precision

Converting the KGen-extracted MG2 kernel to single precision is straightforward and consists of substituting `real(kind=r4)` for `real(kind=r8)` in the extracted kernel. For example, `avg_diameter` is originally:

***Original avg_diameter***:

```
integer, parameter, public :: r8 = selected_real_kind(12)
! Pi to 20 digits; more than enough to reach the limit of double precision.
real(r8), parameter, public :: pi = 3.14159265358979323846_r8
real(r8) elemental function avg_diameter(q, n, rho_air, rho_sub)
  ! Finds the average diameter of particles given their density, and
  ! mass/number concentrations in the air.
  ! Assumes that diameter follows an exponential distribution.
  real(r8), intent(in) :: q         ! mass mixing ratio
  real(r8), intent(in) :: n         ! number concentration (per volume)
  real(r8), intent(in) :: rho_air   ! local density of the air
  real(r8), intent(in) :: rho_sub   ! density of the particle substance

  avg_diameter = (pi * rho_sub * n/(q*rho_air))**(-1._r8/3._r8)

end function avg_diameter
```

The conversion to 4-byte floats via direct substitution appears as follows:

***4-byte avg_diameter***:

```
integer, parameter, public :: r4 = selected_real_kind(6)
! Pi to 20 digits; far beyond the limit of single precision.
real(r4), parameter, public :: pi4 = 3.14159265358979323846_r4
real(r4) elemental function avg_diameter_r4(q, n, rho_air, rho_sub)
  real(r4), intent(in) :: q         ! mass mixing ratio
  real(r4), intent(in) :: n         ! number concentration (per volume)
  real(r4), intent(in) :: rho_air   ! local density of the air
  real(r4), intent(in) :: rho_sub   ! density of the particle substance

  avg_diameter = (pi4 * rho_sub * n/(q*rho_air))**(-1._r4/3._r4)

end function avg_diameter
```

Unfortunately, a naive, wholesale conversion of MG2 to 4-byte floats results in catastrophically different outputs as determined by KGen verification. Nine MG2 variables exhibit NRMS differences (which are relative) above a huge threshold of 1,000, with one variable manifesting NRMS differences exceeding $10^{11}$. To gain more information on possible sources of such tremendous differences, we run MG2 in single precision while integrated into CAM. Running the CESM with the single-precision MG2 variant can generate and help to identify floating-point exceptions, as vastly different values are output from MG2 to the rest of the model.

Converting MG2 to single precision integrated into CAM is similar to the conversion process for the MG2 kernel for modules and subprograms that are only used within MG2. However, for subprograms and modules that are also used outside of MG2 we must create separate 4-byte subprograms and modules. Inputs into the package must be demoted to single precision, and outputs promoted to double precision. Furthermore, we must track values initialized by auxiliary subprograms whose values are used outside of the main `micro_mg_tend` subroutine. After performing the necessary modifications, running CESM with single precision MG2 generates an FPERROR exception during the file output stage. More detailed analysis indicates that the output fields (in double precision) take values which are unrepresentable by the single-precision output arrays. Configuring CAM to write double-precision output eliminates that floating-point exception, but results in a 100% UF-CAM-ECT error rate. In particular, all PCs fail all 10 runs; an indication of egregious statistical difference. Deeper investigation requires using ARM DDT to debug the SCAM. This tool helps us identify a parameter that causes a divide-by-zero exception (`min_mean_mass` in the `MGHydrometeorProps` derived type) and an uninitialized value in `micro_mg_cam.F90`. Careful code inspection related to `min_mean_mass` computations also reveals applications of the Fortran transfer intrinsic function which may result in an incorrect bit pattern type conversion. We fix the former by declaring it as a double (and by modifying related subroutines and functions to take the double argument), and the latter by eliminating the transfer intrinsic in the KGen-extracted kernel. These alterations reduce the number of variables taking values above the NRMS threshold of 1,000 from nine to four, and reduce the number of UF-CAM-ECT PCs that fail from 50 to about

25 (which is still a 100% failure). The combination of different floating-point kinds is known as mixed precision, and it affords both advantages and disadvantages. Mixed precision computations for linear systems (Baboulin et al., 2009) and simple climate component models (e.g. Dawson et al., 2018; Thornes et al., 2018) have proven successful for producing accurate simulations with substantial speedups over double precision. The disadvantage of mixed precision is increased code complexity and reduced maintainability. In the following section, we take a step back and test the outcome of performing expensive MG2 calculations in single precision, while the rest of the package is computed in double precision.

## 5.4 Mixed Precision

As a first step toward statistically consistent full single precision in MG2, we modify two expensive sets of functions and subroutines used within MG2: gamma function computations and water vapor saturation calculations (in `shr_spfn_mod.F90` and `wv_sat_methods.F90`, respectively). In this case, MG2 effectively runs in mixed precision. Computing these functions and subroutines in single precision results in a 0% EET failure rate for UF-CAM-ECT, and considerable cost savings. Timing `micro_mg_tend` calls reports the maximum and minimum runtime of the call during the entire simulation execution. MG2 running in double precision results in an average maximum runtime of 0.410s per call, and an average minimum runtime of 0.371 (averaged over 33 $9^{th}$ time step "ultra-fast" runs). For the mixed precision runs, the MG2 average maximum runtime is 0.342s, and the average minimum runtime is 0.314s (average of 33 runs). Running MG2 in mixed precision represents an average maximum runtime reduction of 17% and an average minimum runtime decrease of 15% (see Table 5.1). We include an example of a subroutine from `wv_sat_methods.F90` called in `micro_mg2_0.F90`.

***Mixed precision subroutine call***:

```
USE wv_sat_methods, ONLY: qsat_water => wv_sat_qsat_water_r4, qsat_ice =>
    wv_sat_qsat_ice_r4
```

```
call qsat_water(t(i,k), p(i,k), esl(i,k), qvl(i,k))
elemental subroutine wv_sat_qsat_water_r4(t, p, es, qs, idx)
  !------------------------------------------------------------------!
  ! Purpose:                                                         !
  !   Calculate SVP over water at a given temperature, and then      !
  !   calculate and return saturation specific humidity.             !
  !------------------------------------------------------------------!
  ! Inputs
  real(r8), intent(in) :: t    ! Temperature
  real(r8), intent(in) :: p    ! Pressure
  ! Outputs
  real(r8), intent(out) :: es  ! Saturation vapor pressure
  real(r8), intent(out) :: qs  ! Saturation specific humidity
  integer,  intent(in), optional :: idx ! Scheme index
  ! single precision temp variables
  real(r4) :: t4
  real(r4) :: es4
  t4 = real(t)
  es4 = wv_sat_svp_water_r4(t4, idx)
  es = dble(es4)
```

From the standpoint of code comprehension, mixed precision is less desirable than uniform precision because it can result in unexpected behavior that is dependent on the compiler or CPU. For example, passing a single-precision argument to a subroutine whose input argument is declared as a double can result in undefined behavior. The compiler may catch an argument mismatch, but even in a simple assignment statement with mixed precision the promotion and/or demotion of types can cause confusion for programmers. Ideally, we can find a way to modify MG2 so that mixed precision arithmetic is minimized or eliminated.

Table 5.1: MG2 mixed precision results

| MG2 precision | Average max runtime (s) | Average min runtime (s) | UF-CAM-ECT fail. rate |
|---|---|---|---|
| double | 0.410 | 0.371 | 0.0% |
| mixed | 0.342 | 0.314 | 0.0% |

## 5.5 Next Steps

While additional optimization to single-precision gamma function and water vapor saturation process computations may further increase the speedup over double precision, our goal is to convert as much of MG2 as possible to run in single precision. Conversion to single precision involves a systematic analysis of variables that are highly divergent in the MG2 kernel. In Section 5.3 we reduced the number from nine to four, and with our graph analysis we may succeed in correcting the remaining four. Note that even if we succeed in correcting the final four variables, we will likely need to reduce the KGen failure threshold from its currently excessive value of 1,000 down to a much more reasonable value before we can expect the modified MG2 to pass UF-CAM-ECT when integrated into CESM. At the same time, we are working closely with the creators of MG2 to identify error sources using traditional methods. These methods feature customized SCAM output data analysis scripts designed specifically for MG2 applied by scientists with substantial climate system expertise. We intend to improve our automatic tools so that they make the same determinations as traditional methods.

To automate the reduction of the number of variables above the KGen failure threshold, we must improve our slicing algorithm from Section 4.5.1. The current method simply traverses paths in reverse without accounting for line number precedence, which does not resolve static paths well enough for detailed debugging. In other words, we must consider paths that move backward through the code, not merely through the graph. To permit such an approach, we embed a binary tree in each node and edge's metadata in the graph. Each binary tree contains the line numbers where the node or edge appears in the source code. The binary tree data structure allows fast lookups for line numbers greater than or less than (for forward and backward slicing, respectively) the current line. Current lines for every subprogram in the path are contained in a Python dictionary. These data structure choices allow rapid path traversal and pruning.

We adopt an optimistic approach for our first attempt at finding source code error causes of the four divergent variables by assuming that they share causes. The assumption allows us to perform path intersection rather than union and dramatically reduce the search space. The MG2 graph consists of 1971 nodes and 2931 edges with 30 subroutines and 197 functions. By intersecting all paths that terminate on the divergent four variables, we reduce the graph to 258 nodes in 30 subprograms. Several of these subprograms are unlikely sources as they are closely related to those already determined not to cause statistical inconsistency (e.g., `goffgratch_svp_water`). As a first step, we will convert all of these variables and subprograms to perform computations in double precision. If converting the flagged variables and subprograms to double precision fails, more substantial progress must be made to perform iterative sampling to realize Algorithm 4.5.4. We discuss implementation details in the next chapter.

# Chapter 6

# Future Work and Concluding Remarks

In this final chapter, we discuss future work and provide concluding remarks.

## 6.1 Future Work

We will complete the CESM-ECT suite by adding functional root cause analysis, which entails realizing Algorithm 4.5.4 for CESM. Broadly speaking, a practical implementation requires two parts. The first consists of building a tool to transform the induced subgraph into an accurate backwards slice with runtime data and to capture variable values for comparative analysis. The values of the most central variables in communities (described in Section 4.5.2) must be compared with a mechanism to detect meaningful difference. In this section, we discuss our approach to achieving iterative convergence of Algorithm 4.5.4 through software implementation of appropriate instrumentation techniques. In the interest of clarity, we reprint Algorithm 4.5.4 here. Note that G-N denotes the Girvan-Newman algorithm.

**Algorithm 6.1**

(1) Perform variable selection detailed in Section 4.3

(2) Map the set of affected CAM output variables in step 1 to their internal CAM variables $\{V_i\}$

(3) For each affected internal variable $V_i$, use BFS to find the set of nodes $\{n_{i_j}\}$ in all shortest paths that terminate on variables with canonical names equal to $V_i$ in the CESM digraph

(4) Form the induced subgraph $G$ via the union of nodes in the paths in step 3

(5) Use G-N to identify the communities $\{C_k\}$ of undirected $G$ (omitting communities smaller than 3 nodes)

(6) Compute the eigenvector in-centrality for each $C_k$ and select $m$ nodes with largest centrality $\{n_{k_l}\}$

(7) Instrument $\{n_{k_l}\}$ for all $k$ in parallel for an ensemble run and an experimental run, noting the set of nodes which take different values $\{d_{k_l}\}$ ($\{d_{k_l}\} \subseteq \{n_{k_l}\}$)

(8) (a) If $\{d_{k_l}\} = \emptyset$ (i.e., no different values are detected), form the induced subgraph on all nodes in $G$ that are not in BFS shortest paths that terminate on $\{n_{k_l}\}$

  (b) Else, form the induced subgraph of $G$ generated by nodes in $G$ that belong to BFS shortest paths that terminate on $\{d_{k_l}\}$

(9) Repeat steps 5-8 until the subgraph is small enough for manual analysis or the bug locations are instrumented

Algorithm 6.1, steps 1-6 produce a coarse slice of CESM, which offers the advantage of rapidly returning an induced subgraph and associated G-N communities. Its primary disadvantage is that the directed graph edges do not encode precedence, branching, or execution order. As mentioned in Section 4.5.2, this means that step 8a or 8b cannot guarantee a correct solution for the next induced subgraph. Providing a strict guarantee of correctness for subgraph resolution can be done with assiduous runtime instrumentation. In effect, we trade slicing accuracy for low complexity over the large model code, which translates the algorithmic cost of slicing into added sampling cost. Fortunately, this delayed cost can be mitigated. Before discussing cost savings details, we describe our proposed method of refining the coarse slice.

KGen is capable of automatically detecting variables that must be sampled to extract a source kernel. After determining the necessary variables, it modifies the source code by inserting calls to its sampling subroutines. Our needs are not as sophisticated: since Algorithm 6.1 gives us the set of variables in the form of an induced subgraph, we will adapt KGen to accept a list of variables and their corresponding code lines as input. The additional input will allow KGen to insert sampling calls in Fortran modules at the appropriate lines. The CESM digraph node and edge location metadata includes lines of code, so adapting KGen will be possible.

KGen will need two new types of instrumentation capability: simple binary sampling, and value tracing. By binary sampling, we mean detecting whether a particular edge is traversed at runtime. Edge detection is cheap, and can be done with a logical value that is set to true if the assignment statement is executed. This data can be written to file, which can be ingested by our graph tool and used to prune execution paths. With this data, more accurate backwards slices can be computed such that Algorithm 6.1 step 8 is guaranteed to return correct subgraphs for the next iteration.

The most central variables of each community are to be sampled by value at Algorithm 6.1 step 7. Values will be recorded for one control (ensemble) and one experimental run, for purposes of comparison in post-processing. Value instrumentation is more involved than binary sampling, and may require new techniques. During runtime, variables selected for sampling can be altered by assignments in loops, meaning that they may be subject to change a very large number of times. Storing each value in memory, to be written to disk incrementally, is not feasible in general. Furthermore, each central variable may be computed by a number of MPI processes (or OpenMP threads). To reduce space complexity, we propose to save the means and standard deviations of the most central variables. Saving means and standard deviations entails storing an auxiliary array composed of pairs for each variable, e.g. for element $x_{ij}$ of a 2-D array: $(\bar{x}_{ij}^{n-1}, \sigma_{ij}^{n-1})$ on each process or thread, and the update number $n$. The size of the array will not grow if the means and standard deviations are updated incrementally, using online methods. Computing these auxiliary values introduces quadratic overhead per instrumented array element, which should not be prohibitively expensive. Upon simulation completion, these auxiliary arrays can be written to a single netCDF file. Finally, post-processing will be done to combine means and standard deviations across processes and threads. With this data, we can determine which of the most central variables exhibit different behavior between the control and experimental runs.

To differentiate between the central variables' means and standard deviations from the control and experimental runs, we must determine adequate classification thresholds. Furthermore, we must conceive of a method to compare arrays of ordered pairs, rather than returning element-wise comparisons. Taking guidance from Baker et al., 2015, we can compute "global means" of each array, meaning that we calculate the average means and standard deviations for the array by row or column (or both). Another potential approach is to compute the normalized Root Mean Square (RMS) differences of means and standard deviations element-wise, in a similar fashion to the KGen verification step. Choosing a classification threshold for means and standard deviations can be done empirically, by using experiments from Sections 2.3 and 3.5 and inspecting the differences between control and experimental values. There are many practical ways to select appropriate classification boundaries, and a superior method may become evident once we generate the data.

In the case that several of the most central variables exhibit differences above the chosen threshold, we can add a condition to Algorithm 6.1 that will accelerate its convergence. Step 8 induces a smaller subgraph based on nodes that can affect the most different central variables. If we add a constraint that the induced subgraph be as small as possible (by ranking induced subgraph sizes returned by BFS shortest paths terminating on the central variables), we can remove more nodes as potential candidates for sources of inconsistency. When used with $9^{th}$ time step ("ultra-fast") runs and the UF-CAM-ECT, each step of the process requires little CPU time and facilitates parallelism via G-N community sampling (see Section 4.5.2). Depending on the induced subgraph structure, the iterative convergence can be approximately logarithmic in the number of nodes.

