REFINING THE COMPOSITION OF CESM-ECT ENSEMBLES

by

DANIEL J. MILROY

B.A., University of Chicago, 2006

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

Master of Science

Department of Computer Science

2015

This thesis entitled:
REFINING THE COMPOSITION OF CESM-ECT ENSEMBLES
written by DANIEL J. MILROY
has been approved for the Department of Computer Science

Elizabeth Jessup

Allison Baker

Dorit Hammerling

Thomas Hauser

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

MILROY, DANIEL J. (M.S., Computer Science)

REFINING THE COMPOSITION OF CESM-ECT ENSEMBLES

Thesis directed by Prof. Elizabeth Jessup

Complex, modular codes such as climate simulations are in a constant state of development, requiring frequent verification. Historically, this process has been computationally expensive and dependent on the interpretation of a climate scientist. In this research, we utilize the recently developed Community Earth System Model (CESM) Ensemble Consistency Test (CESM-ECT) to provide an objective measure of consistency for new CESM simulation runs. This tool relies on the creation of an ensemble of simulations that represent the same climate system, and whose statistical distribution embodies the innate variability of the climate model. A CESM ensemble is created by perturbations to the initial temperature; in this work we investigate whether such perturbations are adequate for characterizing the variability of a consistent climate. To this end we develop a systematic method for introducing minimal changes to the Community Atmospheric Model (CAM) component of CESM which are not expected to modify the climate, and measure the resultant numerical differences and their effects on the output ensemble with CESM-ECT. These minimal changes include mathematically identical reformulations (e.g. order of operation alteration and exchanging division with multiplication by inverse), compiler groups and versions, levels of compiler optimization, and substitution of the random number generator. Modifications to CAM modules are selected to be minimal in the sense that 1: the number of lines of modified code is minimized (i.e. limited to one distinct code block or math kernel), and 2: the alteration should not be climate-changing. These types of changes should translate to a CESM-ECT pass, and we examined the CESM-ECT ensemble in that context. In fact, the primary objective of this research was to ensure that the constituents of the CESM-ECT ensemble provide sufficient variability to represent a consistent climate, in effect verifying the CESM-ECT ensemble composition.

In this thesis, we demonstrate that the current CESM-ECT ensemble does not contain ade-

quate variability to capture such minimal changes, and we suggest means of improvement to reach the desired false positive rate.

Acknowledgements

Much credit and gratitude are due Allison Baker and Dorit Hammerling for their assistance with this thesis, coaching me through mock presentations, and for the considerable time they spent teaching me. Their patient instruction on the details of the inward workings of CESM and the statistics of ensemble consistency are deeply appreciated. I wish to thank Thomas Hauser and Elizabeth Jessup for their support throughout the course of the Masters program. Thanks are also due to Youngsung Kim for fruitful discussions on assembly code instructions.

This research used computing resources including Yellowstone (ark:/85065/d7wd3xhc) provided by the Climate Simulation Laboratory at the NCAR Computational and Information Systems Laboratory (CISL), sponsored by the National Science Foundation and other agencies.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Modeling the Earth's climate is a formidable task, requiring considerable human resources to devise and encode correct mathematical representations of physical processes. These processes typically cover spatial extents of less than a millimeter to thousands of kilometers, with a comparable range of time scales. The code base for these models can be millions of lines and represent decades of software development time for teams of engineers and scientists. A correspondingly complex range of machinery can be required to execute the code and produce results in a timely manner.

Concomitant to vastly different execution scales, runtime hardware, and development is the question about validity and veracity of the results. The great assortment of platforms on which Global Climate Models (GCMs) run and the vast number of available parameters and configurations used to tune their behavior (Pipitone and Easterbrook, 2012) translate to a corresponding difficulty in detecting simulation error. Easterbrook and Johns (2009) provide a detailed overview of identifying incorrect implementations of physical phenomena and algorithms in GCMs via Verification and Validation strategies. They highlight one such mechanism, the bit-for-bit (BFB) test, which determines whether the outputs of two simulations are BFB identical. Such tests performed over short simulation time spans can be indicators of reproducibility over longer time intervals and gross errors in formula implementation (Easterbrook and Johns, 2009).

However, results in general will not be BFB identical, even if they represent the same climate. Small perturbations (within established tolerances) to initial conditions can result in non-BFB results, despite representing nearly identical climates. Given that the modularity of some GCMs,

such as NCAR's Community Earth System Model (CESM), facilitate runs on varied hardware and architectures, they are doubtless run on machines without hardware error correction. In the presence of persistent hard or intermittent soft errors, identical runs on the same machine could result in different climates. These errors are not typically a concern for machines comprised of enterprise hardware, but may affect personal machines and commercial, off-the-shelf (COTS) clusters. Moreover, the increasing prevalence of heterogeneous execution environments which employ GPUs or Xeon Phi coprocessors working in concert with CPUs increases the importance of defining an objective measure of consistency between non-BFB simulation outputs. For example, adapting a GCM to run on a new machine which may feature new compilers and hardware drivers can be a tremendously involved process. Until very recently, when CESM was ported to a new machine or architecture, a simulation of 400 years typically was run (Baker and Hammerling, 2015: hereafter designated BH2015). The output data were then compared to an identical run on a trusted machine, and the results were run through a diagnostics package and given to a senior scientist for approval- a time-consuming and largely subjective process.

Furthermore, the code base is in constant development in terms of bug fixing, algorithm and performance improvement, and feature introduction. A key component of verification is a procedure for identifying software bugs, akin to quality control. Clune and Rood (2011) delineate the challenges of assessing the software quality of GCMs, highlighting the extraordinary difficulty of the undertaking. A CESM codebase that correctly represents the evolution of the global climate may contain bugs, even in equation solving subroutines. Many elements of the time evolution of a climate are highly nonlinear, and introducing or eliminating modules (for example, by running on a single node versus a cluster) that represent different physical processes could conceivably cause an otherwise identical set of simulations (or components) to diverge or converge to a valid true climate depending on their stability properties. Any change in CESM code, hardware, compiler version, or the supporting software stack can alter the simulation output at least equivalent to truncation or round-off errors. We will demonstrate that even mathematically equivalent alterations to subroutines can produce numerically distinct output. 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.

1.1 CESM-ECT Approach

The difficulty of obtaining BFB identical results suggests an approach that utilizes relationships between members of a set of simulations to quantify consistency rather than verisimilitude or correctness. In BH2015 the CESM ensemble consistency test (CESM-ECT) was developed for measuring the consistency of an ensemble of CESM simulations. Given an ensemble of simulations from the original configuration, newly generated simulation output is designated "accepted" if it is statistically indistinguishable from the ensemble's distribution. Hereafter the original configuration is taken to mean accepted in that it is produced on a trusted machine with an accepted software stack. CESM-ECT is designed to accomplish three objectives: determine whether the new output is consistent with the original configuration, return a false positive rate, and provide ease-of-use and lack of computational expense (BH2015). CESM-ECT adheres to the coarse-grained approach in that its purpose is not to identify the cause of the discrepancy, but to identify that a discrepancy exists. An additional benefit of the CESM-ECT approach is that it allows for a measurement of the natural variability in the ensemble, which is studied here to the end of improving the ensemble's constituents and classification power.

1.2 Focus of this work

The basis of the work presented here is BH2015 which established the foundation of ensemble-based consistency testing. The selection of a representative or original configuration is critical to the accurate determination of whether new simulations pass. An improperly chosen ensemble, which in this context signifies that it is unrepresentative of the natural variability present in a consistent climate, will result in misleading CESM-ECT results. One implication of "improper" is that an ensemble could contain too much variability; that is not our focus, and coarse testing in BH2015 suggested this is not the case. As a consequence of the study's novelty, the properties of

CESM-ECT original configurations were not explored in depth in BH2015. Such examination is the principal charge of this thesis. Further, exploring the response of the Community Atmospheric Model component of CESM output under minimal code alterations as tested by CESM-ECT serves to quantify the concept of "minimal, non-climate changing modifications." Put another way, minimal code modifications that do not alter the mathematical structure of CESM, but that produce a measureable difference in CESM-ECT output should be classified as consistent. CESM-ECT should return a failure when, for example, a code modification, machine porting, or change of compiler encompasses a mathematical, coding, or hardware error that produces a distinct climate. This thesis presents examples of minimal, mathematically identical code changes, and includes a successful identification of numerical instability on a Leadership Class supercomputer that was discovered with the assistance of the refined CESM-ECT.

Chapter 2

Background

In this section we present information on which the contributions of this thesis are based.

2.1 CESM

The Community Earth System Model (CESM), principally developed at NCAR, is an open source global climate model that is available to the global climate research community. CESM can be configured to run on a laptop or occupy every core of a large-scale supercomputer.

CESM is composed of multiple geophysical models: atmosphere, ocean, land and sea ice, rivers, and others. These models exchange data via a coupling mechanism which can be tuned. Compilation templates exist for machine type (execution supercomputer) which include details of the machine's hardware configuration (cores per socket, memory per core, fabric interconnect and topology, etc...) and choose sensible settings for runtime. At runtime, CESM writes restart files at user-specified time intervals which contain double-precision floating point values. Output data for analysis are written in time slices to NetCDF history files, by default in single-precision floating point format. This point will be revisited in detail in the section discussing Community Atmospheric Model (CAM) modifications. The CESM-ECT project focuses on the CAM module due to the relatively short time scales for propagation of changes or perturbations in the atmosphere. The corollary of this is that shorter runtimes are needed to ensure sufficient data collection for the comparison. Additionally, CAM variables represent a diverse set, numbering approximately 130. Due to the coupling mechanism, it is assumed that changes propagate through the interface to

other models, but this does not imply that if CAM variables pass CESM-ECT then other models' variables will pass as well. Given the chaotic nature of the system, representing the inherent variability of CESM with a single simulation is difficult (BH2015).

2.2 CESM-ECT background

The CESM-ECT ensemble is a set of CESM simulations with identical parameters, differing only in initial atmospheric temperature conditions (BH2015). The ensemble members are run, evolving in time at the same rates for the same amount of time, and the initial temperature perturbations guarantee the creation of unique trajectories through the global models' phase space, in turn generating a distribution of output variables that can be analyzed for consistency. Both in BH2015 and our work in the later sections, perturbations of the initial atmospheric temperature field are generated in the closed interval $\left[-8.5 \times 10^{-14}, 8.5 \times 10^{-14}\right]$, and the members of the ensemble, as in the experimental sets of this work, are run for 12 months. One year is chosen as it is short enough to permit ensemble generation in a reasonable amount of time, and sufficient to generate a representative statistical distribution. The $O(10^{-14})$ perturbation is selected because it permits a divergence of the phase-space trajectories, but is expected to preserve the variables' distributions (BH2015). An ensemble of 151 members is chosen due to a lower-bound constraint imposed by the primary analytical technique: Principal Component Analysis (PCA- discussed in Section 2.2.1). PCA stipulates that the number of data points (i.e. ensemble members) be larger than the number of variables.

