

Specification and Verification for Climate Modeling: Formalization Leading to Impactful Tooling

Alper Altuntas

NSF National Center for Atmospheric Research
Boulder, CO, USA

altuntas@ucar.edu

Allison H. Baker

abaker@ucar.edu

John Baugh

North Carolina State University
Raleigh, NC, USA

jwb@ncsu.edu

Ganesh Gopalakrishnan

University of Utah
Salt Lake City, UT, USA

ganesh@cs.utah.edu

Stephen Siegel

University of Delaware
Newark, DE, USA

siegel@udel.edu

Earth System Models (ESMs) are critical for understanding past climates and projecting future scenarios. However, the complexity of these models, which include large code bases, a wide community of developers, and diverse computational platforms, poses significant challenges for software quality assurance. The increasing adoption of GPUs and heterogeneous architectures further complicates verification efforts. Traditional verification methods often rely on bitwise reproducibility, which is not always feasible, particularly under new compilers or hardware. Manual expert evaluation, on the other hand, is subjective and time-consuming. Formal methods offer a mathematically rigorous alternative, yet their application in ESM development has been limited due to the lack of climate model-specific representations and tools. Here, we advocate for the broader adoption of formal methods in climate modeling. In particular, we identify key aspects of ESMs that are well suited to formal specification and introduce abstraction approaches for a tailored framework. To demonstrate this approach, we present a case study using CIVL model checker to formally verify a bug fix in an ocean mixing parameterization scheme. Our goal is to develop accessible, domain-specific formal tools that enhance model confidence and support more efficient and reliable ESM development.

1 Introduction

Increasingly frequent extreme weather events [40] and possible climate tipping points such as the weakening or collapse of the Atlantic Meridional Overturning Circulation [12] pose significant risks. Earth system models (ESMs) are among the most comprehensive tools for studying such phenomena by capturing the complexity of climate dynamics and combining atmospheric, oceanic, land, and cryospheric processes into a coupled framework. Over decades, these models have evolved to encompass millions of lines of code that represent a wide range of physical, chemical, and biological processes governing the Earth system. Through the representation of such diverse and interacting processes, ESMs enable researchers to explore complex feedback mechanisms, predict potential future scenarios, and inform both our understanding and policy decisions.

To accurately simulate the behavior of each subsystem, ESMs are composed of a number of components, each responsible for computing the time evolution of physical phenomena within their respective domains. For example, atmospheric components simulate wind patterns, temperature, and precipitation dynamics, while ocean components model circulation, heat transport, and mixing. The coupled system typically includes a driver, which orchestrates execution during runtime, and a coupler, which handles

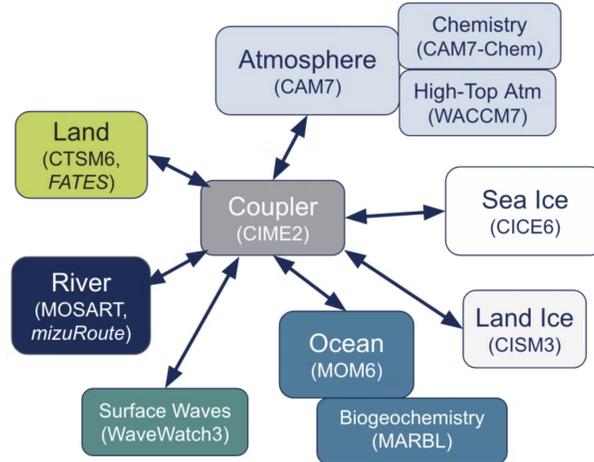


Figure 1: The components of the latest version of the Community Earth System Model (CESM3). Each component interacts with the rest of the system via the coupler in a hub-and-spoke structure [36].

the exchange of data between components. Figure 1 illustrates the components of the latest version of a widely used model, the Community Earth System Model (CESM), and its hub-and-spoke architecture.

Because of their inherent complexity and near continuous development, ESMs are extremely challenging to maintain and validate. Software quality assurance for ESMs relies heavily on traditional testing methodologies, including regression tests that often depend on bitwise reproducibility, meaning that a simulation must produce identical numerical results down to the last bit when re-run with the same inputs. When exact reproducibility is not feasible, such as under new compilers or hardware, field experts must manually evaluate model outcomes, which is both time-consuming and subjective [4]. While ensemble consistency testing [7, 43] offers solutions in such scenarios, development of new modules or algorithms, especially those lacking existing oracles, necessitates additional verification methods.