Algorithm 6.1 does not have a well-defined stopping criterion. Although the algorithm will converge on the sources, it is possible for the subgraph to become small enough that full tracing (storing each value at each iteration) can be enabled for every variable. KGen could be modified to perform this task by extracting a subgraph from the model instead of a kernel. Such an approach allows conversion of our directed subgraph into a true Directed Acyclic Graph (DAG) by capturing temporal precedence and dependencies. The DAG can be fast-forwarded or stepped backward in time to inspect value changes. Combined with assembly output, a correspondence between the source, computed values, and machine instructions can be made.

Implementing Algorithm 6.1 with guaranteed convergence will allow identification of the sources of statistical inconsistency in the CESM. While the testing and variable selection components of the full CESM-ECT quality assurance framework are particular to the CESM, the graph construction and analysis through Algorithm 6.1 can be used with any model written in Fortran. With a different parser, C and C++ can be analyzed in the same way. While implementation details may prove challenging, the method is abstracted from the programming language. We believe that our method is sufficiently general to permit root cause analysis of large-scale simulations, and intend to perform a series of studies to verify this claim.

## 6.2 Concluding Remarks

In this thesis, we make contributions to extend the CESM-ECT created in Baker et al., 2015. First we investigate the variational properties of the accepted ensemble by testing its classifications of modifications that should not produce statistically inconsistent results. These modifications include other CESM supported compilers and code modifications. We perform additional testing of ensemble size and composition and determine that the accepted ensemble should be larger. In particular, it should include a source of variation in addition to initial temperature field perturbations, i.e., the compiler. We conceive of a new test, based on an ensemble of $9^{th}$ time step (“ultra-fast”) runs, which retains most of the desirable properties of the 12-month test and possesses a capability that the longer test does not: detecting spatially localized changes. Furthermore, the ultra-fast test is much less expensive by a factor of approximately 70. By encoding relationships between CESM variables in a directed graph, we devise a method to iteratively converge on sources of statistical inconsistency present in the Fortran source code. We provide simulated examples of such analysis starting from actual model output and demonstrated the method’s convergence to the sources of discrepancy. We make substantive progress on applying these new methods and tools to the goal of modifying an expensive microphysics package (MG2) to perform computations with single precision floating point numbers. Converting only the most expensive computations to single precision yields between 15 and 17% runtime speedup (while retaining statistical indistinguishability) for the MG2 package; we anticipate more significant improvements as we convert additional MG2 code to single precision.

In Section 6.1, we delineate necessary steps for realizing Algorithm 4.5.4 and 6.1, and completing the CESM-ECT suite with an error source identification tool. We believe that this tool will be sufficiently general to trace inconsistencies back to sources in other models as well, and we intend to demonstrate this capability. The most promising part of the tool will be its representation of a model's internal error source DAG, complete with values at high temporal resolution. The time evolution of the complex system embedded into its graphical representation will allow for sophisticated study of its properties, potentially leading to greater insight into high-dimensional nonlinear and dynamical systems.

# Bibliography

Anderson, J., P. J. Burns, D. Milroy, P. Ruprecht, T. Hauser, and H. J. Siegel (2017). “Deploying RMACC Summit: An HPC Resource for the Rocky Mountain Region”. In: **Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact**. PEARC17. New Orleans, LA, USA: ACM, 8:1–8:7. ISBN: 978-1-4503-5272-7. DOI: 10.1145/3093338.3093379. URL: http://doi.acm.org/10.1145/3093338.3093379.

Argonne Leadership Computing Facility (2018). **Mira**. https://www.alcf.anl.gov/mira. Accessed: 2018-08-16.

ARM (2018). **Allinea Forge**. https://developer.arm.com/docs/101136/0701/allinea-forge. Accessed: 2018-08-19.

Baboulin, M., A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov (2009). “Accelerating scientific computations with mixed precision algorithms”. In: **Computer Physics Communications** 180.12. 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures, pp. 2526–2533. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2008.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0010465508003846.

Bailey, D. (2008). **Resolving Numerical Anomalies in Scientific Computation**. Tech. rep. LBNL-548E. Lawrence Berkeley National Laboratory.

Baker, A. H., D. M. Hammerling, M. N. Levy, H. Xu, J. M. Dennis, B. E. Eaton, J. Edwards, C. Hannay, S. A. Mickelson, R. B. Neale, D. Nychka, J. Shollenberger, J. Tribbia, M. Vertenstein, and D. Williamson (2015). “A new ensemble-based consistency test for the Community Earth System Model”. In: **Geoscientific Model Development** 8, pp. 2829–2840. DOI: 10.5194/gmd-8-2829-2015.

Baker, A. H., Y. Hu, D. M. Hammerling, Y.-H. Tseng, H. Xu, X. Huang, F. O. Bryan, and G. Yang (2016). “Evaluating statistical consistency in the ocean model component of the Community Earth System Model (pyCECT v2.0)”. In: **Geoscientific Model Development** 9.7, pp. 2391–2406. DOI: 10.5194/gmd-9-2391-2016. URL: http://www.geosci-model-dev.net/9/2391/2016/.

Bent, L., D. C. Atkinson, and W. G. Griswold (2001). **A Comparative Study of Two Whole Program Slicers for C**. Tech. rep. La Jolla, CA, USA.

Branstator, G. and H. Teng (2010). “Two Limits of Initial-Value Decadal Predictability in a CGCM”. In: **Journal of Climate** 23.23, pp. 6292–6311. DOI: 10.1175/2010JCLI3678.1. eprint: http://dx.doi.org/10.1175/2010JCLI3678.1. URL: http://dx.doi.org/10.1175/2010JCLI3678.1.

Clauset, A., S. Arbesman, and D. B. Larremore (2015). “Systematic inequality and hierarchy in faculty hiring networks”. In: **Science Advances** 1.1. DOI: 10.1126/sciadv.1400005. eprint: http://advances.sciencemag.org/content/1/1/e1400005.full.pdf. URL: http://advances.sciencemag.org/content/1/1/e1400005.

Clune, T. and R. Rood (2011). “Software testing and verification in climate model development”. In: **IEEE Software** 28, pp. 49–55. DOI: 10.1109/MS.2011.117.

Computational and Information Systems Laboratory (2016). **Yellowstone: IBM iDataPlex System (Climate Simulation Laboratory)**. http://n2t.net/ark:/85065/d7wd3xhc. Boulder, CO: National Center for Atmospheric Research.

Computational and Information Systems Laboratory (2017). **Cheyenne: SGI ICE XA Cluster**. https://www2.cisl.ucar.edu/resources/computational-systems/cheyenne/cheyenne. Boulder, CO: National Center for Atmospheric Research. DOI: 10.5065/d6rx99hx.

Craig, A. P., M. Vertenstein, and R. Jacob (2012). “A new flexible coupler for earth system modeling developed for CCSM4 and CESM1”. In: **The International Journal of High Performance Computing Applications** 26.1, pp. 31–42. DOI: 10.1177/1094342011428141. eprint: https://doi.org/10.1177/1094342011428141. URL: https://doi.org/10.1177/1094342011428141.

**D. J. Milroy**, A. H. Baker, D. M. Hammerling, Y. Kim, E. R. Jessup, and T. Hauser (arXiv preprint, 2018). **Making root cause analysis feasible for large code bases: a solution approach for a climate model**. arXiv: arXiv:1810.13432. URL: https://arxiv.org/abs/1810.13432.

Dawson, A., P. D. Düben, D. A. MacLeod, and T. N. Palmer (Oct. 2018). “Reliable low precision simulations in land surface models”. In: **Climate Dynamics** 51.7, pp. 2657–2666. ISSN: 1432-0894. DOI: 10.1007/s00382-017-4034-x. URL: https://doi.org/10.1007/s00382-017-4034-x.

Deser, C., A. Phillips, V. Bourdette, and H. Teng (2012). “Uncertainty in climate change projections: the role of internal variability”. In: **Climate Dynamics** 38.3, pp. 527–546. ISSN: 1432-0894. DOI: 10.1007/s00382-010-0977-x. URL: http://dx.doi.org/10.1007/s00382-010-0977-x.

Easterbrook, S. M. and T. C. Johns (2009). “Engineering the software for understanding climate change”. In: **Comput. Sci. Eng.** 11, pp. 65–74. DOI: doi:10.1109/MCSE.2009.193.

Eyring, V., S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor (2016). “Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization”. In: **Geoscientific Model Development** 9.5, pp. 1937–1958. DOI: 10.5194/gmd-9-1937-2016. URL: https://www.geosci-model-dev.net/9/1937/2016/.

Flato, G. M. (2011). “Earth system models: an overview”. In: **Wiley Interdisciplinary Reviews: Climate Change** 2.6, pp. 783–800. ISSN: 1757-7799. DOI: 10.1002/wcc.148. URL: http://dx.doi.org/10.1002/wcc.148.

Freeman, L. C. (1978). "Centrality in social networks conceptual clarification". In: **Social Networks** 1.3, pp. 215–239. ISSN: 0378-8733. DOI: https://doi.org/10.1016/0378-8733(78)90021-7. URL: http://www.sciencedirect.com/science/article/pii/0378873378900217.

Girvan, M. and M. E. J. Newman (2002). "Community structure in social and biological networks". In: **Proceedings of the National Academy of Sciences** 99.12, pp. 7821–7826. ISSN: 0027-8424. DOI: 10.1073/pnas.122653799. eprint: http://www.pnas.org/content/99/12/7821.full.pdf. URL: http://www.pnas.org/content/99/12/7821.

Gupta, R. and M. L. Soffa (1995). "Hybrid Slicing: An Approach for Refining Static Slices Using Dynamic Information". In: **Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering**. SIGSOFT '95. Washington, D.C., USA: ACM, pp. 29–40. ISBN: 0-89791-716-2. DOI: 10.1145/222124.222137. URL: http://doi.acm.org/10.1145/222124.222137.

Gupta, S., A. Agrawal, K. Gopalakrishnan, and P. Narayanan (2015). "Deep Learning with Limited Numerical Precision". In: **Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37**. ICML'15. Lille, France: JMLR.org, pp. 1737–1746. URL: http://dl.acm.org/citation.cfm?id=3045118.3045303.

Hagberg, A., P. Swart, and D. Schult (2008). "Exploring network structure, dynamics, and function using NetworkX". In: **Proceedings of the 7th Python in Science Conference (SciPy 2008)**, pp. 11–15.

Harris, W. R., S. Sankaranarayanan, F. Ivančić, and A. Gupta (2010). "Program Analysis via Satisfiability Modulo Path Programs". In: **Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**. POPL '10. Madrid, Spain: ACM, pp. 71–82. ISBN: 978-1-60558-479-9. DOI: 10.1145/1706299.1706309. URL: http://doi.acm.org/10.1145/1706299.1706309.

Hashimoto, K. and A. Hori (1989). "Selberg-Ihara's Zeta function for p-adic Discrete Groups". In: **Automorphic Forms and Geometry of Arithmetic Varieties**. Ed. by K. Hashimoto and

Y. Namikawa. Vol. 15. Advanced Studies in Pure Mathematics. Academic Press, pp. 171–210. ISBN: 978-0-12-330580-0. DOI: https://doi.org/10.1016/B978-0-12-330580-0.50014-8. URL: http://www.sciencedirect.com/science/article/pii/B9780123305800500148.

He, Y. and C. Ding (2001). "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications". In: **Journal of Supercomputing** 18.3, pp. 259–277.

Heuvel, M. P. van den and O. Sporns (Dec. 2013). "Network hubs in the human brain". In: **Trends in Cognitive Sciences** 17.12, pp. 683–696. ISSN: 1364-6613. DOI: 10.1016/j.tics.2013.09.012. URL: http://dx.doi.org/10.1016/j.tics.2013.09.012.

Hurrell, J., M. Holland, P. Gent, S. Ghan, J. Kay, P. Kushner, J.-F. Lamarque, W. Large, D. Lawrence, K. Lindsay, W. Lipscomb, M. Long, N. Mahowald, D. Marsh, R. Neale, P. Rasch, S. Vavrus, M. Vertenstein, D. Bader, W. Collins, J. Hack, J. Kiehl, and S. Marshall (2013). "The Community Earth System Model: A Framework for Collaborative Research". In: **Bulletin of the American Meteorological Society** 94, pp. 1339–1360. DOI: 10.1175/BAMS-D-12-00121.1.

Intel (2017). **Intel Fortran Compiler 17.0 Developer Guide and Reference Code Coverage Tool**. https://software.intel.com/en-us/node/680224. Accessed: 2018-01-24.

IPCC (2013). "Summary for Policymakers". In: **Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change**. Ed. by T. Stocker, D. Qin, G.-K. Plattner, M. Tignor, S. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex, and P. Midgley. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press. Chap. SPM, pp. 1–30. ISBN: ISBN 978-1-107-66182-0. DOI: 10.1017/CBO9781107415324.004. URL: www.climatechange2013.org.

Jaffar, J., V. Murali, J. A. Navas, and A. E. Santosa (2012). "Path-Sensitive Backward Slicing". In: **Proceedings of the 19th International Conference on Static Analysis**. SAS'12.

Deauville, France: Springer-Verlag, pp. 231–247. ISBN: 978-3-642-33124-4. DOI: 10.1007/978-3-642-33125-1_17. URL: http://dx.doi.org/10.1007/978-3-642-33125-1_17.

Kay, J. E., C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. C. Bates, G. Danabasoglu, J. Edwards, M. Holland, P. Kushner, J.-F. Lamarque, D. Lawrence, K. Lindsay, A. Middleton, E. Munoz, R. Neale, K. Oleson, L. Polvani, and M. Vertenstein (2015). "The Community Earth System Model (CESM) Large Ensemble Project: A Community Resource for Studying Climate Change in the Presence of Internal Climate Variability". In: **Bulletin of the American Meteorological Society** 96.8, pp. 1333–1349. DOI: 10.1175/BAMS-D-13-00255.1. eprint: http://dx.doi.org/10.1175/BAMS-D-13-00255.1. URL: http://dx.doi.org/10.1175/BAMS-D-13-00255.1.

Kim, Y., J. M. Dennis, and C. Kerr (Sept. 2017). "Assessing Representativeness of Kernels Using Descriptive Statistics". In: **2017 IEEE International Conference on Cluster Computing (CLUSTER)**, pp. 818–825. DOI: 10.1109/CLUSTER.2017.117.

Kim, Y., J. Dennis, C. Kerr, R. R. P. Kumar, A. Simha, A. Baker, and S. Mickelson (2016). "KGEN: A Python Tool for Automated Fortran Kernel Generation and Verification". In: **Procedia Computer Science**. Vol. 80. ICCS 2016. The International Conference on Computational Science, pp. 1450–1460. DOI: 10.1016/j.procs.2016.05.466.

Lattner, C. and V. Adve (2004). "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: **Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization**. CGO '04. Palo Alto, California: IEEE Computer Society, pp. 75–. ISBN: 0-7695-2102-9. URL: http://dl.acm.org/citation.cfm?id=977395.977673.

Martin, T., X. Zhang, and M. E. J. Newman (Nov. 2014). "Localization and centrality in networks". In: **Phys. Rev. E** 90 (5), p. 052808. DOI: 10.1103/PhysRevE.90.052808. URL: https://link.aps.org/doi/10.1103/PhysRevE.90.052808.

McCabe, T. J. (Dec. 1976). "A Complexity Measure". In: **IEEE Transactions on Software Engineering** SE-2.4, pp. 308–320. ISSN: 0098-5589. DOI: 10.1109/TSE.1976.233837.

Méndez, M., F. G. Tinetti, and J. L. Overbey (Nov. 2014). “Climate Models: Challenges for Fortran Development Tools”. In: **2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering**, pp. 6–12. DOI: 10.1109/SE-HPCCSE.2014.7.

Milroy, D. J., A. H. Baker, D. M. Hammerling, and E. R. Jessup (2018). “Nine time steps: ultra-fast statistical consistency testing of the Community Earth System Model (pyCECT v3.0)”. In: **Geoscientific Model Development** 11.2, pp. 697–711. DOI: 10.5194/gmd-11-697-2018. URL: https://www.geosci-model-dev.net/11/697/2018/.