> In summary, the CESM-ECT ensemble consists of $N_{ens} = 151$ one-year climate simulations, denoted by $E = \{E_1, E_2, \ldots, E_{N_{ens}}\}$, and is produced on a trusted machine with an accepted version, model, and configuration of the climate code. The data for these one-year ensemble runs consists of annual temporal averages at each grid point for the selected grid resolution for all N_{var} variables, which are either two or three-dimensional. [...] We denote the dataset for a variable X as $X = \{x_1, x_2, \ldots, x_{N_x}\}$, where x_i is a scalar that represents the annual (temporal) average at grid point i and N_X is the total number of grid points in X[...]. (BH2015)

2.2.1 CESM-ECT: Principal Component Analysis

Excluding redundant variables and those with zero variance results in a set with the number of variables $N_{var} = 120$. In BH2015 a correlation analysis was performed, which concluded that "52 variables are highly correlated [>0.9]." Determining criteria such as false-positive rates required a projection of these variables into a space where the data are linearly independent.

Principal Component Analysis (PCA) is a common tool for dataset dimensionality reduction. As a linear operator, PCA is an orthogonal transformation (rotation) of data from an original space to a subspace. Although there are several precise methods of deriving the projection operator, it involves solving the eigenvalue problem for the covariance matrix of the original data. The eigenvectors can be sorted in descending order of their eigenvalues, forming an $n \times m$ matrix, where n is the number of dimensions selected for the subspace and m is the number of original variables. Typically, the choice of n is the result of a subjective optimization problem: what is the minimum number of linearly independent components in the transformed space that explains the maximum original dataset variance? Slightly more formally, given j eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_j\}$, we seek the smallest n such that $100 \times \frac{\sum_{i=1}^{n} \lambda_i}{\sum_{i=1}^{j} \lambda_k} \geq \alpha$ where α is some minimum acceptable percentage of variance explained.

In CESM-ECT, PCA is applied to the global means data. First, the $N_{var} \times N_{ens}$ data matrix is standardized in the usual way: subtract the ensemble mean, and scale by the ensemble standard deviation (unit variance), denoting the output V_{gm}. This is performed due to the data differing in scale by many orders of magnitude. Next, the eigenvalue problem of the covariance matrix of V_{gm} is solved, yielding the transformation matrix, or "loadings" $\mathbf{P}_{gm}$. Note that this matrix is of size $N_{var} \times N_{var}$ - superfluous components can be discarded after subspace projection. Finally, the matrix $\mathbf{V}_{gm}$ is rotated (projected) onto the subspace: $\mathbf{S}_{gm} = \mathbf{P}_{gm}^{\intercal} \mathbf{V}_{gm}$. Note that $\mathbf{S}_{gm}$ is of size $N_{var} \times N_{ens}$. The standard deviation of the ensemble scores is calculated and denoted $\sigma_{S_{gm}}$. In summary, these quantities are calculated and written to the CESM-ECT summary file:

- N_{var} means of global mean values ($\mu_{V_{gm}}$)

- N_{var} standard deviations of ensemble global mean values ($\sigma_{V_{gm}}$)
- $N_{var} \times N_{var}$ loadings $\mathbf{P}_{gm}$
- N_{var} standard deviations of ensemble global mean scores ($\sigma_{S_{gm}}$)

> The distribution of global mean scores from the ensemble, represented by the standard deviations in $\sigma_{S_{gm}}$, can be used to evaluate data from a new simulation. Note that most of the variance in the data is now largely represented by a few PCs. In fact, the coefficients on the first PC explain about 21% of the variance and the coefficients on the second explain about 17% of the variance. In this work we use $N_{PC} = 50$, as little additional variance is explained by the final 70 PCs. (BH2015)

2.2.2 CESM-ECT: Determining pass and fail

CESM-ECT seeks to determine whether given an "original ensemble" run on an accepted machine like Yellowstone and a small number of runs on a new machine, the results obtained from the new machine are statistically indistinguishable from the original ensemble. CESM-ECT then reports a pass or fail based on the consistency of the new runs with the ensemble.

As in the summary file computation, CESM-ECT calculates the weighted area global means for each variable in all the new runs (number of new runs = N_{new}). These means are standardized using the mean and standard deviations in the Yellowstone summary file ($\mu_{V_{gm}}$ and $\sigma_{V_{gm}}$). Next, the standardized means are rotated into the PC space of the original ensemble via the transformation matrix $\mathbf{P}_{gm}$ in the summary file. Third, the tool determines if the first N Principal Components (N_{PC}) of the new runs are within M_σ standard deviations of the original mean (zero in PC space), also using the standard deviation in PC space ($\sigma_{S_{gm}}$). Finally, for each member of the new runs, CESM-ECT labels each PC score outside M_σ as a fail (BH2015). Let P be the set of sets of failing PCs, grouped by new run ($P = \{X_i\}, i \in \{1, \ldots, N_{new}\}$). CESM-ECT then performs the following computations to determine an overall failure:

$$\bigcup_{\binom{N_{new}}{N_{RunFails}}} \left(\bigcap^{N_{RunFails}} X_i \right); i \in \{1, \ldots, N_{new}\}$$

If $|S| \geq N_{pcFails}$ the new runs are designated as a failure. In words, if at least $N_{pcFails}$ PCs fail at least $N_{runFails}$, CESM-ECT returns a failure. The parameters $M_\sigma, N_{new}, N_{pcFails}$, and $N_{runFails}$ are chosen to obtain a particular false positive rate. The false positive rate is the frequency at which the test returns a failure when the set of new runs should pass. In BH2015 an empirical study was performed to find values of the CESM-ECT parameters to yield a false positive rate of 0.5%. These parameters were chosen with the following values: $M_\sigma = 2$ (95% confidence level), $N_{new} = 3$, $N_{pcFails} = 3$, and $N_{runFails} = 2$. Let $P = \{A, B, C\}$ be the set of sets of failing PCs, then the above set S is expressed as:

$$S_{AB} = A \cap B, S_{AC} = A \cap C, S_{BC} = B \cap C \tag{2.1}$$

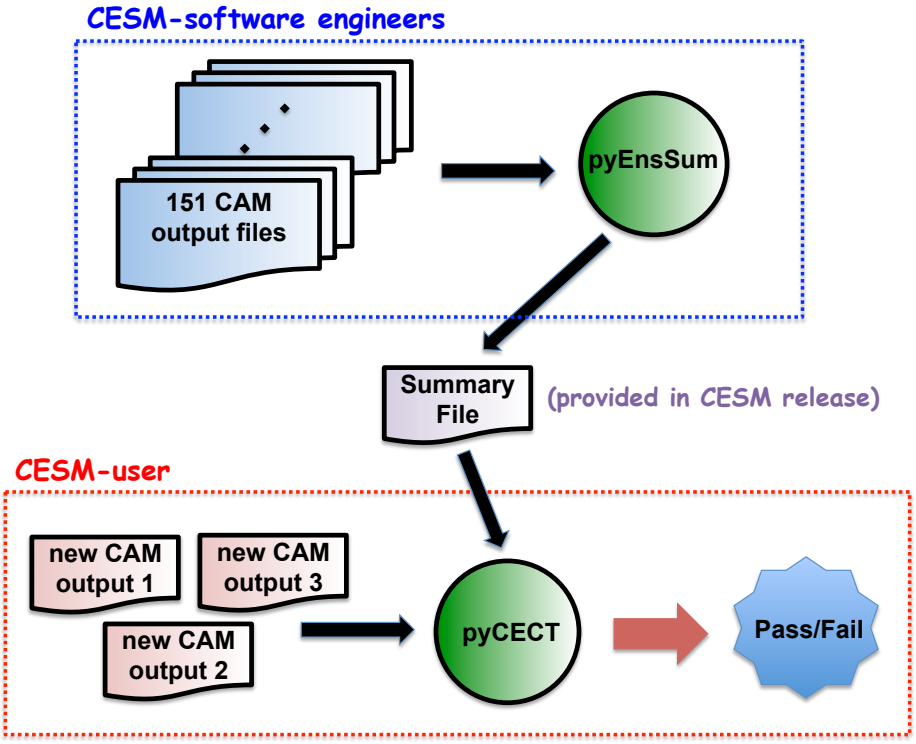
$$S = S_{AB} \cup S_{AC} \cup S_{BC} \tag{2.2}$$

Thus the new runs fail if $|S| \geq 3$. The false positive rate of 0.5% is selected to hedge "against the possibility that our ensemble may not be capturing all the variability that we want to accept" (BH2015). Also, "while perturbing the initial temperature condition is a common method of ensemble creation for studying climate variability, other possibilities exist, and we are currently conducting further research on the initial ensemble composition and its representation of the range of variability, particularly in regard to compilers and machine modifications" (BH2015). This is the interface between our research and BH2015.

2.2.3 pyCECT

The Python CESM Ensemble Consistency Tool (pyCECT) codifies the testing procedure defined above in a portable platform that can be used from any operating system. It is designed to be user-friendly and computationally inexpensive, and to yield clear, conclusive results. Broadly speaking, the workflow of consistency testing amounts to generating accepted output, generating a comprehensive and quantitative description of the accepted distribution, and determining whether the new data conform to the accepted data. See Figure 2.1 below.

Figure 2.1: Typical workflow for CESM-ECT, including two of its constituents. Appears in BH2015.



2.2.4 CESM-ECT: Examples of pass and failure

BH2015 performed three groups of tests to determine how CESM-ECT classifies different types of modifications. The first test consisted of modifications that were not expected to change the climate, the second likely climate-changing, and the third unknown. The results are discussed in the sections below. The results in this work were obtained from the 1.3 release series of CESM, using a current F compset (active atmosphere and land, data ocean, and prescribed ice mode) and CAM5. In BH2015 120 out of 134 CAM variables were examined, as some variables are redundant or possess no variance. Of the 120 variables, 78 are two-dimensional and 42 are three-dimensional. CAM5 uses an approximately one-degree resolution, which contains 48,602 horizontal gridpoints and 30 vertical levels. Simulations were run with 900 MPI processes and two OpenMP threads per process on the Yellowstone machine at NCAR. The default compiler on Yellowstone for our CESM version is Intel 13.1.2 with O2 optimization (BH2015).

2.2.4.1 Modifications not expected to be climate-changing

The following tests ($N_{new} = 3$) on Yellowstone produce non-BFB changes in output, but should not be climate changing:

NO_OPT: changing the Intel compiler flag to remove optimization (-O0)

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)

(BH2015)

Each test is comprised of simulations whose initial atmospheric temperatures are perturbed with a random value in the range used to create the original ensemble. Table 2.1 depicts the results.

Table 2.1: CESM modifications expected to produce same climate. Appears in BH2015.