In this paper, we share our perspectives on how formal methods can be utilized for specifying and verifying ESM processes and mechanisms. Towards this goal, we explore various abstractions and modeling approaches to facilitate such specification and verification efforts. Since formal specification languages and tools often demand specialized expertise and lack the necessary constructs to effectively represent ESM-specific concepts, we envision a comprehensive framework that incorporates ESM-specific constructs, including the continuous evolution of quantities, discrete updates, spatial discretizations, and concurrency. By introducing abstractions tailored to these features, we aim to lower the barriers to applying formal methods, providing explicit representations that facilitate reasoning about data dependencies and algorithmic machinery found in ESMs. Consequently, our approach will enable the formal verification of properties of interest that are typically tested implicitly or through ad hoc methods, thereby enhancing the reliability and correctness of ESMs. This can complement existing verification and validation efforts for both the development of new models and the analysis of existing codebases.

This paper is organized as follows. In Section 2, we identify the key modeling aspects in ESM development that are amenable to formal specification and highlight the challenges arising from these aspects. Section 3 provides a summary of current practices in ESM verification and validation. In Section 4, we discuss our perspectives and abstraction approaches towards a new specification framework. Section 5 demonstrates these concepts through a test case, illustrating how the proposed abstractions can be applied in practice. Finally, Section 6 offers concluding remarks and future directions of this work.

2 Targeted Modeling Aspects

Here, we summarize some high-level modeling aspects and their associated challenges, which are critical for ensuring correctness and are well-suited to the application of formal specification and verification.

2.1 Model Coupling

At every coupling timestep, which is typically on the order of an hour, states and fluxes are exchanged along the interfaces between ESM components that interact. This process, known as inter-component coupling, is the mechanism by which the components inform each other and incorporate the effects of other subsystems. For instance, the ocean component relies on inputs such as atmospheric pressure, wind stress, and heat flux to compute surface currents, temperature changes, and other ocean dynamics. Similarly, the land component requires precipitation and radiation inputs from the atmosphere to simulate processes like runoff and energy exchange. Such data dependencies pose constraints that necessitate careful configuration of run sequencing and processor layout.

In addition to inter-component coupling, ESMs also employ intra-component coupling mechanisms, such as one-way and two-way nesting and adaptive mesh refinement. Furthermore, the combination of inter- and intra-component coupling, such as those found in multi-instance runs [5], introduces additional complexities. The increasing use of Machine Learning (ML), whether for parameterizations or as complete surrogate models, further complicates module dependencies, system robustness, and the handling of edge cases.

2.2 Continuity and Discreteness

A key aspect of our proposed specification approach is the distinction and the interplay between continuous and instantaneous processes. This distinction aligns with the breakdown of ESM components into three main categories: (1) dynamical core, (2) parameterizations, and (3) computational methods.

The governing fluid and thermodynamic equations, formulated and solved numerically as partial differential equations (PDEs), form the foundation of *dynamical cores* [54]. These equations describe the resolved large-scale motion of fluids, such as air in the atmosphere or water in the ocean, as well as the associated energy and mass transport processes. Although these PDEs are discretized in time and space to enable numerical computation, they are fundamentally continuous in nature. That is, they describe smooth changes in the state variables (e.g., velocity, temperature, and density) over time and space. This smoothness contrasts with the inherently discontinuous or sub-grid-scale processes, which are represented differently in ESMs. As an example of a continuous process, Figure 2 illustrates the smooth evolution of ocean currents simulated by a model [41, 45].