Milroy, D. J. (2015). “Refining the composition of CESM-ECT ensembles”. English. In: **ProQuest Dissertations and Theses**. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2016-03-21, p. 60. URL: http://search.proquest.com/docview/1760171657?accountid=28174.

Milroy, D. J., A. H. Baker, D. M. Hammerling, J. M. Dennis, S. A. Mickelson, and E. R. Jessup (2016). “Towards Characterizing the Variability of Statistically Consistent Community Earth System Model Simulations”. In: **Procedia Computer Science** 80.Supplement C. International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, pp. 1589–1600. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.05.489. URL: http://www.sciencedirect.com/science/article/pii/S1877050916309759.

Morrison, H. and A. Gettelman (2008). “A New Two-Moment Bulk Stratiform Cloud Microphysics Scheme in the Community Atmosphere Model, Version 3 (CAM3). Part I: Description and Numerical Tests”. In: **Journal of Climate** 21.15, pp. 3642–3659. DOI: 10.1175/2008JCLI2105.1. eprint: https://doi.org/10.1175/2008JCLI2105.1. URL: https://doi.org/10.1175/2008JCLI2105.1.

Neale, R. (Aug. 2016). **Atmospheric Modeling I: Physics in the Community Atmosphere Model (CAM)**. http://www.cesm.ucar.edu/events/tutorials/2016/lecture2-neale.pdf. Accessed: 2018-10-18.

Newman, M. E. J. and M. Girvan (Feb. 2004). “Finding and evaluating community structure in networks”. In: **Phys. Rev. E** 69 (2), p. 026113. DOI: 10.1103/PhysRevE.69.026113. URL: https://link.aps.org/doi/10.1103/PhysRevE.69.026113.

Oleson, K. W., D. M. Lawrence, G. B, M. G. Flanner, E. Kluzek, P. J, S. Levis, S. C. Swenson, E. Thornton, J. Feddema, C. L. Heald, J. Lamarque, G. Niu, T. Qian, S. Running, K. Sakaguchi, L. Yang, X. Zeng, and X. Zeng (2010). **Technical Description of version 4.0 of the Community Land Model (CLM)**. Tech. rep.

Page, L., S. Brin, R. Motwani, and T. Winograd (Nov. 1999). **The PageRank Citation Ranking: Bringing Order to the Web.** Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab. URL: http://ilpubs.stanford.edu:8090/422/.

Palmer, T. N. (2014). “More reliable forecasts with less precise computations: a fast-track route to cloud-resolved weather and climate simulators?” In: **Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences** 372.2018. ISSN: 1364-503X. DOI: 10.1098/rsta.2013.0391. eprint: http://rsta.royalsocietypublishing.org/content/372/2018/20130391.full.pdf. URL: http://rsta.royalsocietypublishing.org/content/372/2018/20130391.

Peterson, P. (2009). “F2PY: a tool for connecting Fortran and Python programs”. In: **International Journal of Computational Science and Engineering** 4.4, pp. 296–305. DOI: 10.1504/IJCSE.2009.029165. eprint: https://www.inderscienceonline.com/doi/pdf/10.1504/IJCSE.2009.029165. URL: https://www.inderscienceonline.com/doi/abs/10.1504/IJCSE.2009.029165.

PGI (2017). **Flang**. https://github.com/flang-compiler/flang. Accessed: 2018-08-14.

Pipitone, J. and S. Easterbrook (2012). “Assessing climate model software quality: a defect density analysis of three models”. In: **Geoscientific Model Development** 5.4, pp. 1009–1022. DOI: 10.5194/gmd-5-1009-2012.

RogueWave Software (2018). **TotalView**. https://support.roguewave.com/documentation/tvdocs/en/2018/. Accessed: 2018-08-19.

Rosinski, J. M. and D. L. Williamson (Mar. 1997). “The Accumulation of Rounding Errors and Port Validation for Global Atmospheric Models”. In: **SIAM J. Sci. Comput.** 18.2, pp. 552–564. ISSN: 1064-8275. DOI: `10.1137/S1064827594275534`. URL: `http://dx.doi.org/10.1137/S1064827594275534`.

Salathé, M., M. Kazandjieva, J. W. Lee, P. Levis, M. W. Feldman, and J. H. Jones (Dec. 2010). “A high-resolution human contact network for infectious disease transmission”. In: **Proc Natl Acad Sci USA** 107.51. 21149721[pmid], pp. 22020–22025. ISSN: 0027-8424. DOI: `10.1073/pnas.1009094108`. URL: `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3009790/`.

Sawaya, G., M. Bentley, I. Briggs, G. Gopalakrishnan, and D. H. Ahn (Oct. 2017). “FLiT: Cross-platform floating-point result-consistency tester and workload”. In: **2017 IEEE International Symposium on Workload Characterization (IISWC)**, pp. 229–238. DOI: `10.1109/IISWC.2017.8167780`.

Shah, D. and T. Zaman (June 2010). “Detecting Sources of Computer Viruses in Networks: Theory and Experiment”. In: **SIGMETRICS Perform. Eval. Rev.** 38.1, pp. 203–214. ISSN: 0163-5999. DOI: `10.1145/1811099.1811063`. URL: `http://doi.acm.org/10.1145/1811099.1811063`.

Shende, S. S. and A. D. Malony (May 2006). “The Tau Parallel Performance System”. In: **Int. J. High Perform. Comput. Appl.** 20.2, pp. 287–311. ISSN: 1094-3420. DOI: `10.1177/1094342006064482`. URL: `http://dx.doi.org/10.1177/1094342006064482`.

Silva, J. (June 2012). “A Vocabulary of Program Slicing-based Techniques”. In: **ACM Comput. Surv.** 44.3, 12:1–12:41. ISSN: 0360-0300. DOI: `10.1145/2187671.2187674`. URL: `http://doi.acm.org/10.1145/2187671.2187674`.

Stodden, V., J. Borwein, and D. Bailey (2013). “Setting the default to reproducible in computational science research”. In: **SIAM News**.

Thornes, T., P. Düben, and T. Palmer (2018). “A power law for reduced precision at small spatial scales: Experiments with an SQG model”. In: **Quarterly Journal of the Royal Meteorological Society** 144.713, pp. 1179–1188. DOI: `10.1002/qj.3303`. eprint: `https:`

//rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.3303. URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.3303.

Tip, F. (1994). **A Survey of Program Slicing Techniques.** Tech. rep. Amsterdam, The Netherlands, The Netherlands.

University of Illinois/NCSA (2007). **Clang**. http://clang.llvm.org/. Accessed: 2018-08-16.

Váňa, F., P. Düben, S. Lang, T. Palmer, M. Leutbecher, D. Salmond, and G. Carver (2017). "Single Precision in Weather Forecasting Models: An Evaluation with the IFS". In: **Monthly Weather Review** 145.2, pp. 495–502. DOI: 10.1175/MWR-D-16-0228.1. eprint: https://doi.org/10.1175/MWR-D-16-0228.1. URL: https://doi.org/10.1175/MWR-D-16-0228.1.

Wan, H., K. Zhang, P. J. Rasch, B. Singh, X. Chen, and J. Edwards (2017). "A new and inexpensive non-bit-for-bit solution reproducibility test based on time step convergence (TSC1.0)". In: **Geoscientific Model Development** 10.2, pp. 537–552. DOI: 10.5194/gmd-10-537-2017. URL: http://www.geosci-model-dev.net/10/537/2017/.

Weiser, M. (July 1984). "Program Slicing". In: **IEEE Transactions on Software Engineering** SE-10.4, pp. 352–357. ISSN: 0098-5589. DOI: 10.1109/TSE.1984.5010248.

Weiser, M. (1981). "Program Slicing". In: **Proceedings of the 5th International Conference on Software Engineering**. ICSE '81. San Diego, California, USA: IEEE Press, pp. 439–449. ISBN: 0-89791-146-6. URL: http://dl.acm.org/citation.cfm?id=800078.802557.

# Appendix A

## Experiments from Baker et al., 2015

### A.1 Experiment Names

- NO-OPT: changing the Intel compiler flag to remove optimization
- INTEL-15: changing the Intel compiler version to 15.0.0
- NO-THRD: compiling CAM without threading (MPI-only)
- PGI: using the CESM-supported PGI compiler (13.0)
- GNU: using the CESM-supported GNU compiler (4.8.0)
- EDISON: National Energy Research Scientific Computing Center (Cray XC30, Intel)
- DUST: dust emissions; dust_emis_fact = 0.45 (original default 0.55)
- FACTB: wet deposition of aerosols convection factor; sol_factb_interstitial = 1.0 (original default 0.1)
- FACTIC: wet deposition of aerosols convection factor; sol_factic_interstitial = 1.0 (original default 0.4)
- RH-MIN-LOW: min. relative humidity for low clouds; cldfrc_rhminl = 0.85 (original default 0.8975)
- RH-MIN-HIGH: min. relative humidity for high clouds; cldfrc_rhminh = 0.9 (original default 0.8)

- CLDFRC-DP: deep convection cloud fraction; cld_frc_dp1 = 0.14 (original default 0.10)
- UW-SH: penetrative entrainment efficiency - shallow; uwschu_rpen = 10.0 (original default 5.0)
- CONV-LND: autoconversion over land in deep convection; zmconv_c0_lnd = 0.0035 (original default 0.0059)
- CONV-OCN: autoconversion over ocean in deep convection; zmconv_c0_ocn = 0.0035 (original default 0.045)
- NU-P: hyperviscosity for layer thickness (vertical lagrangian dynamics); nu_p = $1 \times 10^{14}$ (original default $1 \times 10^{15}$)
- NU: dynamics hyperviscosity (horizontal diffusion); nu = $9 \times 10^{14}$ (original default $1 \times 10^{15}$)