Test name	CESM-ECT Results	Number PCs failing at least 2 runs
NO-OPT	PASS	1
INTEL-15	PASS	1
PGI	PASS	0
GNU	PASS	2

2.2.4.2 CAM parameter modifications, expected to be climate-changing

To test whether CESM-ECT correctly classifies modifications expected to result in a different climate, scientists provided a list of CAM input parameters that should produce a different climate under time evolution when substantially perturbed. The test results are included in Table 2.2 below. A complete explanation of the test variables can be found in BH2015. Noteworthy is the conclusiveness of failures in this test.

2.2.4.3 Modifications with unknown outcome

This group of tests considers each machine to be a modification. Note that the failures exhibit $N_{pcFails}$ only slightly greater than the classification threshold of two (Table 2.3); these are all borderline cases. Of these failures, Blue Waters and Mira are selected for further discussion in Section 7. Borderline cases such as these raise questions as to the composition of the accepted ensemble: does it contain enough variability that these two machines justifiably fail CESM-ECT?

Table 2.2: Modifications expected to change the climate. Appears in BH2015.

Test name	CESM-ECT Results	Number PCs failing at least 2 runs
DUST	FAIL	9
FACTB	FAIL	36
FACTIC	FAIL	43
RH-MIN-LOW	FAIL	44
RH-MIN-HIGH	FAIL	30
CLDFRC-DP	FAIL	27
UW-SH	FAIL	24
CONV-LND	FAIL	33
CONV-OCN	FAIL	45
NU-P	FAIL	35
NU	PASS	1

Table 2.3: Modifications with unknown outcome. Appears in BH2015.

Test name	CESM-ECT Results	Number PCs failing at least 2 runs
HOPPER	PASS	1
EDISON	PASS	1
TITAN	PASS	0
MIRA	FAIL	5
JANUS	PASS	1
BLUEWATERS	FAIL	5
EOS	FAIL	4
GOLDBACH-INTEL	PASS	0
GOLDBACH-PGI	PASS	0

Chapter 3

Verification of ensemble validity

Our contribution to CESM-ECT begins with a consideration of additional means of ensemble generation. This study endeavors to determine if the accepted ensemble contains sufficient variability that experiments which are not climate-changing are classified as such by CESM-ECT, in effect improving the constituents of the tool and its accuracy. As described in Section 2.2.4 a preliminary investigation into variability in CAM outputs under change of compiler, optimization order, and machine was performed in BH2015. Since generated object code depends greatly on the compiler type, version, and optimization order, it is difficult to predict program output differences given any such change. Moreover, detecting meaningful differences in object code generated from source code in excess of a million lines is a fearsome prospect. In particular, optimization order performs source code alterations to improve program performance. Examples of automatic code optimization include replacing power function calls with multiplication (x**2 becomes x*x), vectorization, and unroll-and-jam. These changes are similar to those that a programmer could make during the engineering or optimization of code. This raises the question as to the impact of mathematically identical code changes on the statistical properties of CAM variable outputs.

3.1 Experimental process

The first stage of this research consists of testing the ensembles against themselves, meaning that members of the ensemble (of various compositions and total sizes: 150, 151, 300, and 453) are passed to pyCECT and tested against summary files generated from all of the ensemble members.

Inter-ensemble tests are also included in this initial testing. The second stage of experimentation is concerned with determining minimal modifications and discerning their effects on the output CAM variables via CESM-ECT.

While initial temperature perturbations to CAM have been discussed and are familiar to scientists, the results of an order of operation change by a software engineer, or the automatic function substitution by a compiler had not been studied. Motivated by the need to understand the relationship between the effects of temperature perturbation, machine, different compilers, and code modification, we first performed a search for a "small" code change that would induce a difference in CAM single-precision output. More directly, we investigate this relationship to determine if perturbations to initial conditions such as temperature are sufficiently large to capture legitimate code changes, and can be considered components of the variability of a consistent climate. In an effort to minimize core hours consumed during testing, we adopted the strategy of running modified cases for 5 simulated days to determine if there was a measurable double-precision difference, and to run an initial simulation with 64 cores (threading disabled) before scaling up to the 900 MPI process, two thread-per-process simulation if preliminary outputs warranted it. If the CAM logs files differed from the control, a 12 month simulation was run. Code modification experiments consist of 30 simulations, and 24 members for Blue Waters and Mira machine tests so that the number is divisible by three, which is the preferred number of runs to compare per execution of pyCECT. We identify three categories of code modification experiments: those representative of coder preference, those representing optimization strategies, and subroutine substitution for potential performance improvement.

3.2 Software tools

To establish whether a modification had a measurable effect to single-precision, and merit extension to a full 30-member experiment, we used two common tools for comparing the NetCDF (single-precision) output: cprnc and nccmp, and GNU "diff" for the double-precision CAM log files. In generating CESM-ECT test results, the default behavior of pyCECT is to accept three

CAM output files and a desired summary file as input, read a single time slice (single-precision output after 12 months), rotate the global means of the three CAM files into the PC space of the summary file, and return the number of failing PCs (outside of two standard deviations from the summary file mean) from each experiment. If three or more PCs fail in two or more of the input CAM files, pyCECT returns an overall failure. Performing the consistency testing for the minor code modification experiments, as well as the machine experiments on ALCF Mira and NCSA Blue Waters, initially entailed feeding the members of the experiment set, three random examples at a time, to pyCECT. In the 30 member experiments this results in 10 total tests, and for the machine experiments, eight. This is a small sample of the total number of tests that can be performed with $N_{members}$, selecting N_{new} at a time; for example, two failures out of 10 is inconclusive. However, since running pyCECT on 30 members takes minutes, running all $\binom{30}{3} = 4060$ tests (or 2024 for the 24 member experiments) would be prohibitively time-consuming. By using pyCECT to provide the PC failures per experiment member, we developed a computationally efficient script (CESM Ensemble Exhaustive Test: EET) to read its output sets of failing PCs and perform all $\binom{N_{members}}{3}$ tests, rendering exhaustive testing feasible. Employing Python sets of dictionaries facilitates consistency testing between full ensembles as well. Indeed, performing 4060 tests takes a fraction of one second, and all 562,475 tests of a 151 member ensemble takes less than two seconds.

Chapter 4

Ensemble member experiments

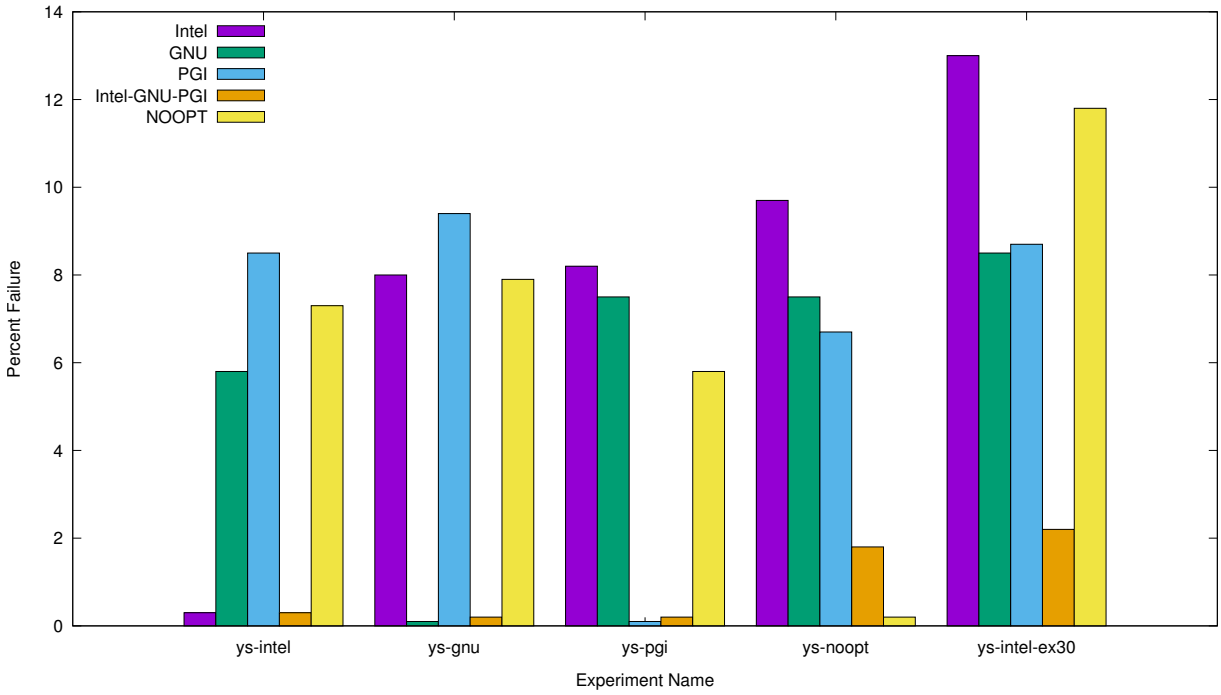
The first phase in verification of the accepted ensemble is to perform consistency testing on its members. These members should pass, as they are consistent by definition. Tests performed on members against summary files generated from the same members (i.e. Intel members tested against the Intel summary file) are expected to manifest error rates approximately equal to our desired false-positive rate (0.5%).

4.1 Self-tests

Empowered by EET, we were able to fully test all four Yellowstone original ensembles (Intel, GNU, PGI, NO-OPT) from BH2015 against themselves. Each of these 151 member ensembles is named for the compiler or optimization used to build it. The NO-OPT ensemble was built with the Intel compiler, disabling optimization (O0). The Intel CESM-ECT ensemble summary is the accepted CESM-ECT ensemble used throughout BH2015 and in current CESM testing. We also include a fifth ensemble, labeled Intel-GNU-PGI, which was created by combining the 151-member Intel, GNU, and PGI ensembles into a single 453 member aggregate. The distribution represented by the Intel-GNU-PGI ensemble should be wider as it includes the variability due to each compiler as well as the variability introduced by perturbations to the initial conditions. The results are presented in Figure 4.1.

The y-axis denotes the percent failure of all possible tests as calculated by EET. The first four experiments' elements are simulations whose executables were built against the designated compiler,

Figure 4.1: EET failure percentage grouped by experiment. Colors indicate ensemble summary file used in comparison, "ys" abbreviates Yellowstone. "Intel" is the CESM-ECT accepted ensemble in BH2015.



and run on NCAR's Yellowstone supercomputer. The "ys-intel-ex30" experiment is discussed in section 4.2.

Note the failure percentages of the 151 member ensembles. For the corresponding numerical values, consult the link to data in Section 10. As expected, the failure percentages are low (<1%) when the ensemble is tested against its own summary file, but are typically order <10% between compilers. Interestingly, NO-OPT (Intel O0) tested against Intel (O2 optimization) exhibits a similar failure rate to between-compiler tests. Because the GNU and PGI compilers on Yellowstone are CESM-supported configurations, they are expected to pass. Note that the results from BH2015 (Table 2.1) test only one random selection of three runs from PGI and GNU, both of which passed. The EET capability was needed to expose the true failure rate. However, note the lower failure rates of the ys-intel, ys-gnu, and ys-pgi tests against the size 453 Intel-GNU-PGI summary. Testing against this new summary is intended to be an overall assessment against the accepted Yellowstone configuration, by including the variability of the Intel, GNU, and PGI compilers.

