Towards characterizing the variability of statistically consistent Community Earth System Model simulations

Daniel J. Milroy¹, Allison H. Baker², Dorit M. Hammerling², John M. Dennis², Sheri A. Mickelson², and Elizabeth R. Jessup¹

¹ University of Colorado, Boulder, Colorado, USA
daniel.milroy@colorado.edu, jessup@cs.colorado.edu
² The National Center for Atmospheric Research, Boulder, Colorado, USA
abaker@ucar.edu, dorith@ucar.edu, dennis@ucar.edu, mickelso@ucar.edu

Abstract

Large, complex codes such as earth system models are in a constant state of development, requiring frequent software quality assurance. The recently developed Community Earth System Model (CESM) Ensemble Consistency Test (CESM-ECT) provides an objective measure of statistical consistency for new CESM simulation runs, which has greatly facilitated error detection and rapid feedback for model users and developers. CESM-ECT determines consistency based on an ensemble of simulations that represent the same earth system model. Its statistical distribution embodies the natural variability of the model. Clearly the composition of the employed ensemble is critical to CESM-ECT’s effectiveness. In this work we examine whether the composition of the CESM-ECT ensemble is adequate for characterizing the variability of a consistent climate. To this end, we introduce minimal code changes into CESM that should pass the CESM-ECT, and we evaluate the composition of the CESM-ECT ensemble in this context. We suggest an improved ensemble composition that better captures the accepted variability induced by code changes, compiler changes, and optimizations, thus more precisely facilitating the detection of errors in the CESM hardware or software stack as well as enabling more in-depth code optimization and the adoption of new technologies.

Keywords: Community Earth System Model, CESM Ensemble Consistency Test, statistical consistency, code modification as source of variability, compiler as source of variability, Community Atmosphere Model, non-bit-for-bit, Fused Multiply-Add

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., [3], [4]). 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., [9]). Therefore, while achieving bit-for-bit (BFB) identical results may be desirable in general (e.g., [10]) 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) [6], 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) [2]. 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 [2], ensemble variability is created by roundoff-level perturbations to the initial temperature field. However, as the goal in [2] is the introduction of the ensemble consistency testing methodology, the important question of the ensemble composition is not addressed.

Our goal 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 [5], where global summation orders are modified with noticeable impact on climate simulation results. However, the aim in [5] is to improve or even achieve reproducibility (via alternative algorithms or increased precision). In this study, we improve upon the work in [2] 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 CESM-ECT that improves the tool’s accuracy and broadens its applicability.

This paper is organized as follows. In Sect. 2, we provide background information on CESM-ECT and describe our experimental setup and tools. In Sect. 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. 4, and an alternative composition is proposed in Sect. 5. We demonstrate the utility of the new ensemble composition in Sect. 6.
2 Preliminaries

The CESM is composed of multiple geophysical models (e.g. atmosphere, ocean, land, etc.), and the CESM-ECT work in [2] focuses on data from the Community Atmosphere Model (CAM) component 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 CESM-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 work on a 1° global grid using the CAM5 model version described in [7]. We run simulations 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.

The CESM-ECT ensemble is a set of CESM simulations with identical parameters, differing only in initial atmospheric temperature conditions [2]. The ensemble members are run, evolving in time at the same rates for the same amount of time. The initial temperature perturbations guarantee the creation of unique trajectories through the global models’ phase space, in turn generating an ensemble of output variables that represents the natural variability in the model. Both in [2] and our work, perturbations of the initial atmospheric temperature field are generated in the closed interval $[-8.5 \times 10^{-14}, 8.5 \times 10^{-14}]$, and are run for 12 simulation months. One year is chosen as it is both short enough to permit ensemble generation in a reasonable amount of time and sufficient to generate a representative statistical distribution. The $\mathcal{O}(10^{-14})$ perturbation is selected because it permits a divergence of the phase space trajectories, but is expected to preserve the variables’ distributions [2]. A minimum ensemble size of 151 members is chosen due to the lower-bound constraint imposed by Principal Component Analysis, namely that the number of data points (i.e. ensemble members) must be larger than the number of variables.

In CESM-ECT, PCA is applied to the global area-weighted means of the ensemble [2]. First, the $N_{var} \times N_{ens}$ (number of variables, number of ensemble members) 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” $P_{gm}$. Finally, the matrix $V_{gm}$ is projected into the subspace. The standard deviation of the ensemble scores is calculated and denoted $\sigma_{S_{gm}}$.