# Appendix B

# Centrality Output

## B.1 AVX2 Subgraph Centrality

```
>>> avx2_subgraph_incentrality[:500]
(dum__micro_mg_tend, 0.455153), (ratio__micro_mg_tend, 0.325264),
(tlat__micro_mg_tend, 0.255383), (qniic__micro_mg_tend, 0.198578),
(nric__micro_mg_tend, 0.196431), (nsic__micro_mg_tend, 0.191075),
(qctend__micro_mg_tend, 0.188477), (qric__micro_mg_tend, 0.180318),
(qitend__micro_mg_tend, 0.15969), (prds__micro_mg_tend, 0.157626),
(pre__micro_mg_tend, 0.157551), (nctend__micro_mg_tend, 0.148088),
(qvlat__micro_mg_tend, 0.132584), (mnuccc__micro_mg_tend, 0.121525),
(nitend__micro_mg_tend, 0.120172), (nsagg__micro_mg_tend, 0.109382),
(nragg__micro_mg_tend, 0.107409), (dum1__micro_mg_tend, 0.0964774),
(qnitend__micro_mg_tend, 0.0875948), (nnuccr__micro_mg_tend, 0.0875681),
(mnuccr__micro_mg_tend, 0.0875681), (npracs__micro_mg_tend, 0.0834417),
(pracs__micro_mg_tend, 0.0834417), (tmp__micro_mg_tend, 0.0822866),
(nrtot__micro_mg_tend, 0.0815155), (qstot__micro_mg_tend, 0.0815155),
(nstot__micro_mg_tend, 0.0815155), (qrtot__micro_mg_tend, 0.0815155),
(ttmp__micro_mg_tend, 0.0814203), (min__2140__micro_mg_tend, 0.0807908),
(min__2122__micro_mg_tend, 0.080781), (psacws__micro_mg_tend, 0.0781575),
(npsacws__micro_mg_tend, 0.0773532), (pra__micro_mg_tend, 0.0767817),
(qrtend__micro_mg_tend, 0.0693434), (nprai__micro_mg_tend, 0.0670967),
(prai__micro_mg_tend, 0.0669185), (qclr__micro_mg_tend, 0.0627478),
(qtmp__micro_mg_tend, 0.0627211), (eci__micro_mg_tend, 0.0621703),
(npra__micro_mg_tend, 0.0621291), (faltndc__micro_mg_tend, 0.0607809),
(faltndnc__micro_mg_tend, 0.0601965), (min__2559__micro_mg_tend, 0.0600089),
(max__3120__micro_mg_tend, 0.0600089), (max__2558__micro_mg_tend, 0.0600089),
(min__2177__micro_mg_tend, 0.0600089), (min__2525__micro_mg_tend, 0.0600089),
(min__3004__micro_mg_tend, 0.0600089), (max__2524__micro_mg_tend, 0.0600089),
(max__3087__micro_mg_tend, 0.0600089), (min__3088__micro_mg_tend, 0.0600089),
(min__3121__micro_mg_tend, 0.0600089), (min__2168__micro_mg_tend, 0.0600089),
(max__1158__micro_mg_tend, 0.0600089), (nprc1__micro_mg_tend, 0.0582542),
(n0s__micro_mg_tend, 0.0581463), (nsubi__micro_mg_tend, 0.0576126),
(n0r__micro_mg_tend, 0.0562208), (nnucct__micro_mg_tend, 0.0552363),
(mnucct__micro_mg_tend, 0.0552363), (msacwi__micro_mg_tend, 0.0531491),
(nnuccc__micro_mg_tend, 0.052809), (bergs__micro_mg_tend, 0.0520531),
(lams__micro_mg_tend, 0.0513731), (prc__micro_mg_tend, 0.0512664),
(prci__micro_mg_tend, 0.051026), (nprci__micro_mg_tend, 0.0509864),
(nrtend__micro_mg_tend, 0.0508104), (lamr__micro_mg_tend, 0.0496719),
(nsubs__micro_mg_tend, 0.0493965), (nsubc__micro_mg_tend, 0.0493965),
(nsubr__micro_mg_tend, 0.0493965), (nstend__micro_mg_tend, 0.0451548),
(dumc__micro_mg_tend, 0.0435571), (t__micro_mg_tend, 0.039804),
(tlat1__micro_mg_tend, 0.038784), (qc__micro_mg_tend, 0.0318119),
(dumi__micro_mg_tend, 0.0310279), (qsout__micro_mg_tend, 0.0301572),
(qctend1__micro_mg_tend, 0.0286232), (qrout__micro_mg_tend, 0.0273842),
(qi__micro_mg_tend, 0.0272577), (max__1722__micro_mg_tend, 0.0258981),
(max__2596__micro_mg_tend, 0.0258981), (nc__micro_mg_tend, 0.0254224),
(dumfice__micro_mg_tend, 0.0252079), (rainrt__micro_mg_tend, 0.0251983),
(max__2597__micro_mg_tend, 0.025192), (max__1723__micro_mg_tend, 0.025192),
(qitend1__micro_mg_tend, 0.0242514), (nctend1__micro_mg_tend, 0.0224895),
(q__micro_mg_tend, 0.020774), (qvlat1__micro_mg_tend, 0.020135),
(ni__micro_mg_tend, 0.0196537), (nitend1__micro_mg_tend, 0.0182501),
(epss__micro_mg_tend, 0.0168008), (epsr__micro_mg_tend, 0.0163226),
(t__svp_water, 0.0159826), (t__svp_ice, 0.0159826),
(ni_secp__micro_mg_tend, 0.0155524), (nfice__micro_mg_tend, 0.0149841),
(faltndi__micro_mg_tend, 0.0132731), (faltndni__micro_mg_tend, 0.0127688),
(min__3006__micro_mg_tend, 0.0127199), (max__1969__micro_mg_tend, 0.012355),
(min__1963__micro_mg_tend, 0.012355), (min__1959__micro_mg_tend, 0.012355),
(state_loc%q__micro_mg_cam_tend, 0.0111397), (prd__micro_mg_tend, 0.0100962),
(dumnc__micro_mg_tend, 0.00993771), (lamc__micro_mg_tend, 0.0083619),
(min__1932__micro_mg_tend, 0.00819674), (max__1931__micro_mg_tend, 0.00819674),
(cmei__micro_mg_tend, 0.00718911), (ncic__micro_mg_tend, 0.0070591),
(min__1785__micro_mg_tend, 0.00678523), (min__2680__micro_mg_tend, 0.00678523),
(min__2681__micro_mg_tend, 0.00678523), (min__1784__micro_mg_tend, 0.00678523),
(ds0__micro_mg_tend, 0.0067732), (nprc__micro_mg_tend, 0.00675914),
(min__1751__micro_mg_tend, 0.00656094), (min__1752__micro_mg_tend, 0.00656094),
(min__2628__micro_mg_tend, 0.00656094), (min__2627__micro_mg_tend, 0.00656094),
(ndfaer2__micro_mg_tend, 0.00596703), (ndfaer1__micro_mg_tend, 0.00596703),
(ndfaer4__micro_mg_tend, 0.00596703), (ndfaer3__micro_mg_tend, 0.00596703),
(faloutc__micro_mg_tend, 0.00585526), (min__3194__micro_mg_tend, 0.00574271),
(qc__micro_mg_cam_tend, 0.00566289), (berg__micro_mg_tend, 0.00547565),
(qie__micro_mg_tend, 0.00526351), (dqsdt__micro_mg_tend, 0.0052479),
(dqsidt__micro_mg_tend, 0.0052479), (mu__micro_mg_tend, 0.0052479),
(exp__2002__micro_mg_tend, 0.0052479), (dv__micro_mg_tend, 0.0052479),
(sqrt__1838__micro_mg_tend, 0.0052479), (t1__micro_mg_tend, 0.0052479),
(exp__1807__micro_mg_tend, 0.0052479), (exp__1820__micro_mg_tend, 0.0052479),
(tcnt__micro_mg_tend, 0.0052479), (exp__1813__micro_mg_tend, 0.0052479),
(exp__2006__micro_mg_tend, 0.0052479), (exp__1826__micro_mg_tend, 0.0052479),
(viscosity__micro_mg_tend, 0.0052479), (rho__micro_mg_tend, 0.0052479),
(qi__micro_mg_cam_tend, 0.00506245), (n0i__micro_mg_tend, 0.00501755),
(qce__micro_mg_tend, 0.00491612), (nc__micro_mg_cam_tend, 0.00482047),
(lami__micro_mg_tend, 0.00481632), (falouti__micro_mg_tend, 0.00419552),
(max__1179__micro_mg_tend, 0.00419419), (qc1__micro_mg_tend, 0.00419419),
(max__1110__micro_mg_tend, 0.00419419), (min__3195__micro_mg_tend, 0.00409082),
(ni__micro_mg_cam_tend, 0.00405991), (qinew__micro_mg_tend, 0.00394143),
(rainrt1__micro_mg_tend, 0.00382677), (qi1__micro_mg_tend, 0.00359375),
(dum2l__micro_mg_tend, 0.00344299), (max__2448__micro_mg_tend, 0.00341162),
(nc1__micro_mg_tend, 0.00335177), (nce__micro_mg_tend, 0.00335177),
(log10__3395__micro_mg_tend, 0.00332223), (nie__micro_mg_tend, 0.00293889),
(ninew__micro_mg_tend, 0.00293889), (min__1441__micro_mg_tend, 0.00283013),
(relhum__micro_mg_tend, 0.00273891), (q1__micro_mg_tend, 0.00273891),
(epsi__micro_mg_tend, 0.00268033), (dumnnuc__micro_mg_tend, 0.00263705),
(nnuccd__micro_mg_tend, 0.00263705), (ni1__micro_mg_tend, 0.00259121),
(max__2452__micro_mg_tend, 0.00259121), (relvar__micro_mg_cam_tend, 0.00259121),
(pgam__micro_mg_tend, 0.00222549), (t__wv_sat_svp_water, 0.0021072),
(t__wv_sat_svp_ice, 0.0021072), (sc__micro_mg_tend, 0.0020757),
(nsacwi__micro_mg_tend, 0.00205049), (dumni__micro_mg_tend, 0.00202004),
(tnd_qsnow__micro_mg_cam_tend, 0.00197555), (uns__micro_mg_tend, 0.00178918),
(ums__micro_mg_tend, 0.00178918), (unr__micro_mg_tend, 0.00173003),
(umr__micro_mg_tend, 0.00173003), (niic__micro_mg_tend, 0.00169304),
(qn__micro_mg_tend, 0.0014687), (tn__micro_mg_tend, 0.0014687),
(faloutnc__micro_mg_tend, 0.00142277), (dc0__micro_mg_tend, 0.00139588),
(mfp__micro_mg_tend, 0.0013838), (max__1103__micro_mg_tend, 0.00133112),
(min__1210__micro_mg_tend, 0.00133112), (min__2887__micro_mg_tend, 0.00131022),
(max__3246__micro_mg_tend, 0.00131022), (max__2889__micro_mg_tend, 0.00131022),
(min__3237__micro_mg_tend, 0.00131022), (max__2899__micro_mg_tend, 0.00114115),
(min__2900__micro_mg_tend, 0.00114115), (umc__micro_mg_tend, 0.00111449),
(unc__micro_mg_tend, 0.00111449), (min__1543__micro_mg_tend, 0.000930695),
(cdist1__micro_mg_tend, 0.000930695), (fc__micro_mg_tend, 0.000853678),
(fnc__micro_mg_tend, 0.000853678), (fni__micro_mg_tend, 0.000794134),
(fi__micro_mg_tend, 0.000794134), (faltndqce__micro_mg_tend, 0.000771977),
(abi__micro_mg_tend, 0.000691901), (ab__micro_mg_tend, 0.000691901),
(max__1545__micro_mg_tend, 0.000691901), (rhof__micro_mg_tend, 0.000691901),
(max__1497__micro_mg_tend, 0.000691901), (dz__micro_mg_tend, 0.000691901),
(uni__micro_mg_tend, 0.000670713), (umi__micro_mg_tend, 0.000670713),
(exp__1665__micro_mg_tend, 0.000635), (exp__1667__micro_mg_tend, 0.000635),
(min__2880__micro_mg_tend, 0.000635), (max__2879__micro_mg_tend, 0.000635),
(faltndqie__micro_mg_tend, 0.000553152), (qiic__micro_mg_tend, 0.000519651),
(ncmax__micro_mg_tend, 0.000453936), (faloutni__micro_mg_tend, 0.00037103),
(rhin__micro_mg_tend, 0.000361107), (mnuccd__micro_mg_tend, 0.000347677),
(max__1012__micro_mg_tend, 0.000347677), (max__1425__micro_mg_tend, 0.000347677),
(relvar__micro_mg_tend, 0.000341634), (lammax__micro_mg_tend, 0.000293416),
(min__3306__micro_mg_tend, 0.000293416), (max__1551__micro_mg_tend, 0.000293416),
(min__2893__micro_mg_tend, 0.000293416), (max__2892__micro_mg_tend, 0.000293416),
(lammin__micro_mg_tend, 0.000293416), (min__1552__micro_mg_tend, 0.000293416),
(max__3250__micro_mg_tend, 0.000293416), (min__3251__micro_mg_tend, 0.000293416),