The self-tests were intended as a first-order assessment for CESM-ECT. The limitation of self-testing is that the files used to generate the ensemble summaries (thus PCs) are used in the test itself. Furthermore, this is true for the size 453 Intel-GNU-PGI summary, which obfuscates the effectiveness of the added variability via the reuse of the test files for ensemble generation. This aliasing effect is dispelled in the ys-noopt and ys-intel-ex30 experiments.

4.2 Random selection experiments

The so-called yellowstone-intel-extra30 experiment (rightmost experiment in Figure 4.1) presents a remarkable anomaly- the 13.0% failure rate against the Intel ensemble summary. The "extra30" experiment consists of 30 members removed from an ensemble of 181 Intel simulations, yielding the yellowstone-intel 151 ensemble used by CESM-ECT. Extra30 is selected from the maximal and minimal elements of the set of initial temperature perturbations, namely the set of values in the disjoint intervals $\left[-9.9 \times 10^{-14}, -8.5 \times 10^{-14}\right)$, $\left(8.5 \times 10^{-14}, 9.9 \times 10^{-14}\right]$. Extra30's 13.0% failure rate against the Intel CESM-ECT ensemble is very high in comparison with the 0.5% false-positive

rate, given that it should not be climate-changing.

Therefore, for a more rigorous test of mutual-exclusivity of ensemble summary composition and experimental set, we created more ensemble summary files by randomly excluding 30 simulations from the 181 Yellowstone simulations available to us for each compiler (Intel, GNU, and PGI). We randomly selected three sets of 30 (labeled rand1, rand2, and rand3) to exclude from the 181, and these new ensemble summaries are labeled by the set of randomly excluded simulations, e.g. I-rand2 which corresponds to 151 Intel runs resulting from excluding the rand2 set from the 181 element Intel set. Note that the "rand2" label is consistent across summaries: P-rand2 excludes the same 30 simulations as I-rand2 and G-rand2. Exhaustive tests of the 30 randomly excluded simulations per compiler were run against these nine summary files, resulting in 81 tests (Table 4.1).

Table 4.1: Exhaustive testing failure rates of 30 randomly excluded simulations from 181 member ensembles (rand1, rand2, rand3) against summary files generated by excluding the corresponding simulations. For the experiment names, "ys" = Yellowstone, "i" = Intel, "g" = GNU, "p" = PGI.

Experiment	**I-rand1**	**I-rand2**	**I-rand3**	**G-rand1**	**G-rand2**	**G-rand3**	**P-rand1**	**P-rand2**	**P-rand3**
ys-i-e30-rand1	5.9	0.2	1.8	11.1	4.1	6.2	5.4	3.2	6.7
ys-i-e30-rand2	0.1	3.9	1.0	6.2	13.7	6.7	5	1.9	7.8
ys-i-e30-rand3	0.1	0.5	8.8	11.9	7.8	9.9	4.4	6.5	12.2
ys-g-e30-rand1	6.8	7.1	1.7	7.0	1.2	0.1	7.4	3.1	5
ys-g-e30-rand2	6.6	6.7	6.9	0.1	7.2	0.5	18.5	8.6	11.7
ys-g-e30-rand3	4.7	4.6	4.3	1.0	0.8	6.2	3.2	8.0	9.4
ys-p-e30-rand1	8.6	5.9	9.5	7.2	8.3	8.2	5.1	0.1	0.6
ys-p-e30-rand2	3.8	9.3	11.8	5.6	14.0	11.8	0.2	5.9	0.7
ys-p-e30-rand3	7.8	13.3	15.2	9.4	9.7	3.3	0.3	0.7	8.4

The failure rates of experiments against the summary files with corresponding excluded sets (e.g. ys-i-e30-rand1 versus I-rand1, or the highlighted diagonal of Table 4.1 are significantly higher than off-diagonal failures when testing experiments against summaries formed from the same compiler. This outcome is consistent with the Extra30 result in Figure 4.1, as the intersection of the sets of simulations that form the I-rand1 summary and the ys-i-e30-rand1 experiment is zero. This is further evidence that 151 member ensembles are variationally deficient. Accordingly, testing the ys-i-e30-rand1 experiment against the I-rand2 summary file results in a "partial self-test" in the

sense that the experimental set is a subset of the simulations that comprise the summary file. Also significant are the off-diagonal blocks of Table 4.1, or inter-compiler tests. For example, the ys-g-e30-rand versus P-rand block: in these experiments there is no evident relationship between failure rates, excluded sets, and compilers, signifying that the inter-compiler tests lose their dependence on the initial conditions.

4.3 Aggregate ensembles

In keeping with the previous results in Figure 4.1 demonstrating uniform, relative lower failure rates of experiments against the combined-compiler summary (Intel-GNU-PGI), we ran similar exhaustive tests of the "rand" experiments against combined-compiler summaries to determine if an ensemble composed of 150 simulations from the three compilers (50 each) would form a suitable accepted ensemble. Three new summaries were created from subsets of the size 151 rand ensembles from section 4.2. Then three random selections of 50 simulations from each ensemble were made such that the corresponding CAM initial temperature perturbations form a disjoint cover (mathematically precise: exact cover) of the 150 (zero perturbation was excluded) perturbations (Table 4.2). The three new summary files are labeled sz150.IGP-rand1, sz150.IGP-rand2, and sz150.IGP-rand3 to designate the randomly excluded set.

Table 4.2: EET failure rates against the designated ensembles. Note the percentages are high in comparison with the desired false positive rate (0.5%), and their instability. The experiment nomenclature is identical to Table 4.1.

Experiment	**sz150.IGP-rand1**	**sz150.IGP-rand2**	**sz150.IGP-rand3**
ys-i-e30-rand1	10.8	3.8	10.3
ys-i-e30-rand2	1.0	13.0	9.3
ys-i-e30-rand3	6.9	4.6	13.3
ys-g-e30-rand1	4.7	8.3	6.0
ys-g-e30-rand2	6.5	16.1	8.0
ys-g-e30-rand3	5.4	4.4	15.3
ys-p-e30-rand1	6.1	10.7	7.4
ys-p-e30-rand2	3.3	6.1	5.3
ys-p-e30-rand3	4.9	11.8	6.6

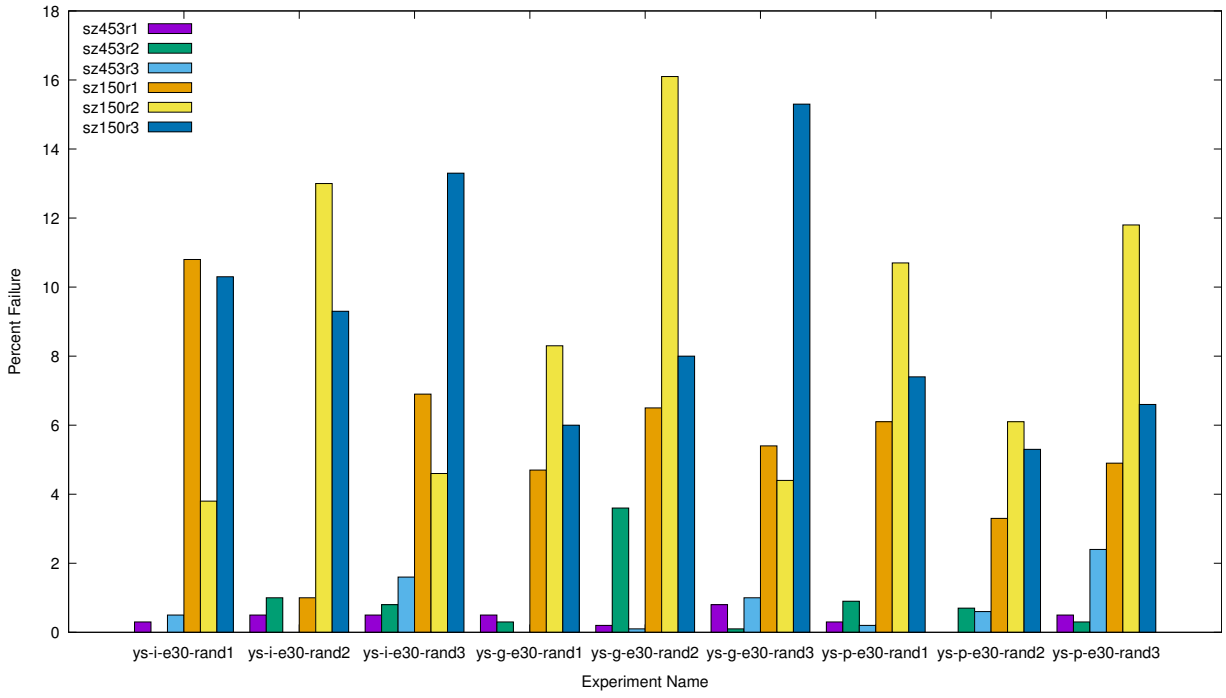
The failure rates' large magnitudes in Table 4.2 as compared with the desired false positive rate (0.5%) suggest that the sample size of 50 from each compiler ensemble is too small. Another salient feature of Table 4.2 is the high variability of the failure rates (e.g., from 0.8 to 15.8 for ys-i-e30-rand2) across the summary files. This is also a characteristic of a low sample size- the underlying distribution is not adequately represented by 50 selections from each compiler ensemble, thus the distribution characteristics depend too heavily on the random selection.

To verify the suitability of the larger aggregate ensemble from Figure 4.1, we performed exhaustive tests of the nine experiments in Table 4.2 against three size 453 summaries constructed by combining the 151 member ensembles (through the rand* exclusion of 30 simulations from section 4.2) from each compiler. The numerical values are given in Table 4.3, and Tables 4.2 and 4.3 are depicted graphically in Figure 4.2. Since the failure rates are now fairly consistent across all three summaries in Table 4.3 and the rates are approximately equal to our desired 0.5% false positive rate, this suggests that these 453 member summaries are providing adequate CESM-ECT failure rate determination. However, this assessment is examined in greater detail in Section 5 where we consider code modifications, and in Section 6 where an intermediate ensemble of size 300 is created and tested.

Table 4.3: EET failure rates against the designated ensembles. Note the percentages are approximately equivalent to the desired false positive rate (0.5%), and their stability relative to Table 4.2.