To check for consistency with the ensemble, a small set of new runs ($N_{test} = 3$ by default) is fed to the Python CESM Ensemble Consistency Tool (pyCECT), which issues a pass or fail based on the number of PC scores that fall outside a specified confidence interval (typically 95%) [2]. The tool calculates the area-weighted global means for each variable in all the new runs. These means are standardized using the mean and standard deviations of the ensemble. Next, the standardized means are rotated into the PC space of the ensemble via the transformation matrix $P_{gm}$. Third, the tool determines if the first 50 principal components of the new runs are within two 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 any PC score outside two standard deviations as a failure [2]. Let $P = \{A, B, C\}$ be the set of
sets of failing PCs. CESM-ECT performs the following set operations to determine an overall failure: $S_{AB} = A \cap B, S_{AC} = A \cap C, S_{BC} = B \cap C; S = S_{AB} \cup S_{AC} \cup S_{BC}$; the test returns a failure if $|S| \geq 3$. Parameters specifying the pass/fail criteria can be tuned to achieve a desired false positive rate, and we use the default parameters to yield a rate of 0.5%. Note that the false positive rate is the frequency at which the test returns a failure when the set of new runs truly represents a pass. We use this rate as guidance for evaluating the ensemble composition. An improperly chosen ensemble will produce misleading CESM-ECT results.

To thoroughly assess the appropriateness of the ensemble composition, we perform more exhaustive testing than was done in [2], 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 CESM-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 $\binom{N_{tot}}{N_{test}}$ 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.

3 Code Modifications

In this section we define the “minimal” code changes that should produce the same climate when evaluated by CESM-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 [1]) 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).

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)
```

**Modified:**

```
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)

3.2 Modifications representing optimization strategies

The code changes in this subsection target improving the performance of existing code by 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
    qtenbiharmonic(:,:,k,q,ie) = &
    -rhs_viss*dt*nu_q*dp0*Qtenbiharmonic(:,:,k,q,ie) / elem(ie)%spheremp
      (:,:)  

Modified:

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

Consistent CESM simulations Milroy, Baker, Hammerling, et al.

1593
**Unpack-order (UO)** changes the order that an MPI receive buffer is unpacked in the *edge-Vunpack* 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:**

```fortran
  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:**

```fortran
  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 coprocessor 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:**

```fortran
  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
  end function
```
4 Ensemble consistency testing results

In this section we test whether the ensemble distribution suggested in [2] 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 CESM-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 [2]). Note that the Intel ensemble is the 151 member set generated on Yellowstone and the suggested default for CESM-ECT in [2].

4.1 Code modifications

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. 1 shows that the code modification experiments’ EET failure rates against the Intel, GNU, and PGI compiler CESM-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 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’ variabilities more closely match that of the GNU ensemble than that of Intel or PGI, thus explaining the lower failure rates against the GNU ensemble.
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 CESM-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. 1 against the ensembles composed of them— a total of 562,475 pyCECT evaluations. The results are presented in Fig. 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 [2], 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. 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. 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. 2b are evidence that 151 member ensembles with a single compiler are variationally deficient. Notice that experiments in both plots of Fig. 2 manifest failure rates comparable to those of the code modification experiments in Fig. 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 CESM-ECT false positive rate.

5 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. 4.2. 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
Figure 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.

6 Applying the new ensemble

The results from CESM-supported machine testing in [2] with CESM-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 CESM-ECT to determine whether there is truly a machine issue or whether the initial CESM-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 [2] that pass CESM-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 [8] 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 large 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.

7 Conclusions and Future Work

In this paper, we introduce minimal and legitimate code modifications into CESM to test whether the CESM-ECT ensembles from [2] 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 CESM-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 CESM-ECT failure rates and correct it by disabling FMA. The improved CESM-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.

Acknowledgements

This research used computing resources provided by the Climate Simulation Laboratory at NCAR’s Computational and Information Systems Laboratory (CISL), sponsored by the National Science Foundation and other agencies. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was funded in part by the Intel Parallel Computing Center for Weather and Climate Simulation (https://software.intel.com/en-us/articles/intel-parallel-computing-center-at-the-university-of-colorado-boulder-and-the-national).

References