(max__3305__micro_mg_tend, 0.000293416), (t__goffgratch_svp_water, 0.000277821),
(t__goffgratch_svp_ice, 0.000277821), (min__3202__micro_mg_tend, 0.000266329),
(min__2875__micro_mg_tend, 0.000266329), (tnd_qsnow__micro_mg_tend, 0.000260464),
(nslip3__micro_mg_tend, 0.000206499), (nslip4__micro_mg_tend, 0.000206499),
(nslip2__micro_mg_tend, 0.000206499), (nslip1__micro_mg_tend, 0.000206499),
(exp__1843__micro_mg_tend, 0.000182445), (exp__1844__micro_mg_tend, 0.000182445),
(exp__1841__micro_mg_tend, 0.000182445), (exp__1842__micro_mg_tend, 0.000182445),
(min__2924__micro_mg_tend, 0.000179652), (min__2925__micro_mg_tend, 0.000179652),
(max__3046__micro_mg_tend, 0.000112552), (max__3045__micro_mg_tend, 0.000112552),
(max__3044__micro_mg_tend, 0.000104701), (max__3043__micro_mg_tend, 0.000104701),
(acn__micro_mg_tend, 9.12225e-05), (ain__micro_mg_tend, 9.12225e-05),
(asn__micro_mg_tend, 9.12225e-05), (arn__micro_mg_tend, 9.12225e-05),
(max__2507__micro_mg_tend, 9.12225e-05), (max__2510__micro_mg_tend, 9.12225e-05),
(max__2508__micro_mg_tend, 9.12225e-05), (max__2509__micro_mg_tend, 9.12225e-05),
(min__1516__micro_mg_tend, 6.85125e-05), (qcvar__micro_mg_tend, 4.50421e-05),
(es__goffgratch_svp_water, 4.14581e-05), (es__goffgratch_svp_ice, 4.14581e-05),
(log10__374__goffgratch_svp_water, 3.66288e-05),
(log10__387__goffgratch_svp_ice, 3.66288e-05),
(cons18__micro_mg_tend, 5.93851e-06), (cons20__micro_mg_tend, 5.93851e-06),
(cons19__micro_mg_tend, 5.93851e-06), (es__wv_sat_svp_water, 5.46598e-06),
(es__wv_sat_svp_ice, 5.46598e-06), (es__wv_sat_qsat_water, 7.33401e-07),
(es__svp_water, 7.20653e-07), (es__svp_ice, 7.20653e-07),
(esn__micro_mg_tend, 1.90027e-07), (esl__micro_mg_tend, 1.20067e-07),
(esi__micro_mg_tend, 1.10843e-07), (es__qsat_water, 9.66941e-08),
(min__216__wv_sat_qsat_water, 9.66941e-08), (es__wv_sat_svp_to_qsat, 9.66941e-08),
(esl__aist_vector, 9.50133e-08), (esl__aist_single, 9.50133e-08),
(esi__aist_single, 9.50133e-08), (esi__aist_vector, 9.50133e-08),
(min__1167__micro_mg_tend, 3.06656e-08), (exp__2609__aist_single, 2.50578e-08),
(exp__2799__aist_vector, 2.50542e-08), (es__deriv_outputs, 1.27485e-08),
(qs__wv_sat_svp_to_qsat, 1.27485e-08), (aist__aist_single, 3.36269e-09),
(aist__aist_vector, 3.36174e-09), (dqsdt_loc__deriv_outputs, 1.93549e-09),
(desdt__deriv_outputs, 1.68091e-09), (qs__qsat, 1.6808e-09),
(qvl__micro_mg_tend, 1.6808e-09), (qvs__micro_mg_tend, 1.6808e-09),
(qsn__micro_mg_tend, 1.6808e-09), (qs__wv_sat_qsat_water, 1.6808e-09),
(qs__micro_mg_tend, 1.6808e-09), (qvi__micro_mg_tend, 1.6808e-09),
(max__2652__aist_single, 4.43348e-10),
(ai_st_nc__instratus_condensate, 4.43348e-10),
(max__2842__aist_vector, 4.43223e-10), (aist_out__aist_vector, 4.43223e-10),
(q__tq_enthalpy, 2.88425e-10), (qsp__findsp, 2.85227e-10),
(gam__deriv_outputs, 2.55182e-10), (dqsdt__deriv_outputs, 2.55182e-10),
(q1__findsp, 2.50819e-10), (qs__findsp, 2.50819e-10),
(qs__deriv_outputs, 2.50819e-10), (qs__compute_eddy_diff, 2.21603e-10),
(qs__sfdiag, 2.21603e-10), (qs__compute_uwshcu, 2.21603e-10),
(satq__gas_phase_chemdr, 2.21603e-10), (qs__conden, 2.21603e-10),
(qs__qsinvert, 2.21603e-10), (qs__qsat_water, 2.21603e-10),
(ai_st__instratus_condensate, 6.61569e-11),
(ai0_st_nc_in__instratus_condensate, 5.8436e-11),
(enthalpy__tq_enthalpy, 3.82222e-11), (qsp__findsp_vc, 3.76053e-11),
(qvd__findsp, 3.74289e-11), (gam__qsat_water, 3.36441e-11),
(gam__qsat, 3.3644e-11), (dqsdt__qsat_water, 3.3644e-11),
(dqsdt__qsat, 3.3644e-11), (derrdps__qsinvert, 3.31927e-11),
(r1b__findsp, 3.30693e-11), (qs__instratus_condensate, 3.30689e-11),
(dlnqsdt__qsinvert, 2.98017e-11), (max__4692__conden, 2.92349e-11),
(excessu__compute_uwshcu, 2.92177e-11), (max__3815__compute_uwshcu, 2.92172e-11),
(relhum__gas_phase_chemdr, 2.9217e-11), (excess0__compute_uwshcu, 2.92168e-11),
(temps__compute_eddy_diff, 2.92168e-11), (qxbot__sfdiag, 2.92168e-11),
(qxtop__sfdiag, 2.92168e-11), (rhi__qsinvert, 2.92168e-11),
(rvls__conden, 2.92168e-11), (err__qsinvert, 2.92168e-11),
(qsat_in__instratus_condensate, 2.92168e-11), (qsat_b__mmacro_pcond, 2.92168e-11),
(qsat0__instratus_condensate, 2.92168e-11), (qs__gridmean_rh, 2.92168e-11),
(qm__qsat_hpa, 2.92168e-11), (qs__aist_single, 2.92168e-11),
(qs__funcd_instratus, 2.92168e-11), (qsat_in__aist_vector, 2.92168e-11),
(enout__findsp, 1.39108e-11), (qi__instratus_condensate, 1.0047e-11),
(qi_st__instratus_condensate, 1.0047e-11), (gam__findsp, 8.87148e-12),
(dps__qsinvert, 8.22828e-12), (xsat__compute_uwshcu, 7.7042e-12),
(qxm__sfdiag, 7.7041e-12), (t__qsat, 5.11467e-12),
(tsp__findsp, 5.09003e-12), (enin__findsp, 5.03934e-12),
(qw0_in__compute_uwshcu, 4.95801e-12), (max__1047__sfdiag, 4.86778e-12),
(max__1057__sfdiag, 4.86778e-12), (abs__1057__sfdiag, 4.86778e-12),
(abs__1047__sfdiag, 4.86778e-12), (u__instratus_condensate, 4.53598e-12),
(temps__conden, 4.46065e-12), (beta__mmacro_pcond, 4.43767e-12),
(beta__funcd_instratus, 4.43689e-12), (dqsdt_b__mmacro_pcond, 4.43574e-12),
(gam__trbintd, 4.43574e-12), (dqsdt__funcd_instratus, 4.43574e-12),
(dqsdt__gridmean_rh, 4.43574e-12), (gam__qsinvert, 4.43574e-12),
(ncf__aist_single, 4.02669e-12), (f__gridmean_rh, 3.87683e-12),
(min__1899__gridmean_rh, 3.86244e-12), (u0__instratus_condensate, 3.86106e-12),
(qc__conden, 3.85444e-12), (tv__drydep_fromlnd, 3.85329e-12),
(u__mmacro_pcond, 3.85285e-12), (subsat__compute_uwshcu, 3.8521e-12),
(relhum__usrrxt, 3.85208e-12), (u__funcd_instratus, 3.85207e-12),
(u0_in__instratus_condensate, 3.85207e-12), (log__4816__qsinvert, 3.85205e-12),
(alpha__mmacro_pcond, 3.85204e-12), (qst__entropy, 3.85204e-12),
(qst__ientropy, 3.85204e-12), (qst__cldprp, 3.85204e-12),
(qs__aist_vector, 3.85204e-12), (alpha__funcd_instratus, 3.85204e-12),
(g__findsp, 2.49845e-12), (t__tq_enthalpy, 1.47498e-12),
(bb__mmacro_pcond, 1.38113e-12), (t__calc_hltalt, 1.34544e-12),
(t__instratus_condensate, 1.34066e-12), (qv__instratus_condensate, 1.33543e-12),
(qi__aist_single, 1.32464e-12), (qi_out__instratus_condensate, 1.32463e-12),
(sflh__sfdiag, 1.28357e-12), (sfuh__sfdiag, 1.28357e-12),
(ps__qsinvert, 1.25976e-12), (dgdt__findsp, 1.16965e-12),
(t1__findsp, 1.1547e-12), (x_cu__compute_uwshcu, 1.1497e-12),
(gammai__mmacro_pcond, 1.09295e-12), (dudt__funcd_instratus, 1.09284e-12),
(betast__mmacro_pcond, 1.09269e-12), (thv_x0__compute_uwshcu, 1.03041e-12),
(thv_x1__compute_uwshcu, 1.03041e-12), (qtxsat__compute_uwshcu, 1.01659e-12),
(thlxsat__compute_uwshcu, 1.01659e-12), (min__2502__compute_uwshcu, 1.01597e-12),
(t__qsat_water, 9.825e-13), (t__no_ip_hltalt, 9.78024e-13),
(t__deriv_outputs, 8.03883e-13), (t__gridmean_rh, 7.21826e-13),
(qu__cldprp, 7.172e-13), (chs__trbintd, 6.73635e-13),
(max__3823__compute_uwshcu, 6.54091e-13),
(u_nc__instratus_condensate, 5.98038e-13),
(fg__gridmean_rh, 5.95169e-13), (th__conden, 5.90522e-13),
(u_in__astg_pdf, 5.74841e-13), (qsthat__cldprp, 5.74824e-13),
(acos__2634__aist_single, 5.30894e-13), (evprain__compute_uwshcu, 5.27919e-13),
(evpsnow__compute_uwshcu, 5.2772e-13), (qv__conden, 5.26283e-13),
(qv__gridmean_rh, 5.18249e-13), (log__3529__entropy, 5.17889e-13),
(tv__drydep_xactive, 5.10641e-13), (u0_nc__instratus_condensate, 5.09056e-13),
(qi__conden, 5.08183e-13), (ql__conden, 5.08181e-13),
(term__drydep_fromlnd, 5.08033e-13), (u_nc__mmacro_pcond, 5.07972e-13),
(ncf__aist_vector, 5.0797e-13), (xr__usrrxt, 5.07876e-13),
(min__3526__entropy, 5.07871e-13), (u_nc__funcd_instratus, 5.07871e-13),
(tlcl__qsinvert, 5.0787e-13), (hsat__cldprp, 5.07868e-13),
(qsmix__parcel_dilute, 5.07866e-13), (log__2185__cldprp, 5.07866e-13),
(gamma__cldprp, 5.07866e-13), (min__991__sfdiag, 3.38458e-13),
(min__960__sfdiag, 3.38457e-13), (min__1050__sfdiag, 3.38457e-13),
(sfi__sfdiag, 3.03096e-13), (bquad__compute_uwshcu, 2.77358e-13),
(u__astg_pdf_single, 2.12925e-13), (t__aist_single, 1.89303e-13),
(dpisdps__qsinvert, 1.87993e-13), (dqlstdt__mmacro_pcond, 1.82095e-13),
(dalstdt__mmacro_pcond, 1.82094e-13), (sten__compute_uwshcu, 1.77447e-13),
(tc__calc_hltalt, 1.7739e-13), (t_out__instratus_condensate, 1.76751e-13),
(qv_out__instratus_condensate, 1.76066e-13),
(qi_star__mmacro_pcond, 1.74643e-13),
(qi_0__mmacro_pcond, 1.74642e-13), (qvten__compute_uwshcu, 1.74133e-13),
(xc__compute_uwshcu, 1.71568e-13), (sflh__trbintd, 1.6923e-13),
(sfuh__trbintd, 1.69229e-13), (p__qsat, 1.66583e-13),
(dlnqsdps__qsinvert, 1.66149e-13), (qsinvert__qsinvert, 1.66092e-13),
(pis__qsinvert, 1.66091e-13), (su__cldprp, 1.53827e-13),
(max__2516__compute_uwshcu, 1.51584e-13), (tmp1__compute_uwshcu, 1.50551e-13),
(tmp2__compute_uwshcu, 1.5055e-13), (hltalt__no_ip_hltalt, 1.48528e-13),
(dalstdt__funcd_instratus, 1.44154e-13), (hsthat__cldprp, 1.42745e-13),
(cquad__compute_uwshcu, 1.41406e-13), (thl__conden, 1.37391e-13),
(qt__conden, 1.37388e-13), (t__wv_sat_qsat_water, 1.29542e-13),
(qc__gridmean_rh, 1.09113e-13), (tc__gridmean_rh, 1.0573e-13),
(qds__cldprp, 1.04607e-13), (cms__trbintd, 1.02303e-13),
(t0__instratus_condensate, 9.51897e-14), (qu__zm_convr, 9.45563e-14),
```