Experiment	**sz453.IGP-rand1**	**sz453.IGP-rand2**	**sz453.IGP-rand3**
ys-i-e30-rand1	0.3	0.0	0.5
ys-i-e30-rand2	0.5	1.0	0.0
ys-i-e30-rand3	0.5	0.8	1.6
ys-g-e30-rand1	0.5	0.3	0.0
ys-g-e30-rand2	0.2	3.6	0.1
ys-g-e30-rand3	0.8	0.1	1.0
ys-p-e30-rand1	0.3	0.9	0.2
ys-p-e30-rand2	0.0	0.7	0.6
ys-p-e30-rand3	0.5	0.3	2.4

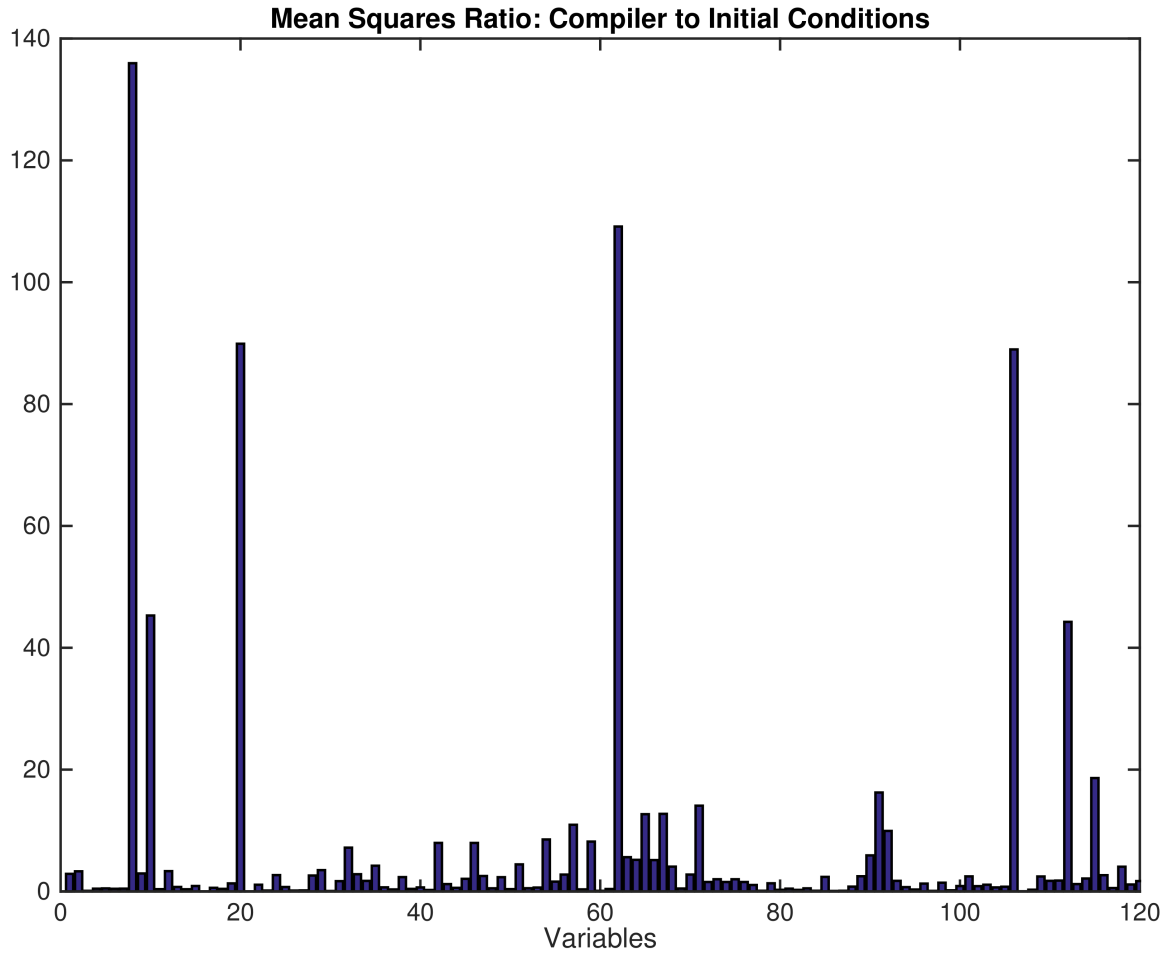
Figure 4.2: EET failure rates versus designated experiment. Experiment names are abbreviated, e.g. yellowstone-intel-extra30-rand1 becomes ys-i-e30-r1 as in Table 4.1 and 4.2. Note that the failure rates against the 453 member ensemble were comparable to our desired false positive rate (0.5%).



4.4 Effect on variability of compiler and CAM temperature initial temperature conditions

To probe the relationship between the effect of compiler and CAM temperature initial conditions on variability, Dorit Hammerling performed an ANOVA study of the Mean Squares ratio (sum of mean squares divided by degrees of freedom) of compiler to CAM temperature initial conditions. The plot (Figure 4.3) indicates that the compiler effect is generally more influential on variability than the temperature initial conditions due to the mean ratio of 6.6 and median ratio of 1.3 (compiler to CAM temperature initial conditions), however, note that the ratio varies greatly by variable.

Figure 4.3: **Courtesy Dorit Hammerling.** The ANOVA Mean Squares Ratio is determined by the ratio of the sum of the mean squares of the compiler divided by the compiler degrees of freedom (two) divided by the sum of the mean squares of the initial conditions divided by the initial condition degrees of freedom (180). The mean ratio is 6.6, while the median is 1.3. If the effects were equal, these two values would be 1.



Chapter 5

Code modification experiments

Our goal for the modification experiments was to perform a "minimal" change to CAM code that, when propagated through the year-long simulation, would result in a difference in single precision output in the CAM history file. In this section, we describe preliminary attempts at code modification and the lessons learned which were applied to later experiments. We then describe three categories of code modifications selected for exhaustive testing in Section 5.1.2, 5.1.3, and 5.1.4.

5.1 Test cases

This subsection contains preliminary modifications, those representing different coding styles, optimization strategies, and subroutines and performance. In the code blocks provided in this subsection green text signifies code added, red deleted, and black unchanged.

5.1.1 Preliminary modifications

The first attempt was a plausible change a software engineer would make for optimization: exchanging a division with a multiplication by the inverse. We modified the CAM micro_mg1_0.F90 (Morrison-Gettelman microphysics) module with the following changes, referred to as MG1:

```
gamma(pgam(k)+4._r8)/ &
gamma(pgam(k)+3._r8)/lamc(k)/2._r8*1.e6_r8
0.5e-6_r8*gamma(pgam(k)+4._r8)/ &
```

```
gamma(pgam(k)+3._r8)/lamc(k)
```

This is not a power-of-two change, so it was expected to result in a rounding error, which propagated over many calls to micro_mg1_0, would result in a measurable difference in CAM output. After running a simulation with this code change for 12 months, there were no measurable differences in either the double-precision CAM log files, or in the single-precision yearly outputs from our unmodified control case. To investigate this unexpected outcome, we wrote two short programs in FORTRAN that perform exactly the same steps using the same constants as the MG1 original and modified lines. The programs populate variables `pgam` and `lamc` with random values, calling the gamma function identically as in the example. It is important to replicate the MG1 code as faithfully as possible to account for the possibility that the compiler is sensitive to the "context" of the code, such as the lines being encapsulated in a loop, and variables inside function calls. We compiled the two programs with the identical compiler version and flags as CESM used to build the module originally. The assembly code of the original was carefully compared against the modified version, and the important instructions corresponding to the MG1 modification are detailed in Table 5.1.

Table 5.1: Assembly code instruction counts for original and modified MG1 module simulation code.

Instruction	Original count	Modified count
vmulsd	2	1
vdivsd	2	2
vmovsd	7	6
mov	10	10

That each version of the code contains 2 divisions indicates that the compiler has replaced `lamc(k)/2._r8*1.e6_r8` with a multiplication by the inverse of `2._r8*1.e6_r8`. The additional vmulsd and vmovsd likely are a result of the original code needing one additional step to evaluate the multiplication of the two constants. The compiler has rearranged the operations in such a way that it is numerically identical to the modification.

Next we experimented with a subroutine (preq_hydrostatic) in prim_si_mod.F90 and made three different modifications. The first modification was the following:

```
hkk = dp(i,j,nlev)*0.5d0/p(i,j,nlev)
hkl = 2*hkk
hkk = dp(i,j,nlev)/p(i,j,nlev)
hkl = hkk
hkk = hkk/2
```

This alteration also did not result in a measurable double or single-precision floating-point difference in CAM logs, or the output file, respectively. To determine if the lack of an observable difference in the output was due to a low frequency of modified code execution or a lack of rounding error (change of a multiplication to division by 2), we multiplied and divided the `hkk` variable by 3. The double-precision logs differed, confirming the propagation of the change and suggesting an absence of rounding error. Next, to test the effect of call frequency, we performed a modification to a subroutine in derivative_mod.F90, known to be executed often:

```
gv(i,j,1)=elem%metdet(i,j)*(elem%Dinv(1,1,i,j)*v(i,j,1) +
   elem%Dinv(1,2,i,j)*v(i,j,2))
gv(i,j,2)=elem%metdet(i,j)*(elem%Dinv(2,1,i,j)*v(i,j,1) +
   elem%Dinv(2,2,i,j)*v(i,j,2))
gv(i,j,1)=elem%metdet(i,j)*elem%Dinv(1,1,i,j)*v(i,j,1)
gv(i,j,1)=gv(i,j,1) + elem%metdet(i,j)*elem%Dinv(1,2,i,j)*v(i,j,2)
gv(i,j,2)=elem%metdet(i,j)*elem%Dinv(2,1,i,j)*v(i,j,1)
gv(i,j,2)=gv(i,j,2) + elem%metdet(i,j)*elem%Dinv(2,2,i,j)*v(i,j,2)
```

This indeed produced a measurable difference in the double-precision logs, but the high frequency of the execution of the change meant that this example was not the “minimum” code change that we desired from the outset. These examples demonstrate that code changes small enough for a compiler to perform and sensitivity to call frequency are two primary constraints for

future modification experiments.

5.1.2 Modifications representing different coding styles

In this subsection, we describe code modifications that are mathematically equivalent formulations that could arise from two software engineers solving the same problem in different ways.

5.1.2.1 DYN3

The first code change in CAM that matched our criteria for low frequency of execution and minimal alteration, and brought about an observable single-precision difference from the control was to the preq_omega_ps subroutine of prim_si_mod.F90 (referred to as DYN3):

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

This is effectively a single line change. Note that we verified that the difference in single and double precision output was not due to a "catastrophic cancellation" of `vgrad_p(i,j,1)` and `0.5d0*divdp(i,j,1)`.

5.1.2.2 DYN3b

Another characteristic of DYN3 is that it constitutes a reduction in the number of lines of code via the elimination of temporary variables. To test the effect of increasing the number of lines by the addition of a temporary variable, we created the DYN3b experiment:

```
ckk = 0.5d0/p(i,j,1)
term = divdp(i,j,1)
tmpvar = 1.0d0/p(i,j,1)
```

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

5.1.2.3 DYN4

Another successful modification was to the preq_hydrostatic subroutine of prim_si_mod.F90 (denoted DYN4 hereafter). Three successive and similar changes centered on the creation of a new variable `tt_real` (e.g. the first):

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

5.1.2.4 DYN4a

Similarly to DYN3b, DYN4a seeks to test the effect of the removal of temporary variables and hence lines of code. The experiment denoted DYN4a removes the changes to the `phii` array, alters the definition of `tt_real` to be consistent across all three "do loops" in preq_hydrostatic, but the changes to `phi` persist, e.g.:

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

5.1.2.5 DYN4a3

This experiment takes the reduction of scope of modification of the base DYN4 experiment to its logical conclusion: isolating the temporary variable to the third (thus the "3" suffix) "do loop" in preq_hydrostatic.

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

5.1.3 Modifications representing optimization strategies

The code changes in this subsection were targeted at improving the performance of existing code via optimization of mathematical expression, for example reducing the number of expensive division operations.

5.1.3.1 ADV1

ADV1 designates the modification to the euler_step subroutine of prim_advection_mod.F90. The original version of this subroutine includes an operation that divides by a spherical mass matrix `spheremp`. The modification to this kernel consisted of declaring a temporary variable (`tmpsphere`) defined as the inverse of `spheremp`. Multiplication is substituted for division to complete the modification. Note that creation of this temporary variable removes an assignment operation from an inner loop, speeding up the qtens_biharmonic evaluation.

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

5.1.3.2 EDG1

EDG1 is a permutation of the evaluation of an MPI receive buffer unpacking subroutine, the edges of which are boundaries between Galerkin domains. Changing the order of buffer unpacking has implications for performance, as traversing the buffer sub-optimally can prevent cache prefetching. However, it also results in non-BFB results:

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

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

5.1.4 Modifications representing performance enhancement

This set of code modification experiments is characterized by more substantial, performance oriented changes that are small enough not to be considered climate-changing. The modifications are limited to substitution or alteration of a single subroutine.

5.1.4.1 RAND

The default (pseudo) random number generator used by CESM is KISSVEC. However, should a scientist want to use a different random number generator with superior performance, it is necessary to determine the response of CESM-ECT. In this experiment we substituted CESM's alternate generator, Mersenne Twister (MT), whose properties are well-established and can even be made adequate for cryptography. In the initial trial, we simply changed a parameter to select MT rather than KISSVEC, e.g. in mcica_subcol_gen_lw.f90:

```
! Set random number generator to use (0 = kissvec; 1 = mersennetwister)
irnd = 0
irnd = 1
```

This resulted in a resounding failure in CESM-ECT: not a single test passed. Interpreting this as an indication of a bug, we inspected the code in more detail. Indeed, the seed passed to MT was uninitialized, which exhibits undefined behavior. On Yellowstone with the default Intel compiler, it is likely that the same value (zero) is passed as a seed. It appears that MT responded by returning a single random number which substantially affected the climate's properties. This

issue prompted the utilization of the same method of seeding MT as that used for KISSVEC- in short, seeding with a vector rather than a single integer. The code to pass the seed vector to MT was modified as follows:

```
  seed1(i) = (pmid(i,nlay) - int(pmid(i,nlay))) * 1000000000
  seed2(i) = (pmid(i,nlay-1) - int(pmid(i,nlay-1))) * 1000000000
  seed3(i) = (pmid(i,nlay-2) - int(pmid(i,nlay-2))) * 1000000000
  seed4(i) = (pmid(i,nlay-3) - int(pmid(i,nlay-3))) * 1000000000
enddo
iseed = seed1 + seed2 + seed3 + seed4
randomNumbers = new_RandomNumberSequence(seed = changeSeed)
randomNumbers = new_RandomNumberSequence(seed = iseed)
```

Note that this change was expected to be minimal not in the number of lines of code modified, but that interchanging two widely-used pseudo random number generators which exhibit desirable properties (such as long periods) is not expected to result in a different climate.

5.1.4.2 PREC

PREC is another performance-oriented modification to the wv_sat_methods.F90 module 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 coprocessor acceleration due to superior single-precision computation speed. To accompish this, we used TotalView to trace which elemental function was called to compute saturation vapor pressure. After determining that the GoffGratch function was called, we created a similar elemental function to compute the output in 4-byte floating-point format rather than typical double-precision:

```
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
```

5.1.5 Results

The code modification experiments shown in Figure 5.1 against the size 453 and 150 combined compiler summaries as well as the default CESM-ECT compiler (labeled Orig-Intel) are illuminating as well. DYN3, DYN4, and PREC behave much like different compilers, exhibiting very low failure percentages when tested against the sz453 compiler summaries. While increasing the number of lines of code for the DYN3 experiment (DYN3b) translates to an increase in EET failure rate, combining lines for DYN4 (DYN4a, DYN4a3) has little effect. ADV1, EDG1, and RAND fit neatly between DYN3 and DYN3a, conforming more closely to the behavior of DYN3 (Figure 5.1). The failure rates of the code modification experiments against the sz150 compiler aggregated summaries are similar to those of the random selection experiments (Table 4.2). Another salient feature of this figure is the experiments' EET failure rates against Intel CESM-ECT ensemble used in BH2015 (Orig-Intel) in comparison with the sz453 compiler-aggregate ensembles.

Another potential source of difference between the behaviors of the code modifications is function call count. It is possible that code blocks with high call counts during CESM runtime would propagate their perturbations more extensively to the CESM execution environment. A study of the relative call count numbers is presented in Table 5.2.

Table 5.2 taken in conjunction with Figure 5.1 indicates that the mechanism by which code modification perturbations propagate throughout the execution of CESM is more complex than

Figure 5.1: Note that the size 453 failure rates are on the order of our desired false positive rate. The CESM-ECT ensemble from BH2015 is featured (Orig-Intel) for comparison with the compiler aggregate rand summaries.

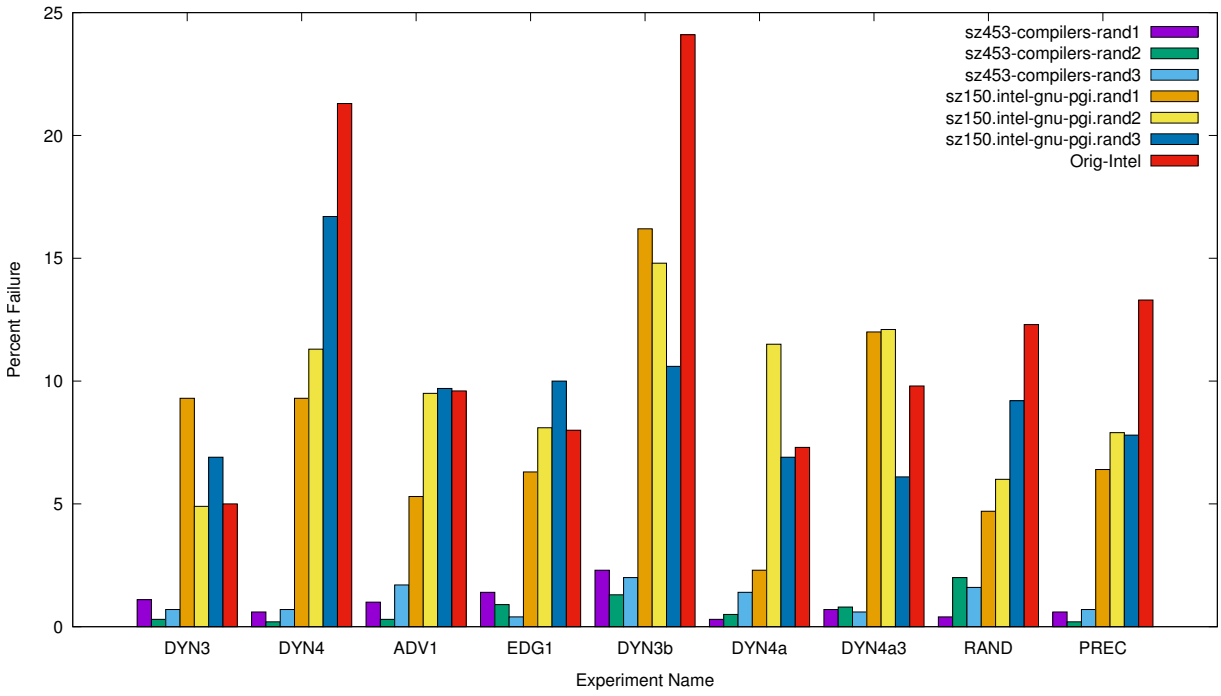


Table 5.2: **Courtesy Allison Baker.** Call frequencies relative to ADV1.

Experiment name	**Function**	**CAM module**	**Relative freq.**
DYN3	preq_omega_ps	prim_si_mod	5
DYN4	preq_hydrostatic	prim_si_mod	5
ADV1	euler_step	prim_advection_mod	1
EDG1	edgeVunpack	edge_mod	45.5

simply the number of times a modified function is called. It is quite conceivable that while a modified section of code is only executed once, its perturbation is carried throughout the execution environment by objects which depend on it. Establishing a causal, weighted relationship between code line content changes and failure percentages is of interest. In fact, an object dependency hierarchy may assist in measuring modification propagation. This is of interest as an area for future study.

Chapter 6

Does an ensemble need 453 members?

In an effort to optimize the number of constituent members of an accepted ensemble, ensembles of 300 members were created in the same way as the size 150 and 453 sets in Section 4.2, but using three new random selections. We ran exhaustive tests of all experiments against the sz300 ensembles; the results of the modification experiments from Section 5 together with the sz300 results are shown below in Table 6.1. The size 300 ensembles exhibit failure rates closer to the size 453 ensembles than to those of the 150 member ensembles.

Table 6.1: EET failure rates of code modification experiments versus three different sizes of compiler aggregate CESM-ECT ensembles. Means are shown per ensemble between experiments, and for each experiment per ensemble.