## B.2 AVX2 Subgraph Centrality: First Community

```
>>> avx2_community1_incentrality[:500]
(dum__micro_mg_tend, 0.455153), (ratio__micro_mg_tend, 0.325264),
(tlat__micro_mg_tend, 0.255383), (qniic__micro_mg_tend, 0.198578),
(nric__micro_mg_tend, 0.196431), (nsic__micro_mg_tend, 0.191075),
(qctend__micro_mg_tend, 0.188477), (qric__micro_mg_tend, 0.180318),
(qitend__micro_mg_tend, 0.15969), (prds__micro_mg_tend, 0.157626),
(pre__micro_mg_tend, 0.157551), (nctend__micro_mg_tend, 0.148088),
(qvlat__micro_mg_tend, 0.132584), (mnuccc__micro_mg_tend, 0.121525),
(nitend__micro_mg_tend, 0.120172), (nsagg__micro_mg_tend, 0.109382),
(nragg__micro_mg_tend, 0.107409), (dum1__micro_mg_tend, 0.0964774),
(qnitend__micro_mg_tend, 0.0875948), (nnuccr__micro_mg_tend, 0.0875681),
(mnuccr__micro_mg_tend, 0.0875681), (pracs__micro_mg_tend, 0.0834417),
(npracs__micro_mg_tend, 0.0834417), (tmp__micro_mg_tend, 0.0822866),
(qrtot__micro_mg_tend, 0.0815155), (nrtot__micro_mg_tend, 0.0815155),
(qstot__micro_mg_tend, 0.0815155), (nstot__micro_mg_tend, 0.0815155),
(ttmp__micro_mg_tend, 0.0814203), (min__2140__micro_mg_tend, 0.0807908),
(min__2122__micro_mg_tend, 0.080781), (psacws__micro_mg_tend, 0.0781575),
(npsacws__micro_mg_tend, 0.0773532), (pra__micro_mg_tend, 0.0767817),
(qrtend__micro_mg_tend, 0.0693434), (nprai__micro_mg_tend, 0.0670967),
(prai__micro_mg_tend, 0.0669185), (qclr__micro_mg_tend, 0.0627478),
(qtmp__micro_mg_tend, 0.0627211), (eci__micro_mg_tend, 0.0621703),
(npra__micro_mg_tend, 0.0621291), (faltndc__micro_mg_tend, 0.0607809),
(faltndnc__micro_mg_tend, 0.0601965), (max__1158__micro_mg_tend, 0.0600089),
(min__2559__micro_mg_tend, 0.0600089), (max__3087__micro_mg_tend, 0.0600089),
(min__2168__micro_mg_tend, 0.0600089), (min__2177__micro_mg_tend, 0.0600089),
(min__3004__micro_mg_tend, 0.0600089), (min__2525__micro_mg_tend, 0.0600089),
(max__2524__micro_mg_tend, 0.0600089), (max__3120__micro_mg_tend, 0.0600089),
(min__3121__micro_mg_tend, 0.0600089), (min__3088__micro_mg_tend, 0.0600089),
(max__2558__micro_mg_tend, 0.0600089), (nprc1__micro_mg_tend, 0.0582542),
(n0s__micro_mg_tend, 0.0581463), (nsubi__micro_mg_tend, 0.0576126),
(n0r__micro_mg_tend, 0.0562208), (nnucct__micro_mg_tend, 0.0552363),
(mnucct__micro_mg_tend, 0.0552363), (msacwi__micro_mg_tend, 0.0531491),
(nnuccc__micro_mg_tend, 0.052809), (bergs__micro_mg_tend, 0.0520531),
(lams__micro_mg_tend, 0.0513731), (prc__micro_mg_tend, 0.0512664),
(prci__micro_mg_tend, 0.051026), (nprci__micro_mg_tend, 0.0509864),
(nrtend__micro_mg_tend, 0.0508104), (lamr__micro_mg_tend, 0.0496719),
(nsubr__micro_mg_tend, 0.0493965), (nsubs__micro_mg_tend, 0.0493965),
(nsubc__micro_mg_tend, 0.0493965), (nstend__micro_mg_tend, 0.0451548),
(dumc__micro_mg_tend, 0.0435571), (t__micro_mg_tend, 0.039804),
(tlat1__micro_mg_tend, 0.038784), (qc__micro_mg_tend, 0.0318119),
(dumi__micro_mg_tend, 0.0310279), (qsout__micro_mg_tend, 0.0301572),
(qctend1__micro_mg_tend, 0.0286232), (qrout__micro_mg_tend, 0.0273842),
(qi__micro_mg_tend, 0.0272577), (max__2596__micro_mg_tend, 0.0258981),
(max__1722__micro_mg_tend, 0.0258981), (nc__micro_mg_tend, 0.0254224),
(dumfice__micro_mg_tend, 0.0252079), (rainrt__micro_mg_tend, 0.0251983),
(max__1723__micro_mg_tend, 0.025192), (max__2597__micro_mg_tend, 0.025192),
(qitend1__micro_mg_tend, 0.0242514), (nctend1__micro_mg_tend, 0.0224895),
(q__micro_mg_tend, 0.020774), (qvlat1__micro_mg_tend, 0.020135),
(ni__micro_mg_tend, 0.0196537), (nitend1__micro_mg_tend, 0.0182501),
(epss__micro_mg_tend, 0.0168008), (epsr__micro_mg_tend, 0.0163226),
(t__svp_water, 0.0159826), (t__svp_ice, 0.0159826),
(ni_secp__micro_mg_tend, 0.0155524), (nfice__micro_mg_tend, 0.0149841),
(faltndi__micro_mg_tend, 0.0132731), (faltndni__micro_mg_tend, 0.0127688),
(min__3006__micro_mg_tend, 0.0127199), (max__1969__micro_mg_tend, 0.012355),
(min__1959__micro_mg_tend, 0.012355), (min__1963__micro_mg_tend, 0.012355),
(state_loc%q__micro_mg_cam_tend, 0.0111397), (prd__micro_mg_tend, 0.0100962),
(dumnc__micro_mg_tend, 0.00993771), (lamc__micro_mg_tend, 0.0083619),
(max__1931__micro_mg_tend, 0.00819674), (min__1932__micro_mg_tend, 0.00819674),
(cmei__micro_mg_tend, 0.00718911), (ncic__micro_mg_tend, 0.0070591),
(min__1785__micro_mg_tend, 0.00678523), (min__2680__micro_mg_tend, 0.00678523),
(min__2681__micro_mg_tend, 0.00678523), (min__1784__micro_mg_tend, 0.00678523),
(ds0__micro_mg_tend, 0.0067732), (nprc__micro_mg_tend, 0.00675914),
(min__2628__micro_mg_tend, 0.00656094), (min__2627__micro_mg_tend, 0.00656094),
(min__1751__micro_mg_tend, 0.00656094), (min__1752__micro_mg_tend, 0.00656094),
(ndfaer2__micro_mg_tend, 0.00596703), (ndfaer1__micro_mg_tend, 0.00596703),
(ndfaer4__micro_mg_tend, 0.00596703), (ndfaer3__micro_mg_tend, 0.00596703),
(faloutc__micro_mg_tend, 0.00585526), (min__3194__micro_mg_tend, 0.00574271),
(qc__micro_mg_cam_tend, 0.00566289), (berg__micro_mg_tend, 0.00547565),
(qie__micro_mg_tend, 0.00526351), (dqsidt__micro_mg_tend, 0.0052479),
(dqsdt__micro_mg_tend, 0.0052479), (exp__1820__micro_mg_tend, 0.0052479),
(t1__micro_mg_tend, 0.0052479), (tcnt__micro_mg_tend, 0.0052479),
(rho__micro_mg_tend, 0.0052479), (mu__micro_mg_tend, 0.0052479),
(viscosity__micro_mg_tend, 0.0052479), (exp__2006__micro_mg_tend, 0.0052479),
(exp__1826__micro_mg_tend, 0.0052479), (exp__1813__micro_mg_tend, 0.0052479),
(sqrt__1838__micro_mg_tend, 0.0052479), (dv__micro_mg_tend, 0.0052479),
(exp__2002__micro_mg_tend, 0.0052479), (exp__1807__micro_mg_tend, 0.0052479),
(qi__micro_mg_cam_tend, 0.00506245), (n0i__micro_mg_tend, 0.00501755),
(qce__micro_mg_tend, 0.00491612), (nc__micro_mg_cam_tend, 0.00482047),
(lami__micro_mg_tend, 0.00481632), (falouti__micro_mg_tend, 0.00419552),
(qc1__micro_mg_tend, 0.00419419), (max__1110__micro_mg_tend, 0.00419419),
(max__1179__micro_mg_tend, 0.00419419), (min__3195__micro_mg_tend, 0.00409082),
(ni__micro_mg_cam_tend, 0.00405991), (qinew__micro_mg_tend, 0.00394143),
(rainrt1__micro_mg_tend, 0.00382677), (qi1__micro_mg_tend, 0.00359375),
(dum2l__micro_mg_tend, 0.00344299), (max__2448__micro_mg_tend, 0.00341162),
(nce__micro_mg_tend, 0.00335177), (nc1__micro_mg_tend, 0.00335177),
(log10__3395__micro_mg_tend, 0.00332223), (nie__micro_mg_tend, 0.00293889),
(ninew__micro_mg_tend, 0.00293889), (min__1441__micro_mg_tend, 0.00283013),
(relhum__micro_mg_tend, 0.00273891), (q1__micro_mg_tend, 0.00273891),
(epsi__micro_mg_tend, 0.00268033), (nnuccd__micro_mg_tend, 0.00263705),
(dumnnuc__micro_mg_tend, 0.00263705), (ni1__micro_mg_tend, 0.00259121),
(relvar__micro_mg_cam_tend, 0.00259121), (max__2452__micro_mg_tend, 0.00259121),
(pgam__micro_mg_tend, 0.00222549), (t__wv_sat_svp_water, 0.0021072),
(t__wv_sat_svp_ice, 0.0021072), (sc__micro_mg_tend, 0.0020757),
(nsacwi__micro_mg_tend, 0.00205049), (dumni__micro_mg_tend, 0.00202004),
(tnd_qsnow__micro_mg_cam_tend, 0.00197555), (uns__micro_mg_tend, 0.00178918),
(ums__micro_mg_tend, 0.00178918), (umr__micro_mg_tend, 0.00173003),
(unr__micro_mg_tend, 0.00173003), (niic__micro_mg_tend, 0.00169304),
(tn__micro_mg_tend, 0.0014687), (qn__micro_mg_tend, 0.0014687),
(faloutnc__micro_mg_tend, 0.00142277), (dc0__micro_mg_tend, 0.00139588),
(mfp__micro_mg_tend, 0.0013838), (max__1103__micro_mg_tend, 0.00133112),
(min__1210__micro_mg_tend, 0.00133112), (min__2887__micro_mg_tend, 0.00131022),
(max__3246__micro_mg_tend, 0.00131022), (max__2889__micro_mg_tend, 0.00131022),
(min__3237__micro_mg_tend, 0.00131022), (max__2899__micro_mg_tend, 0.00114115),
(min__2900__micro_mg_tend, 0.00114115), (umc__micro_mg_tend, 0.00111449),
(unc__micro_mg_tend, 0.00111449), (cdist1__micro_mg_tend, 0.000930695),
(min__1543__micro_mg_tend, 0.000930695), (fnc__micro_mg_tend, 0.000853678),
(fc__micro_mg_tend, 0.000853678), (fni__micro_mg_tend, 0.000794134),
(fi__micro_mg_tend, 0.000794134), (faltndqce__micro_mg_tend, 0.000771977),
(abi__micro_mg_tend, 0.000691901), (ab__micro_mg_tend, 0.000691901),
(rhof__micro_mg_tend, 0.000691901), (dz__micro_mg_tend, 0.000691901),
(max__1497__micro_mg_tend, 0.000691901), (max__1545__micro_mg_tend, 0.000691901),
(uni__micro_mg_tend, 0.000670713), (umi__micro_mg_tend, 0.000670713),
(exp__1667__micro_mg_tend, 0.000635), (max__2879__micro_mg_tend, 0.000635),
(min__2880__micro_mg_tend, 0.000635), (exp__1665__micro_mg_tend, 0.000635),
(faltndqie__micro_mg_tend, 0.000553152), (qiic__micro_mg_tend, 0.000519651),
(ncmax__micro_mg_tend, 0.000453936), (faloutni__micro_mg_tend, 0.00037103),
(rhin__micro_mg_tend, 0.000361107), (mnuccd__micro_mg_tend, 0.000347677),
(max__1012__micro_mg_tend, 0.000347677), (max__1425__micro_mg_tend, 0.000347677),
(relvar__micro_mg_tend, 0.000341634), (lammin__micro_mg_tend, 0.000293416),
(max__3250__micro_mg_tend, 0.000293416), (max__2892__micro_mg_tend, 0.000293416),
(min__2893__micro_mg_tend, 0.000293416), (lammax__micro_mg_tend, 0.000293416),
(min__3306__micro_mg_tend, 0.000293416), (max__1551__micro_mg_tend, 0.000293416),
(max__3305__micro_mg_tend, 0.000293416), (min__1552__micro_mg_tend, 0.000293416),
(min__3251__micro_mg_tend, 0.000293416), (t__goffgratch_svp_water, 0.000277821),
(t__goffgratch_svp_ice, 0.000277821), (min__3202__micro_mg_tend, 0.000266329),
(min__2875__micro_mg_tend, 0.000266329), (tnd_qsnow__micro_mg_tend, 0.000260464),
(nslip1__micro_mg_tend, 0.000206499), (nslip3__micro_mg_tend, 0.000206499),
(nslip2__micro_mg_tend, 0.000206499), (nslip4__micro_mg_tend, 0.000206499),
(exp__1841__micro_mg_tend, 0.000182445), (exp__1843__micro_mg_tend, 0.000182445),
(exp__1842__micro_mg_tend, 0.000182445), (exp__1844__micro_mg_tend, 0.000182445),
(min__2925__micro_mg_tend, 0.000179652), (min__2924__micro_mg_tend, 0.000179652),
(max__3046__micro_mg_tend, 0.000112552), (max__3045__micro_mg_tend, 0.000112552),
(max__3043__micro_mg_tend, 0.000104701), (max__3044__micro_mg_tend, 0.000104701),
(acn__micro_mg_tend, 9.12225e-05), (arn__micro_mg_tend, 9.12225e-05),
(ain__micro_mg_tend, 9.12225e-05), (asn__micro_mg_tend, 9.12225e-05),
(max__2507__micro_mg_tend, 9.12225e-05), (max__2510__micro_mg_tend, 9.12225e-05),
(max__2508__micro_mg_tend, 9.12225e-05), (max__2509__micro_mg_tend, 9.12225e-05),
(min__1516__micro_mg_tend, 6.85125e-05), (qcvar__micro_mg_tend, 4.50421e-05),
(es__goffgratch_svp_water, 4.14581e-05), (es__goffgratch_svp_ice, 4.14581e-05),
(log10__374__goffgratch_svp_water, 3.66288e-05),
(log10__387__goffgratch_svp_ice, 3.66288e-05),
(cons19__micro_mg_tend, 5.93851e-06), (cons20__micro_mg_tend, 5.93851e-06),
(cons18__micro_mg_tend, 5.93851e-06), (es__wv_sat_svp_water, 5.46598e-06),
(es__wv_sat_svp_ice, 5.46598e-06), (es__wv_sat_qsat_water, 7.33401e-07),
(es__svp_water, 7.20653e-07), (es__svp_ice, 7.20653e-07),
(esn__micro_mg_tend, 1.90027e-07), (esl__micro_mg_tend, 1.20067e-07),
(esi__micro_mg_tend, 1.10843e-07), (es__wv_sat_svp_to_qsat, 9.66941e-08),
(min__216__wv_sat_qsat_water, 9.66941e-08), (es__qsat_water, 9.66941e-08),
(esi__aist_vector, 9.50133e-08), (esi__aist_single, 9.50133e-08),
(esl__aist_single, 9.50133e-08), (esl__aist_vector, 9.50133e-08),
(min__1167__micro_mg_tend, 3.06656e-08), (exp__2609__aist_single, 2.50578e-08),
(exp__2799__aist_vector, 2.50542e-08), (es__deriv_outputs, 1.27485e-08),
(qs__wv_sat_svp_to_qsat, 1.27485e-08), (aist__aist_single, 3.36269e-09),
(aist__aist_vector, 3.36174e-09), (dqsdt_loc__deriv_outputs, 1.93549e-09),
(desdt__deriv_outputs, 1.68091e-09), (qs__qsat, 1.6808e-09),
(qvi__micro_mg_tend, 1.6808e-09), (qvs__micro_mg_tend, 1.6808e-09),
(qs__micro_mg_tend, 1.6808e-09), (qsn__micro_mg_tend, 1.6808e-09),
(qvl__micro_mg_tend, 1.6808e-09), (qs__wv_sat_qsat_water, 1.6808e-09),
(max__2652__aist_single, 4.43348e-10),
(ai_st_nc__instratus_condensate, 4.43348e-10),
(max__2842__aist_vector, 4.43223e-10), (aist_out__aist_vector, 4.43223e-10),
(q__tq_enthalpy, 2.88425e-10), (qsp__findsp, 2.85227e-10),
(gam__deriv_outputs, 2.55182e-10), (dqsdt__deriv_outputs, 2.55182e-10),
(q1__findsp, 2.50819e-10), (qs__deriv_outputs, 2.50819e-10),
(qs__findsp, 2.50819e-10), (qs__sfdiag, 2.21603e-10),
(qs__qsinvert, 2.21603e-10), (qs__compute_uwshcu, 2.21603e-10),
(qs__conden, 2.21603e-10), (qs__compute_eddy_diff, 2.21603e-10),
(qs__qsat_water, 2.21603e-10), (ai_st__instratus_condensate, 6.61569e-11),
(ai0_st_nc_in__instratus_condensate, 5.8436e-11),
(enthalpy__tq_enthalpy, 3.82222e-11),
(qsp__findsp_vc, 3.76053e-11), (qvd__findsp, 3.74288e-11),
(gam__qsat, 3.3644e-11), (gam__qsat_water, 3.3644e-11),
(dqsdt__qsat_water, 3.3644e-11), (dqsdt__qsat, 3.3644e-11),
(derrdps__qsinvert, 3.31927e-11), (r1b__findsp, 3.30693e-11),
(qs__instratus_condensate, 3.30689e-11), (dlnqsdt__qsinvert, 2.98017e-11),
(max__4692__conden, 2.92349e-11), (excessu__compute_uwshcu, 2.92177e-11),
(max__3815__compute_uwshcu, 2.92172e-11), (excess0__compute_uwshcu, 2.92168e-11),
(qxtop__sfdiag, 2.92168e-11), (err__qsinvert, 2.92168e-11),
(rhi__qsinvert, 2.92168e-11), (qxbot__sfdiag, 2.92168e-11),
(rvls__conden, 2.92168e-11), (qsat0__instratus_condensate, 2.92168e-11),
(qs__aist_single, 2.92168e-11), (qsat_in__instratus_condensate, 2.92168e-11),
(qsat_b__mmacro_pcond, 2.92168e-11), (qs__funcd_instratus, 2.92168e-11),
(qsat_in__aist_vector, 2.92168e-11), (qm__qsat_hpa, 2.92168e-11),
(qs__gridmean_rh, 2.92168e-11), (temps__compute_eddy_diff, 2.92168e-11),
(enout__findsp, 1.39108e-11), (qi__instratus_condensate, 1.0047e-11),
(qi_st__instratus_condensate, 1.0047e-11), (gam__findsp, 8.87149e-12),
(dps__qsinvert, 8.22828e-12), (xsat__compute_uwshcu, 7.7042e-12),
(qxm__sfdiag, 7.70408e-12), (t__qsat, 5.11466e-12),
(tsp__findsp, 5.09004e-12), (enin__findsp, 5.03934e-12),
(qw0_in__compute_uwshcu, 4.95801e-12), (abs__1057__sfdiag, 4.86778e-12),
(max__1047__sfdiag, 4.86778e-12), (max__1057__sfdiag, 4.86778e-12),
(abs__1047__sfdiag, 4.86778e-12), (u__instratus_condensate, 4.53598e-12),
(temps__conden, 4.46067e-12), (beta__mmacro_pcond, 4.43767e-12),
(beta__funcd_instratus, 4.4369e-12), (gam__qsinvert, 4.43575e-12),
(gam__trbintd, 4.43575e-12), (dqsdt_b__mmacro_pcond, 4.43574e-12),
(dqsdt__funcd_instratus, 4.43574e-12), (dqsdt__gridmean_rh, 4.43574e-12),
(ncf__aist_single, 4.02669e-12), (f__gridmean_rh, 3.87683e-12),
(min__1899__gridmean_rh, 3.86244e-12), (u0__instratus_condensate, 3.86106e-12),
(qc__conden, 3.85444e-12), (u__mmacro_pcond, 3.85285e-12),
(subsat__compute_uwshcu, 3.8521e-12), (u__funcd_instratus, 3.85208e-12),
(u0_in__instratus_condensate, 3.85207e-12), (log__4816__qsinvert, 3.85205e-12),
(qs__aist_vector, 3.85205e-12), (alpha__funcd_instratus, 3.85205e-12),
(alpha__mmacro_pcond, 3.85204e-12), (qst__entropy, 3.85204e-12),
(qst__cldprp, 3.85204e-12), (qst__ientropy, 3.85204e-12),
(g__findsp, 2.49845e-12), (t__tq_enthalpy, 1.47496e-12),
(bb__mmacro_pcond, 1.38112e-12), (t__calc_hltalt, 1.34543e-12),
(t__instratus_condensate, 1.34067e-12), (qv__instratus_condensate, 1.33544e-12),
(qi__aist_single, 1.32465e-12), (qi_out__instratus_condensate, 1.32463e-12),
(sflh__sfdiag, 1.28357e-12), (sfuh__sfdiag, 1.28356e-12),
(ps__qsinvert, 1.25976e-12), (dgdt__findsp, 1.16965e-12),
(t1__findsp, 1.1547e-12), (x_cu__compute_uwshcu, 1.1497e-12),
(gammai__mmacro_pcond, 1.09294e-12), (dudt__funcd_instratus, 1.09285e-12),
(betast__mmacro_pcond, 1.09269e-12), (thv_x0__compute_uwshcu, 1.03041e-12),
(thv_x1__compute_uwshcu, 1.03041e-12), (qtxsat__compute_uwshcu, 1.01659e-12),
(thlxsat__compute_uwshcu, 1.01659e-12), (min__2502__compute_uwshcu, 1.01597e-12),
(t__qsat_water, 9.82502e-13), (t__no_ip_hltalt, 9.78012e-13),
(t__deriv_outputs, 8.03864e-13), (t__gridmean_rh, 7.21828e-13),
(qu__cldprp, 7.17193e-13), (chs__trbintd, 6.73632e-13),
(max__3823__compute_uwshcu, 6.54088e-13),
(u_nc__instratus_condensate, 5.98039e-13),
(fg__gridmean_rh, 5.95163e-13), (th__conden, 5.90526e-13),
(u_in__astg_pdf, 5.74843e-13), (qsthat__cldprp, 5.74829e-13),
(acos__2634__aist_single, 5.30894e-13), (evprain__compute_uwshcu, 5.27926e-13),
(evpsnow__compute_uwshcu, 5.27728e-13), (qv__conden, 5.26301e-13),
(qv__gridmean_rh, 5.18254e-13), (log__3529__entropy, 5.17896e-13),
(u0_nc__instratus_condensate, 5.09051e-13), (qi__conden, 5.08181e-13),
(ql__conden, 5.08181e-13), (ncf__aist_vector, 5.0798e-13),
(u_nc__mmacro_pcond, 5.07972e-13), (log__2185__cldprp, 5.07869e-13),
(min__3526__entropy, 5.07868e-13), (u_nc__funcd_instratus, 5.07867e-13),
(hsat__cldprp, 5.07867e-13), (tlcl__qsinvert, 5.07867e-13),
(gamma__cldprp, 5.07865e-13), (qsmix__parcel_dilute, 5.07864e-13),
(min__991__sfdiag, 3.38462e-13), (min__960__sfdiag, 3.38462e-13),
(min__1050__sfdiag, 3.3846e-13), (sfi__sfdiag, 3.03103e-13),
(bquad__compute_uwshcu, 2.77361e-13), (u__astg_pdf_single, 2.12923e-13),
(t__aist_single, 1.89305e-13), (dpisdps__qsinvert, 1.87991e-13),
(dalstdt__mmacro_pcond, 1.821e-13), (dqlstdt__mmacro_pcond, 1.82099e-13),
(sten__compute_uwshcu, 1.77428e-13), (tc__calc_hltalt, 1.77388e-13),
(t_out__instratus_condensate, 1.76755e-13),
(qv_out__instratus_condensate, 1.76067e-13),
(qi_star__mmacro_pcond, 1.74644e-13), (qi_0__mmacro_pcond, 1.74641e-13),
(qvten__compute_uwshcu, 1.74137e-13), (xc__compute_uwshcu, 1.71561e-13),
(sflh__trbintd, 1.69233e-13), (sfuh__trbintd, 1.69228e-13),
(p__qsat, 1.66594e-13), (dlnqsdps__qsinvert, 1.66156e-13),
(qsinvert__qsinvert, 1.66093e-13), (pis__qsinvert, 1.66091e-13),
(su__cldprp, 1.53819e-13), (max__2516__compute_uwshcu, 1.51589e-13),
(tmp1__compute_uwshcu, 1.50547e-13), (tmp2__compute_uwshcu, 1.50543e-13),
(hltalt__no_ip_hltalt, 1.48521e-13), (dalstdt__funcd_instratus, 1.44153e-13),
(hsthat__cldprp, 1.42747e-13), (cquad__compute_uwshcu, 1.41414e-13),
(qt__conden, 1.37389e-13), (thl__conden, 1.37387e-13),
(t__wv_sat_qsat_water, 1.29535e-13), (qc__gridmean_rh, 1.09111e-13),
(tc__gridmean_rh, 1.05726e-13), (qds__cldprp, 1.04608e-13),
(cms__trbintd, 1.02301e-13), (t0__instratus_condensate, 9.51859e-14),
(qu__zm_convr, 9.45578e-14), (ch__trbintd, 9.40818e-14),
(chs__compute_eddy_diff, 8.88135e-14), (evplimit__compute_uwshcu, 8.62434e-14),
(ql__gridmean_rh, 8.01248e-14), (qt__gridmean_rh, 7.8889e-14),
(sign__1887__gridmean_rh, 7.84719e-14), (thj__compute_uwshcu, 7.78603e-14),
```