Experiment	sz150r1	sz150r2	sz150r3	mean	sz300r4	sz300r5	sz300r6	mean	sz453r1	sz453r2	sz453r3	mean
DYN3	9.3	4.9	6.9	7.0	3.1	1.3	1.4	1.9	1.1	0.3	0.7	0.7
DYN4	9.3	11.3	16.7	12.4	1.0	0.8	1.6	1.1	0.6	0.2	0.7	0.5
ADV1	5.3	9.5	9.7	8.2	1.5	0.4	1.2	1.0	1.0	0.3	1.7	1.0
EDG1	6.3	8.1	10.0	8.1	2.5	2.3	0.7	1.8	1.4	0.9	0.4	0.9
DYN3b	16.2	14.8	10.6	13.9	2.6	1.1	2.1	1.9	2.3	1.3	2.0	1.9
DYN4a	2.3	11.5	6.9	6.9	2.4	3.4	0.1	2.0	0.3	0.5	1.4	0.7
DYN4a3	12.0	12.1	6.1	10.1	1.8	1.3	3.8	2.3	0.7	0.8	0.6	0.7
RAND	4.7	6.0	9.2	6.6	0.4	1.1	1.1	0.9	0.4	2.0	1.6	1.3
PREC	6.4	7.9	7.8	7.4	1.0	0.8	1.6	1.1	0.6	0.2	0.7	0.5
Mean	8.0	9.6	9.3		1.8	1.4	1.5		0.9	0.7	1.1	

Now we consider the rand experiments again. In Table 10, results from the size 300 ensembles are inserted between those of the 150 and 453 ensembles (from Tables 5 and 6, respectively). The comparative advantage of the size 453 is less distinct here than in Table 6.1, and it appears less stable (more variation in experiment failures within the same compiler aggregate summary size). For example, the EET failure rates of ys-i-e30-r2 experiment tested against the sz453 compiler aggregate summaries: 0.2, 1.8, 0.4, or also ys-g-e30-r2 against sz453: 0.3, 3.3, and 0.8. We expect

this to be an artifact of the small sample size (three ensembles), but this will be investigated in an upcoming formal study of the stability properties of these ensembles. An ensemble of arbitrarily large size is untenable from the standpoint of computational cost, as the CESM-ECT accepted ensembles are recreated for climate-changing CESM code releases.

Table 6.2: EET failure rates of code modification experiments versus three different sizes of compiler aggregate CESM-ECT ensembles. Means are shown per ensemble between experiments, and for each experiment per ensemble. In the interest of space savings the experiment nomenclature differs slightly from previous tables: rand1, rand2, and rand3 selections are abbreviated r1, r2, and r3.

Experiment	**sz150r1**	**sz150r2**	**sz150r3**	**mean**	**sz300r4**	**sz300r5**	**sz300r6**	**mean**	**sz453r1**	**sz453r2**	**sz453r3**	**mean**
ys-i-e30-r1	10.8	3.8	10.3	8.3	2.0	0.3	0.5	0.9	0.3	0.0	0.5	0.3
ys-i-e30-r2	1.0	13.0	9.3	7.8	0.4	0.3	0.0	0.2	0.5	1.0	0.0	0.5
ys-i-e30-r3	6.9	4.6	13.3	8.3	0.3	0.8	0.9	0.7	0.5	0.8	1.6	1.0
ys-g-e30-r1	4.7	8.3	6.0	6.3	0.2	0.7	0.4	0.4	0.5	0.3	0.0	0.3
ys-g-e30-r2	6.5	16.1	8.0	10.2	1.3	0.3	0.4	0.7	0.2	3.6	0.1	1.3
ys-g-e30-r3	5.4	4.4	15.3	8.4	0.4	0.3	0.9	0.5	0.8	0.1	1.0	0.6
ys-p-e30-r1	6.1	10.7	7.4	8.1	0.1	0.4	0.9	0.5	0.3	0.9	0.2	0.5
ys-p-e30-r2	3.3	6.1	5.3	4.9	1.6	1.4	0.3	1.1	0.0	0.7	0.6	0.4
ys-p-e30-r3	4.9	11.8	6.6	7.8	0.2	0.7	1.2	0.7	0.5	0.3	2.4	1.1
Mean	5.5	8.8	9.1		0.7	0.6	0.6		0.4	0.9	0.7	

Chapter 7

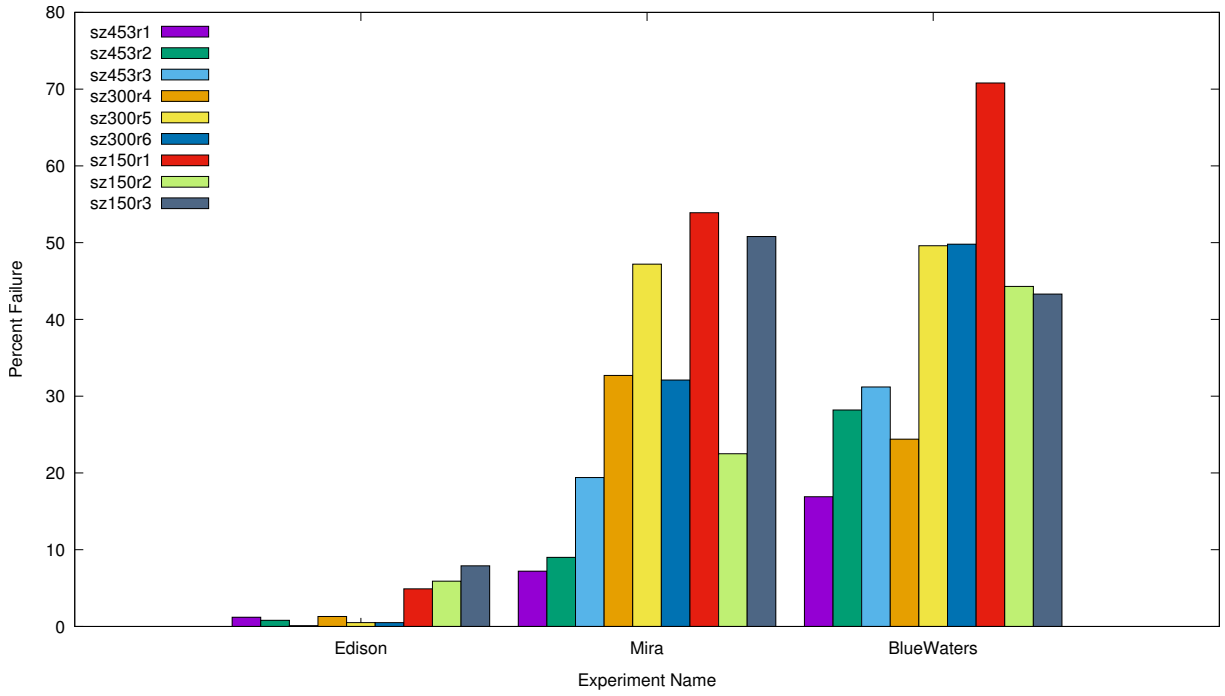
Mira and Blue Waters

The results from CESM-supported machine testing in BH2015 and shown in Table 3 indicate that the Mira and Blue Waters experiments are only slightly above the failure threshold of $N_{pcFails}$. In this section we perform a more detailed analysis of these results with the added insight provided by EET and knowledge of ensemble composition. First, we re-examine the Mira and Blue Waters results in the context of the new compiler-aggregate ensembles with CESM-ECT (Figure 7.1). Note that we include the NERSC Edison results to demonstrate that most CESM-supported machines exhibit a clear, low failure rate. With the new size 453 aggregate ensembles, the failure rates are still quite high for Mira and Blue Waters, which raises the question of whether the ensemble distribution is still too narrow, or whether the failures are true indications of an error in the supercomputers' software or hardware. In particular, because an upcoming CESM experiment was scheduled on Mira, an investigation into the validity of its high failure rate was of utmost importance.

7.1 Excluding CAM variables

CESM-ECT is a coarse-grained testing method, and pyCECT simply returns sets of failing Principal Components. To relate failing PCs to sections of code and perhaps hardware, we needed to understand which CAM variables were problematic. A systematic elimination of variables was undertaken, which involved modifying pyCECT to accept a list of CAM variables to be excluded from the PC calculation. This was done to omit variables one at a time, recording the CESM-ECT failure rate per excluded variable. Since pyCECT was written to read the PC loadings from a

Figure 7.1: Exhaustive testing failure rates of three CESM-supported supercomputers. Mira and Blue Waters were borderline failures using CESM-ECT (see Table 2.3).

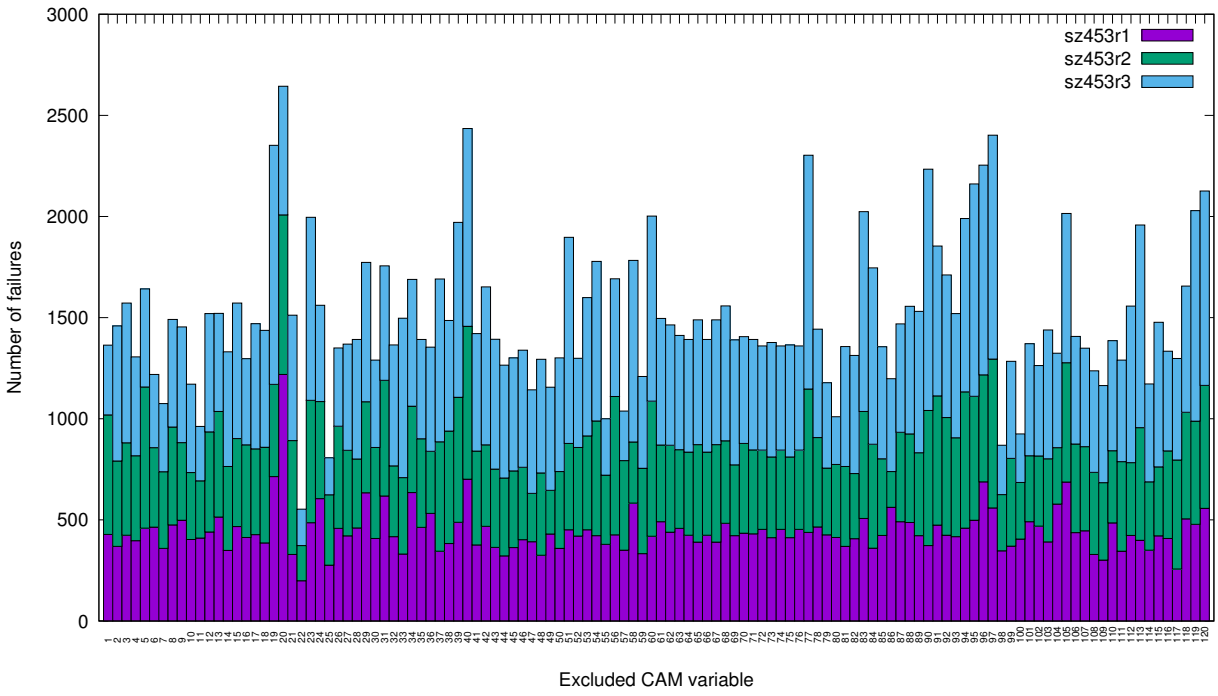


summary file, we altered the tool to compute new loadings from the summary file's global means, with the designated CAM variable excluded from the calculation. The new experiment's CAM output files were then rotated into the summary's PC space as usual. The results for the Mira experiment are reported as a stacked histogram in Figure 7.2. The y-axis indicates the number of EET failures (out of $\binom{24}{3} = 2024$ tests for each sz453 ensemble, or 6072 total), and the x-axis records the index of the CAM variable omitted.

The minima of this plot correspond to CAM variables whose elimination results in a decreased failure rate- suggestive of a causal relationship with Mira's failure rates. Based in large part on the data in Figure 7.2, Dorit Hammerling determined that these six variables merited further investigation: AREL, AWNC, CLDLIQ, FICE, ICWMR, and FSNTOA (numbers 6, 7, 11, 16, 22, and 74). We repeated pyCECT testing on the Mira experiment with these six variables removed, and observed nearly five times lower failure rates. With climate scientists' input, we found that four of the above six variables (AREL, AWNC, FICE, and ICWMR) are featured prominently in the Morrison-Gettelman microphysics kernel (MG1).

Once the physics kernel had been targeted for analysis as the primary source of the failures, Youngsung Kim's open-source KGEN tool (See Section 10: Software) was used to extract the MG1 kernel from CAM and build it as a stand-alone executable that could be run on a single core in less than a second. This facilitated the rapid testing of MG1 on several machines passing CESM-ECT and comparison with Mira. A subset of MG1 variables with larger normalized Root Mean Square (RMS) errors was found on Mira, and these variables' values were output and juxtaposed with those executed on Yellowstone. Given the code lines that compute these variables, software engineers hypothesized that Fused Multiply-Accumulate (FMA) instructions may have caused the RMS error values, and the instructions were disabled via compiler switch. A repeat of the KGEN RMS error testing confirmed that the values were then consistent with those produced on Yellowstone. To definitively prove that FMA caused the high CESM-ECT failure rates on Mira, CESM was rebuilt with the FMA instructions disabled (-qfloat=nomaf), which yielded a remarkable 0.7% failure rate against one sz453 summary file. Beginning with a coarse-grained test like CESM-ECT and ending

Figure 7.2: Stacked histogram of Mira failures against the three sz453 summary files, with the excluded CAM variable indicated on the x-axis. Note that the number of failures is out of 6072 possible failures.



with the identification (via a fine-grained test like KGEN) of specific lines of code and corresponding machine instructions is a resounding success.

While Mira CPUs are Power architecture (PowerPC A2), examining the effect of FMA on the MG1 kernel running on Intel Haswell CPUs is of considerable interest, as these CPUs are becoming commonplace. We built and executed the KGEN extracted MG1 kernel on a Haswell Xeon E7-8890, which failed the RMS error test with FMA enabled, and passed when disabled. We confirmed that the failing variables were the same as those failing KGEN tests on Mira with FMA enabled. We intend to continue working with Haswell CPUs to determine if MG1 can be refactored to be less sensitive to FMA, and identify other sections of code with similar sensitivities.

Chapter 8

Conclusions

In BH2015, a consistency test for CESM was developed to calculate whether a new "configuration (e.g., resulting from a code modification, compiler change, or new hardware platform) is consistent with the original 'accepted' (or control) configuration." That paper sought to find an original configuration that would "capture the natural variability in the modeled climate system." In this thesis, we extend the findings of BH2015 by further specifying the characteristics of an accepted CESM-ECT ensemble. To do so, we adopted the strategy of verifying the components of an accepted ensemble to ensure that it would contain variability sufficient to represent a consistent climate. Through a tool we developed to perform every possible combination of CESM-ECT tests (EET), we were able to gauge whether new experiments exhibited failure rates on the order of CESM-ECT's desired false positive rate (0.5%). With this tool, the base constituents of the ensembles used in BH2015 were examined for sufficient variability.

Since it was found that no single compiler-based ensemble indeed possessed enough variability, new CESM-ECT ensembles were created by aggregating Yellowstone ensembles of simulations differing in the compiler used to build CESM. The compiler aggregate ensembles were tested by devising experiments that introduced minimal code modifications to CAM modules. Since these modifications were not expected to be climate-changing, their EET failure rates effectively gauged whether the ensemble contained enough variability to represent a consistent climate. The size 453 compiler aggregate ensembles consistently resulted in false positive order EET failure rates when testing code modification experiments.

Examining two machines manifesting borderline CESM-ECT failures (Blue Waters and Mira) under EET and the size 453 ensembles resulted in high failure rates in comparison with other CESM-supported machines. A study of the CAM variables contributing to the high Mira failure rates led to the identification of Fused Multiply-Add instructions as causative agents. This successful process involved the coupling of coarse-grained testing facilitated by CESM-ECT with fine-grained testing provided by KGEN, together with the expertise of scientists and software engineers.

Chapter 9

Future directions

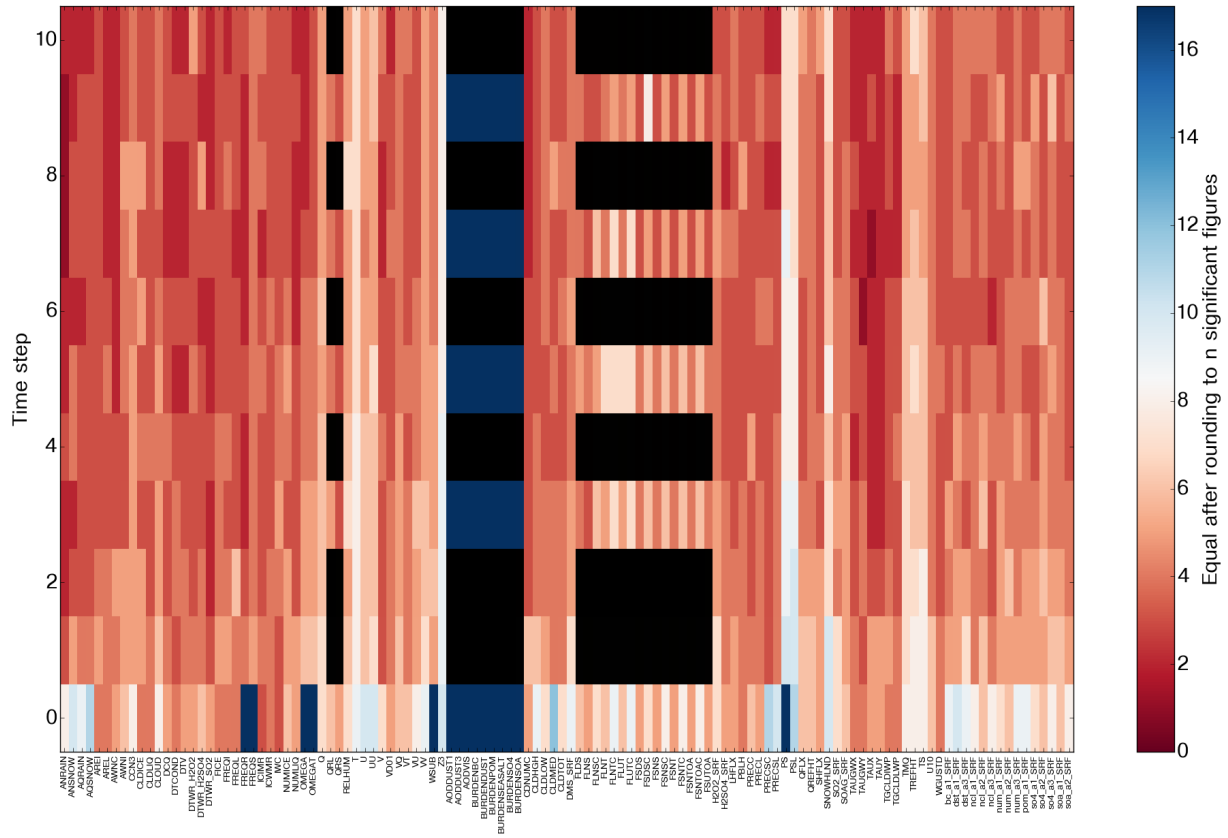
Possible future directions of this research include deeper analysis of the relationship between test failures, failing PCs, and CAM variable space. Such a study could provide information necessary to improve the composition of the accepted ensemble, with the goal of more accurately determining runs that should pass or fail. This may require the application of a machine learning strategy.

Another promising (and possibly related) direction of this research is to explore the propagation of perturbations within the execution of CESM. Figure 9.1 represents our preliminary investigation on this front. To generate these data we ran two CESM simulations: one with an initial CAM temperature perturbation of 1×10^{-14}, and another unperturbed. The simulations ran for 11 time steps including $t = 0$, and saved CAM variable double-precision floating point values to disk for each step. With this output data we compared the number of identical significant figures between the perturbed and unperturbed simulations for the global means of each CAM variable (x-axis of plot) at each time step (y-axis of plot). This was accomplished by equality testing after rounding to n significant figures (displayed in the color bar). The plot does not account for the magnitude of the difference beyond 0 significant figures. Black designates time steps where the corresponding variable was not output. These variables are sub-cycled, meaning that their values are not computed at each time step in the figure. This can occur if the variable depends on sub-cycled variables, or if the function or subroutine is not called. Figure 9.1 illustrates that most CAM variables are calculated for the zeroth time step, and are at least indirectly dependent on the initial

temperature perturbation.

There is high information density in Figure 9.1, and it points us in several interesting directions. An examination of the perturbation propagation will need to account for differing orders of magnitude, and the multiple timescales of the model. It may be informative to analyze the propagation across the spatial field rather than global means, which will require more sophisticated techniques to detect patterns in such high dimensional data.

Figure 9.1: Representation of effects of initial CAM temperature perturbation after 11 time steps (including $t = 0$). CAM variables are listed on the x-axis, and the y-axis records the simulation time step. The color bar designates equality of the corresponding variables between the unperturbed and perturbed simulations after being rounded to n significant digits (n is the color). Time steps where the corresponding variable was not computed are colored black.



Chapter 10

Software

Data sets and modifications to CAM modules can be found at https://github.com/milroy/cesm_ensemble_mods. KGEN is available at https://proxy.subversion.ucar.edu/pubasap/kgen/trunk. The Python tools described in Section 2 can be downloaded from the collection of parallel Python tools available on the NCAR's Application Scalability and Performance website (https://www2.cisl.ucar.edu/tdd/asap/application-scalability) or obtained directly from NCAR's public Subversion repository (https://proxy.subversion.ucar.edu/pubasap/pyCECT/tags/1.0.0/). CESM simulation data is available upon request.

Bibliography

[1] A. H. Baker, D. M. Hammerling, M. N. Levy, H. Xu, J. M. Dennis, B. Eaton, J. Edwards, C. Hannay, S. Mickelson, R. Neale, D. Nychka, J. Shollenberger, J. Tribbia, M. Vertenstein, and D. Williamson. A new ensemble-based consistency test for the community earth system model. Geosci. Model Dev., 8:3823–3859, 2015.

[2] T. Clune and R. Rood. Software testing and verification in climate model development. IEEE Software, 28:49–55, 2011.

[3] S. M. Easterbrook and T. C. Johns. Engineering the software for understanding climate change. Comput. Sci. Eng., 11:65–74, 2009.

[4] J. Pipitone and S. Easterbrook. Assessing climate model software quality: a defect density analysis of three models. Geosci. Model Dev., 5:1009–1022, 2012.