## B.3 AVX2 Subgraph Centrality: Second Community

```
>>> avx2_community2_incentrality[:500]
(vmr__gas_phase_chemdr, 0.621436),
(vmr__aero_model_gasaerexch, 0.242134),
(base_sol__exp_sol, 0.209415),
(del_h2so4_gasprod__gas_phase_chemdr, 0.209381),
(hcl_gas__gas_phase_chemdr, 0.209381),
(base_sol__imp_sol, 0.173494),
(conc__charge_balance, 0.171145),
(vmr__noy_ubc_set, 0.168327),
(h2ovmr__gas_phase_chemdr, 0.167234),
(vmr__mmr2vmr, 0.156613),
(hno3_gas__gas_phase_chemdr, 0.156613),
(fld__negtrc, 0.156613),
(vmr__rate_diags_calc, 0.156613),
(vmr__flbc_set, 0.156613),
(sulfate__gas_phase_chemdr, 0.156613),
(vmr0__gas_phase_chemdr, 0.156613),
(vmr__setinv, 0.156613),
(q__modal_aero_gasaerexch_sub, 0.09933),
(dvmrdt__aero_model_gasaerexch, 0.0948804),
(q__modal_aero_coag_sub, 0.0899662),
(loss_out__imp_sol, 0.0895197),
(q__modal_aero_newnuc_sub, 0.0815994),
(del_h2so4_aeruptk__aero_model_gasaerexch, 0.0815821),
(qin__setsox, 0.0684489),
(wrk__charge_balance, 0.057664),
(del_h2so4_gasprod__aero_model_gasaerexch, 0.0527676),
(dqdt__modal_aero_gasaerexch_sub, 0.0526754),
(lsol__imp_sol, 0.0484051),
(xso4__setsox, 0.0483835),
(max__793__imp_sol, 0.0437234),
(max__781__imp_sol, 0.0437234),

(xno2__noy_ubc_set, 0.0424214),

(xno__noy_ubc_set, 0.0424214),

(h2o_gas__gas_phase_chemdr, 0.042146),

(h2ovmr__usrrxt, 0.042146),

(h2ovmr__setinv, 0.042146),

(relhum__gas_phase_chemdr, 0.042146),

(vmr0__aero_model_gasaerexch, 0.0394692),

(sol__set_rates, 0.0394692),

(sum1__setinv, 0.0394692),

(sulfate__usrrxt, 0.0394692),

(xh2o2__setsox, 0.0364824),

(xso2__setsox, 0.0364119),

(vol_core__modal_aero_gasaerexch_sub, 0.0334672),

(vol_core__modal_aero_coag_sub, 0.0303123),

(qin__sox_cldaero_update, 0.0294701),

(xnox__noy_ubc_set, 0.0285861),

(invariants__setinv, 0.0274986),

(sum_dqdt_soa__modal_aero_gasaerexch_sub, 0.0260087),

(xferrate__modal_aero_gasaerexch_sub, 0.0251289),

(sum_dqdt_nh4__modal_aero_gasaerexch_sub, 0.025033),

(sum_dqdt_msa__modal_aero_gasaerexch_sub, 0.025033),

(sum_dqdt_so4__modal_aero_gasaerexch_sub, 0.025033),

(qold_so4__modal_aero_gasaerexch_sub, 0.0250329),

(g_soa_in__modal_aero_soaexch, 0.0250329),

(qold_soa__modal_aero_gasaerexch_sub, 0.0250329),

(qold_poa__modal_aero_gasaerexch_sub, 0.0250329),

(qold_nh4__modal_aero_gasaerexch_sub, 0.0250329),

(q__gas_aer_uptkrates, 0.0250329),

(xnumbconc__modal_aero_coag_sub, 0.0242108),

(dqdt_other__modal_aero_gasaerexch_sub, 0.0239115),

(xferamt__modal_aero_coag_sub, 0.0229231),

(o3s_loss__imp_sol, 0.0225605),
(max__300__modal_aero_newnuc_sub, 0.0205645),
(qh2so4_cur__modal_aero_newnuc_sub, 0.0205645),
(del_h2so4_aeruptk__modal_aero_newnuc_sub, 0.0205601),
(solution__imp_sol, 0.0185767),
(rno__noy_ubc_set, 0.0178951),
(dqdt__modal_aero_rename_sub, 0.0177518),
(xho2__setsox, 0.0174722),
(xo3__setsox, 0.0174722),
(xnh3__setsox, 0.0172503),
(xh2so4__setsox, 0.0172503),
(xmsa__setsox, 0.0172503),
(rxt_rates__set_rates, 0.0161302),
(xdelso4hp__setsox, 0.0152665),
(rxt__usrrxt, 0.0151567),
(del_h2so4_gasprod__modal_aero_newnuc_sub, 0.0132984),
(dqdt_nh4__modal_aero_gasaerexch_sub, 0.0132226),
(dqdt_so4__modal_aero_gasaerexch_sub, 0.0126199),
(y__imp_prod_loss, 0.0121989),
(xso4_init__setsox, 0.0121935),
(xso4__sox_cldaero_update, 0.0121935),
(fc__usrrxt, 0.0110617),
(tv__drydep_fromlnd, 0.0106723),
(relhum__usrrxt, 0.0106215),
(sur__usrrxt, 0.0106215),
(qnew_so4__modal_aero_gasaerexch_sub, 0.00948918),
(h2o2g__setsox, 0.0091942),
(xh2o2__sox_cldaero_update, 0.0091942),
(so2g__setsox, 0.00917643),
(xso2__sox_cldaero_update, 0.00917643),
(reaction_rates__gas_phase_chemdr, 0.00881511),
(max__580__modal_aero_gasaerexch_sub, 0.00843434),
(rxt_rates__rate_diags_calc, 0.00840482),
(vol_shell__modal_aero_gasaerexch_sub, 0.00829866),
(tmp2__modal_aero_coag_sub, 0.00815733),
(deldryvol_a__modal_aero_rename_sub, 0.00801148),
(vol_shell__modal_aero_coag_sub, 0.00790697),
(pso4__setsox, 0.00782152),
(o3s_loss__gas_phase_chemdr, 0.00760132),
(max__448__sox_cldaero_update, 0.00742698),
(max__445__sox_cldaero_update, 0.00742698),
(dqdt_soa__modal_aero_gasaerexch_sub, 0.00708636),
(g_soa_tend__modal_aero_soaexch, 0.00706446),
(invariants__gas_phase_chemdr, 0.00693014),
(forcing__imp_sol, 0.00673023),
(max__1061__modal_aero_soaexch, 0.00630873),
(a_poa_in__modal_aero_soaexch, 0.00630873),
(a_soa_in__modal_aero_soaexch, 0.00630873),
(num_a__gas_aer_uptkrates, 0.00630873),
(tmpn__modal_aero_coag_sub, 0.00610154),
(max__306__modal_aero_coag_sub, 0.00610154),
(dqdt_other__modal_aero_rename_sub, 0.00602613),
(tmp_q3__modal_aero_newnuc_sub, 0.00545362),
(qh2so4_cur__mer07_veh02_nuc_mosaic_1box, 0.00518261),
(qnh3_cur__modal_aero_newnuc_sub, 0.00518261),
(max__264__modal_aero_newnuc_sub, 0.00518151),
(dqdt_wr__sox_cldaero_update, 0.00499887),
(iter_invariant__imp_sol, 0.00470764),
(sum_dqdt_nh4_b__modal_aero_gasaerexch_sub, 0.00445509),
(o3g__setsox, 0.0044033),
(ho2s__setsox, 0.0044033),
(nh3g__setsox, 0.00434739),
(xh2so4__sox_cldaero_update, 0.00434739),
(xmsa__sox_cldaero_update, 0.00434739),
(xnh3__sox_cldaero_update, 0.00434739),
(tmp1__modal_aero_gasaerexch_sub, 0.00387178),
(delso4_o3rxn__sox_cldaero_update, 0.00384742),
(delso4_hprxn__sox_cldaero_update, 0.00384742),
(ko__usrrxt, 0.00372702),
(rate__adjrxt, 0.00355853),
(p_rate__o1d_to_2oh_adj, 0.00341498),
(max__260__modal_aero_newnuc_sub, 0.00335142),
(log10__727__usrrxt, 0.0032279),
(loss__imp_prod_loss, 0.00310991),
(prod__imp_prod_loss, 0.00310991),
(xso4_init__sox_cldaero_update, 0.00307298),
(g_soa__modal_aero_soaexch, 0.0029987),
(tv__drydep_xactive, 0.00281063),
(term__drydep_fromlnd, 0.00268962),
(tmp_q2__modal_aero_newnuc_sub, 0.00268024),
(xr__usrrxt, 0.00267682),
(tmpa__modal_aero_soaexch, 0.00266351),
(min__577__modal_aero_coag_sub, 0.00255798),
(qmax_nh4__modal_aero_gasaerexch_sub, 0.00239144),
(extfrc__gas_phase_chemdr, 0.00233497),
(ccc__setsox, 0.00225798),
(rate__setrxt, 0.00222156),
(reaction_rates__imp_sol, 0.00222156),
(reaction_rates__exp_sol, 0.00222156),
(p_rate__phtadj, 0.00222156),
(tmp2__modal_aero_gasaerexch_sub, 0.0021256),
(a_soa_tend__modal_aero_soaexch, 0.0021001),
(qh2so4_avg__modal_aero_newnuc_sub, 0.00209201),
(max__573__modal_aero_coag_sub, 0.00205579),
(log__282__modal_aero_newnuc_sub, 0.00204988),
(tmp_n2__mer07_veh02_nuc_mosaic_1box, 0.00204739),
(xnumbconcavg__modal_aero_coag_sub, 0.0020318),
(a_soa__modal_aero_soaexch, 0.00202442),
(dryvol_t_del__modal_aero_rename_sub, 0.00201954),
(tmp1__modal_aero_coag_sub, 0.0019927),
(dqdt__sox_cldaero_update, 0.00197235),
(xnumbconcnew__modal_aero_coag_sub, 0.0019606),
(dso4dt_aqrxn__sox_cldaero_update, 0.00193924),
(exp__849__gas_phase_chemdr, 0.00191567),
(b__lu_slv, 0.00185352),
(invariants__usrrxt, 0.00174652),
(xhnm__imp_sol, 0.00174652),
(invariants__aero_model_gasaerexch, 0.00174652),
(inv__adjrxt, 0.00174652),
(airdens__aero_model_gasaerexch, 0.00174652),
(inv__o1d_to_2oh_adj, 0.00174652),
(m__usrrxt, 0.00174652),
(a_opoa__modal_aero_soaexch, 0.00169774),
(tot_soa__modal_aero_soaexch, 0.00169244),
(tmpc__modal_aero_coag_sub, 0.00167494),
(tmp_n1__mer07_veh02_nuc_mosaic_1box, 0.00163527),
(max__1064__modal_aero_soaexch, 0.00158991),
(const__gas_aer_uptkrates, 0.00158991),
(min__584__modal_aero_gasaerexch_sub, 0.00151145),
(dqdt_aq__sox_cldaero_update, 0.00146475),
(term1__usrrxt, 0.00137227),
(delnh3__sox_cldaero_update, 0.00137173),
(min__1012__mer07_veh02_nuc_mosaic_1box, 0.00134368),
(freducea__mer07_veh02_nuc_mosaic_1box, 0.00131209),
(qnh3_cur__mer07_veh02_nuc_mosaic_1box, 0.00130611),
(phi__modal_aero_soaexch, 0.00130372),
(term2__usrrxt, 0.00128511),
(sat__modal_aero_soaexch, 0.00124678),
(a_soa_tmp__modal_aero_soaexch, 0.00122504),
(g_star__modal_aero_soaexch, 0.00113313),
(rxt__indprd, 0.00111975),
(r1h2o2__setsox, 0.00110971),
(nh3g__sox_cldaero_update, 0.00109562),
(dmsadt_gasuptk__sox_cldaero_update, 0.00109562),
(dso4dt_gasuptk__sox_cldaero_update, 0.00109562),
(tmpc__modal_aero_newnuc_sub, 0.00107538),
(max__1118__modal_aero_soaexch, 0.0010413),
(qcw__sox_cldaero_update, 0.0010169),
(dso4dt_hprxn__sox_cldaero_update, 0.000969618),
(ko_m__usrrxt, 0.000939274),
(log__745__usrrxt, 0.000939274),
(log10__740__usrrxt, 0.000939274),
(log10__735__usrrxt, 0.000939274),
(molenh4a_per_moleso4a__mer07_veh02_nuc_mosaic_1box, 0.000928095),
(tmpa__modal_aero_newnuc_sub, 0.000851864),
(dflx__drydep_fromlnd, 0.00079889),
(loss__imp_sol, 0.000783751),
(prod__imp_sol, 0.000783751),
(max__1155__modal_aero_soaexch, 0.000755726),
(tmpb__modal_aero_newnuc_sub, 0.000734965),
(term__drydep_xactive, 0.000708334),
(dtcur__modal_aero_soaexch, 0.000685929),
(min__1013__mer07_veh02_nuc_mosaic_1box, 0.000681509),
(xferfrac_pcage__modal_aero_coag_sub, 0.000644657),
(b__lu_slv01, 0.000624508),
(extfrc__exp_sol, 0.000588454),
(extfrc__imp_sol, 0.000588454),
(max__831__setsox, 0.000569051),
(max__784__setsox, 0.000569051),
(lrxt__imp_sol, 0.000559873),
(tmpa__modal_aero_coag_sub, 0.000544571),
(qh2so4_avg__mer07_veh02_nuc_mosaic_1box, 0.000527224),
(tmp_m2__mer07_veh02_nuc_mosaic_1box, 0.000515978),
(tmp_n3__mer07_veh02_nuc_mosaic_1box, 0.000515978),
(dumloss__modal_aero_coag_sub, 0.000512049),
(dryvol_t_new__modal_aero_rename_sub, 0.00050896),
(dqdt_aqso4__sox_cldaero_update, 0.000496298),

(tmpb__modal_aero_soaexch, 0.000478374),

(qh2so4_del__mer07_veh02_nuc_mosaic_1box, 0.000452728),

(lwc__usrrxt, 0.000440153),

(o2__usrrxt, 0.000440153),

(xhnm__setsox, 0.000440153),

(invariants__setsox, 0.000440153),

(o2_rate__o1d_to_2oh_adj, 0.000440153),

(n2_rate__o1d_to_2oh_adj, 0.000440153),

(h2o_rate__o1d_to_2oh_adj, 0.000440153),

(min__991__mer07_veh02_nuc_mosaic_1box, 0.000428861),

(max__1067__modal_aero_soaexch, 0.000427861),

(exp__520__modal_aero_coag_sub, 0.000422113),

(exp__433__modal_aero_coag_sub, 0.000422113),

(exp__496__modal_aero_coag_sub, 0.000422113),

(tmp_m1__mer07_veh02_nuc_mosaic_1box, 0.000412117),

(prod__indprd, 0.000408883),

(uptkrate__gas_aer_uptkrates, 0.000400686),

(max__393__modal_aero_newnuc_sub, 0.000399909),

(freduceb__mer07_veh02_nuc_mosaic_1box, 0.000389616),

(xferfrac_pcage__modal_aero_gasaerexch_sub, 0.000380911),

(kgaero_per_moleso4a__mer07_veh02_nuc_mosaic_1box, 0.000356516),

(tmpb__mer07_veh02_nuc_mosaic_1box, 0.00035494),

(dqdt_aqh2so4__sox_cldaero_update, 0.000353277),

(xferfracvol__modal_aero_coag_sub, 0.000347642),

(log10__621__usrrxt, 0.000345835),

(delnh4__sox_cldaero_update, 0.000345701),

(abs__1119__modal_aero_soaexch, 0.00032856),

(dflx__drydep_xactive, 0.000307777),

(extfrc__indprd, 0.000296601),

(r2h2o2__setsox, 0.000279666),

(sflx__gas_phase_chemdr, 0.0002789),

(qcw__setsox, 0.00027696),
(dmsadt_gasuptk_toso4__sox_cldaero_update, 0.000276115),
(dmsadt_gasuptk_tomsa__sox_cldaero_update, 0.000276115),
(dens_part__mer07_veh02_nuc_mosaic_1box, 0.000266668),
(max__440__sox_cldaero_update, 0.000256277),
(max__264__sox_cldaero_update, 0.000256277),
(max__432__sox_cldaero_update, 0.000256277),
(max__436__sox_cldaero_update, 0.000256277),
(qmolnh4a_del_max__mer07_veh02_nuc_mosaic_1box, 0.000239879),
(log10__821__usrrxt, 0.000236714),
(tcur__modal_aero_soaexch, 0.00023111),
(qnh3_del__mer07_veh02_nuc_mosaic_1box, 0.000229621),
(mass1p_aitlo__modal_aero_newnuc_sub, 0.000214685),
(mass1p_aithi__modal_aero_newnuc_sub, 0.000214685),
(tvs__gas_phase_chemdr, 0.000201336),
(max__395__modal_aero_newnuc_sub, 0.000185224),
(exp__294__modal_aero_newnuc_sub, 0.000185224),
(tmp_uptkrate__modal_aero_newnuc_sub, 0.000185224),
(xferfrac_vol__modal_aero_rename_sub, 0.000180183),
(beta__modal_aero_soaexch, 0.000173021),
(min__359__modal_aero_rename_sub, 0.000160592),
(rxt__imp_prod_loss, 0.000141098),
(rxt__linmat, 0.000141098),
(exp__498__modal_aero_coag_sub, 0.000137241),
(exp__435__modal_aero_coag_sub, 0.000137241),
(exp__522__modal_aero_coag_sub, 0.000137241),
(so4vol_in__mer07_veh02_nuc_mosaic_1box, 0.00013287),
(tmp_m3__mer07_veh02_nuc_mosaic_1box, 0.000130036),
(exp__453__modal_aero_coag_sub, 0.000129045),
(max__542__modal_aero_coag_sub, 0.000129045),
(exp__584__modal_aero_coag_sub, 0.000129045),
(exp__543__modal_aero_coag_sub, 0.000129045),
(dum__modal_aero_rename_sub, 0.000128267),
(dmdt_ait__modal_aero_newnuc_sub, 0.000123557),
(gr_kk__mer07_veh02_nuc_mosaic_1box, 0.000123303),

(qso4a_del__mer07_veh02_nuc_mosaic_1box, 0.000114095),

(solar_flux__drydep_fromlnd, 0.000109383),

(freduce__mer07_veh02_nuc_mosaic_1box, 0.00010808),

(ind_prd__exp_sol, 0.000103046),

(ind_prd__imp_sol, 0.000103046),

(uptkrate__modal_aero_gasaerexch_sub, 0.00010098),

(tmp_frso4__modal_aero_newnuc_sub, 0.000100784),

(tmpe__mer07_veh02_nuc_mosaic_1box, 9.42025e-05),

(cs_prime_kk__mer07_veh02_nuc_mosaic_1box, 9.38053e-05),

(dndt_ait__modal_aero_newnuc_sub, 9.03608e-05),

(wet_volfrac_so4a__mer07_veh02_nuc_mosaic_1box, 8.97277e-05),

(vmrcw__aero_model_gasaerexch, 8.20702e-05),

(ocnice_dflx__drydep_fromlnd, 7.75653e-05),

(fsds__gas_phase_chemdr, 7.75653e-05),

(lchnk__gas_phase_chemdr, 7.75653e-05),

(nu_kk__mer07_veh02_nuc_mosaic_1box, 7.16698e-05),

(spec_hum__drydep_xactive, 7.02901e-05),

(pressure_sfc__drydep_fromlnd, 7.02896e-05),

(qnum_c__sox_cldaero_update, 6.89665e-05),

(gamma_kk__mer07_veh02_nuc_mosaic_1box, 6.72753e-05),

(mass_part__mer07_veh02_nuc_mosaic_1box, 6.7205e-05),

(qnuma_del__mer07_veh02_nuc_mosaic_1box, 6.02747e-05),

(dtmax__modal_aero_soaexch, 5.82438e-05),

(qnh4a_del__mer07_veh02_nuc_mosaic_1box, 5.78684e-05),

(dso4dt_ait__modal_aero_newnuc_sub, 5.65378e-05),

(dnh4dt_ait__modal_aero_newnuc_sub, 5.65378e-05),

(dqdt__modal_aero_newnuc_sub, 5.12696e-05),

(air_temp__drydep_fromlnd, 5.07403e-05),

(solar_flux__drydep_xactive, 4.71144e-05),

(h2so4_uptkrate__mer07_veh02_nuc_mosaic_1box, 4.66798e-05),

(min__360__modal_aero_rename_sub, 4.54093e-05),

(xfercoef__modal_aero_rename_sub, 4.54093e-05),
(rxt__linmat01, 3.55592e-05),
(so4vol_bb__mer07_veh02_nuc_mosaic_1box, 3.34855e-05),
(faqgain_so4__sox_cldaero_update, 3.00623e-05),
(faqgain_msa__sox_cldaero_update, 3.00623e-05),
(qso4a_del__modal_aero_newnuc_sub, 2.8754e-05),
(qqcw__modal_aero_gasaerexch_sub, 2.80097e-05),
(dvmrcwdt__aero_model_gasaerexch, 2.76519e-05),
(uptkratebb__modal_aero_gasaerexch_sub, 2.54487e-05),
(qmolso4a_del_max__mer07_veh02_nuc_mosaic_1box, 2.37407e-05),
(z__drydep_xactive, 2.12228e-05),
(tha__drydep_xactive, 2.09403e-05),
(vmr__qqcw2vmr, 2.06832e-05),
(qnuma_del__modal_aero_newnuc_sub, 2.03084e-05),
(sumf__sox_cldaero_update, 2.02578e-05),
(icefrc__drydep_xactive, 1.95478e-05),
(thg__drydep_xactive, 1.88448e-05),
(exp__962__mer07_veh02_nuc_mosaic_1box, 1.81324e-05),
(pressure_sfc__drydep_xactive, 1.77142e-05),
(max__270__sox_cldaero_update, 1.73808e-05),
(tmpa__mer07_veh02_nuc_mosaic_1box, 1.72775e-05),
(max__969__mer07_veh02_nuc_mosaic_1box, 1.72297e-05),
(ribn__drydep_xactive, 1.64193e-05),
(qnh4a_del__modal_aero_newnuc_sub, 1.45839e-05),
(air_temp__drydep_xactive, 1.27951e-05),
(crs__drydep_xactive, 1.1875e-05),
(rdc__drydep_xactive, 1.18736e-05),
(xfertend__modal_aero_rename_sub, 1.14439e-05),
(hvar__drydep_xactive, 1.14391e-05),
(fgain_soa__modal_aero_gasaerexch_sub, 9.67247e-06),
(fgain_so4__modal_aero_gasaerexch_sub, 9.67247e-06),
(fgain_nh4__modal_aero_gasaerexch_sub, 9.67247e-06),
(mat__linmat01, 9.61353e-06),
(zovl__drydep_xactive, 8.55816e-06),
(so4vol__pbl_nuc_wang2008, 8.43894e-06),

(so4vol__binary_nuc_vehk2002, 8.43894e-06),

(xmol__drydep_xactive, 8.42215e-06),

(h__drydep_xactive, 8.08204e-06),

(sqrt__2606__drydep_xactive, 6.97066e-06),

(sqrt__2625__drydep_xactive, 6.97066e-06),

(dqqcwdt_other__modal_aero_gasaerexch_sub, 6.96877e-06),

(ustar__drydep_xactive, 6.49201e-06),

(sqrt__2629__drydep_xactive, 5.90943e-06),

(sqrt__2609__drydep_xactive, 5.90943e-06),

(log__2584__drydep_xactive, 5.80577e-06),

(sqrt__2589__drydep_xactive, 5.80577e-06),

(sqrt__2624__drydep_xactive, 5.76085e-06),

(log__2628__drydep_xactive, 5.76085e-06),

(log__2623__drydep_xactive, 5.76085e-06),

(ustarb__drydep_xactive, 5.66727e-06),

(cvar__drydep_xactive, 5.59951e-06),

(log__2608__drydep_xactive, 5.34852e-06),

(sqrt__2605__drydep_xactive, 5.34852e-06),

(log__2604__drydep_xactive, 5.34852e-06),

(lcl_frc_landuse__drydep_xactive, 4.92765e-06),

(sqrt__2590__drydep_xactive, 4.59962e-06),

(factor_kk__mer07_veh02_nuc_mosaic_1box, 4.56968e-06),

(pg__drydep_xactive, 4.46429e-06),

(log__828__mer07_veh02_nuc_mosaic_1box, 4.35423e-06),

(max__951__mer07_veh02_nuc_mosaic_1box, 4.35423e-06),

(dqqcwdt__modal_aero_rename_sub, 4.21359e-06),

(b__drydep_xactive, 4.21093e-06),

(min__2508__drydep_xactive, 4.13795e-06),

(vds__drydep_xactive, 3.75863e-06),

(wrk__drydep_xactive, 3.64803e-06),

(dep_ra__drydep_xactive, 3.59083e-06),

(resc__drydep_xactive, 3.41125e-06),
(sum_uprt_soa__modal_aero_gasaerexch_sub, 3.25894e-06),
(sum_uprt_so4__modal_aero_gasaerexch_sub, 3.25894e-06),
(sum_uprt_nh4__modal_aero_gasaerexch_sub, 3.25894e-06),
(rs__drydep_xactive, 2.99271e-06),
(mat__linmat, 2.58709e-06),
(exp__1386__binary_nuc_vehk2002, 2.46718e-06),
(uptkrate_soa__modal_aero_gasaerexch_sub, 2.43763e-06),
(min__2664__drydep_xactive, 2.15681e-06),
(max__2659__drydep_xactive, 2.15681e-06),
(psih__drydep_xactive, 2.15681e-06),
(log__1256__binary_nuc_vehk2002, 2.12676e-06),
(log__1316__binary_nuc_vehk2002, 2.12676e-06),
(tmp_ratenucl__pbl_nuc_wang2008, 2.12676e-06),
(deldryvol_c__modal_aero_rename_sub, 2.01142e-06),
(bb__drydep_xactive, 1.8319e-06),
(z0b__drydep_xactive, 1.81433e-06),
(dqqcwdt_other__modal_aero_rename_sub, 1.75625e-06),
(z0water__drydep_xactive, 1.6361e-06),
(dep_rb__drydep_xactive, 1.6361e-06),
(cvarb__drydep_xactive, 1.46316e-06),
(uustar__drydep_xactive, 1.42965e-06),
(lnd_frc__drydep_xactive, 1.24185e-06),
(dvel__drydep_xactive, 1.22913e-06),
(ratenuclt_kk__mer07_veh02_nuc_mosaic_1box, 1.16218e-06),
(log__2499__drydep_xactive, 1.12645e-06),
(wetvol_dryvol__mer07_veh02_nuc_mosaic_1box, 1.09734e-06),
(dqqcwdt__modal_aero_gasaerexch_sub, 1.0619e-06),
(rds__drydep_xactive, 9.47241e-07),
(tmpa__binary_nuc_vehk2002, 9.35836e-07),
(max__2866__drydep_xactive, 8.59695e-07),
(exp__431__modal_aero_gasaerexch_sub, 8.21311e-07),
(exp__433__modal_aero_gasaerexch_sub, 8.21311e-07),
(exp__432__modal_aero_gasaerexch_sub, 8.21311e-07),
(rsmx__drydep_xactive, 8.02119e-07),
(exp__2739__drydep_xactive, 7.54216e-07),
(lin_jac__imp_sol, 6.51993e-07),
(cnum_tot__binary_nuc_vehk2002, 6.21773e-07),
(xferrate__modal_aero_soaexch, 6.14326e-07),
(ratenucl__pbl_nuc_wang2008, 5.77797e-07),
(crit_x__binary_nuc_vehk2002, 5.35981e-07),
(log__1186__pbl_nuc_wang2008, 5.35981e-07),
(exp__2583__drydep_xactive, 4.57243e-07),
(depvel__gas_phase_chemdr, 3.37736e-07),
(ocnfrac__gas_phase_chemdr, 3.09763e-07),
(ocnice_dvel__drydep_fromlnd, 3.09763e-07),
(snowhland__gas_phase_chemdr, 3.09763e-07),
(max__1025__mer07_veh02_nuc_mosaic_1box, 2.9289e-07),
(cnum_h2so4__binary_nuc_vehk2002, 2.91774e-07),
(dfin_kk__mer07_veh02_nuc_mosaic_1box, 2.76672e-07),
(min__1331__binary_nuc_vehk2002, 2.35847e-07),
(rateloge__binary_nuc_vehk2002, 2.35847e-07),
(exp__1332__binary_nuc_vehk2002, 2.35847e-07),
(avg_uprt_soa__modal_aero_gasaerexch_sub, 2.06985e-07),
(avg_uprt_so4__modal_aero_gasaerexch_sub, 2.06985e-07),
(avg_uprt_nh4__modal_aero_gasaerexch_sub, 2.06985e-07),
(dv_pan__drydep_xactive, 1.90076e-07),
(ratenuclt__mer07_veh02_nuc_mosaic_1box, 1.65922e-07),
(lmat__nlnmat, 1.64314e-07),
(tmp_rateloge__pbl_nuc_wang2008, 1.35077e-07),
(exp__1404__binary_nuc_vehk2002, 1.35077e-07),
(gcoe__binary_nuc_vehk2002, 1.35077e-07),
(dcoe__binary_nuc_vehk2002, 1.35077e-07),
(fcoe__binary_nuc_vehk2002, 1.35077e-07),
(ccoe__binary_nuc_vehk2002, 1.35077e-07),
(jcoe__binary_nuc_vehk2002, 1.35077e-07),
(acoe__binary_nuc_vehk2002, 1.35077e-07),
(hcoe__binary_nuc_vehk2002, 1.35077e-07),
(ecoe__binary_nuc_vehk2002, 1.35077e-07),
(icoe__binary_nuc_vehk2002, 1.35077e-07),
(bcoe__binary_nuc_vehk2002, 1.35077e-07),
(dvelocity__drydep_fromlnd, 1.10997e-07),
(mmr__drydep_xactive, 9.46652e-08),
(wind_speed__drydep_xactive, 8.70377e-08),
(sfc_temp__drydep_fromlnd, 8.51154e-08),
(rateloge__mer07_veh02_nuc_mosaic_1box, 8.38826e-08),
(cnum_h2so4__mer07_veh02_nuc_mosaic_1box, 7.85218e-08),
(pressure_10m__drydep_fromlnd, 7.80657e-08),
(ocnfrac__drydep_fromlnd, 7.80657e-08),
(ratenucl__binary_nuc_vehk2002, 5.94377e-08),
(rateloge__pbl_nuc_wang2008, 5.51816e-08),
(log__768__mer07_veh02_nuc_mosaic_1box, 4.18152e-08),
(ratenuclt_bb__mer07_veh02_nuc_mosaic_1box, 4.18152e-08),
(lmat__nlnmat_finit, 4.14099e-08),
(radius_cluster__binary_nuc_vehk2002, 3.40417e-08),
(mmr__gas_phase_chemdr, 3.02658e-08),
(max__2503__drydep_xactive, 2.1935e-08),
(pressure_10m__drydep_xactive, 2.15962e-08),
(sfc_temp__drydep_xactive, 2.14506e-08),
(exp__822__mer07_veh02_nuc_mosaic_1box, 2.11399e-08),
(cnum_h2so4__pbl_nuc_wang2008, 1.97982e-08),
(max__833__mer07_veh02_nuc_mosaic_1box, 1.97889e-08),
(ocnfrc__drydep_fromlnd, 1.96739e-08),
(lndfrac__drydep_fromlnd, 1.96739e-08),
(mat__nlnmat_finit, 1.53512e-08),
(radius_cluster__mer07_veh02_nuc_mosaic_1box, 9.16095e-09),
(q__get_short_lived_species, 9.09704e-09),
(wind_speed__drydep_fromlnd, 7.62752e-09),
(mmr__mmr2vmr, 7.62752e-09),
(mmr__drydep_fromlnd, 7.62752e-09),
(mmr__set_mean_mass, 7.62752e-09),

(qh2o__gas_phase_chemdr, 7.62752e-09),

(voldry_clus__mer07_veh02_nuc_mosaic_1box, 6.66746e-09),

(va__drydep_xactive, 5.52802e-09),

(p__drydep_xactive, 5.44262e-09),

(tc__drydep_xactive, 5.40593e-09),

(ocnfrc__drydep_xactive, 4.95818e-09),

(mat__nlnmat, 4.15217e-09),
```