Parameterizations, on the other hand, encapsulate empirical and theoretical algorithms that represent processes unresolved by the model’s spatial and temporal resolution. Examples include turbulence, cloud formation, and ocean mixing. Some of these processes may appear *instantaneous*, for instance, when triggered by threshold-based or switch-like criteria, as with wetting and drying algorithms in ocean models [8] or ice-sheet calving [10]. Others, such as vertical ocean mixing [35] and radiative transfer [47], generally evolve more smoothly over time. Even in schemes considered “continuous,” discontinuities can still occur. For example, in the K-Profile Parameterization (KPP) [35], the ocean boundary-layer depth is determined by stability criteria that may shift from one timestep to the next, causing changes in mixing rates. Parameterizations are critical for closing the system of equations and ensuring that models remain computationally feasible while capturing the essential behavior of unresolved processes. How-

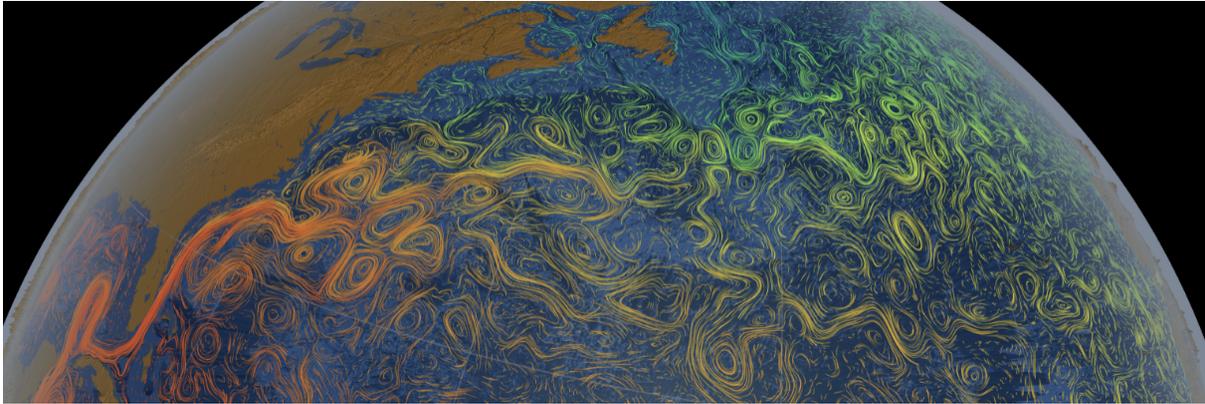


Figure 2: The visualization of the Gulf Stream as simulated by the MIT General Circulation Model (MITgcm). The ocean flows are color-coded with sea surface temperature data. Adapted from [41, 45].

ever, their reliance on empirical approximations introduces uncertainties that affect the overall reliability of model predictions. Unlike the governing equations in dynamical cores, parameterizations often lack direct theoretical or analytical “oracles” to guide their design, and they are often algorithmically complex with extensive logical branching (e.g., the KPP scheme [35]). This complexity is compounded by the interplay between continuous and instantaneous processes, which adds significant challenges to accurately representing these sub-grid-scale phenomena.

Beyond the physical processes described above, ESMs also employ a range of *computational methods* to ensure numerical accuracy, stability, and efficiency. Examples include remapping in Arbitrary Lagrangian–Eulerian (ALE) algorithms [29], adaptive mesh refinement [58], and halo exchanges, where parallel communication is used to exchange boundary data between adjacent computational subdomains assigned to each processing element. These steps, typically executed *instantaneously* at predefined points in the simulation workflow (e.g., after each timestep), do not represent physical phenomena but are instead essential for preserving solution integrity across evolving model grids, coordinating parallel processes, and maintaining computational performance.

2.3 Spatial Discretization

The governing equations in a dynamical core must be discretized in both time and space to enable numerical solutions. In the spatial dimension, this involves using a grid to represent either the entire globe (in global models) or a selected region (in regional models). ESM grids are categorized as either structured or unstructured. Structured grids exhibit uniform connectivity, whereas unstructured grids accommodate irregular connectivity and cell shapes.

Here, we focus on structured, logically rectangular grids, among the most common discretization choices in ESMs. These grids preserve a consistent indexing scheme in each dimension, even when mapped onto spherical geometries and curvilinear coordinates, thereby simplifying numerical methods. Figure 3 shows an example of a structured grid, specifically a tripolar grid, frequently used in ocean modeling to avoid complications arising from polar singularities in latitude–longitude grids.

The representation of spatial grids in ESMs presents a set of challenges. Each component typically operates on a unique grid tailored to its resolution and physical characteristics, requiring remapping or interpolation for data exchange across components with different grids. The placement or “staggering”

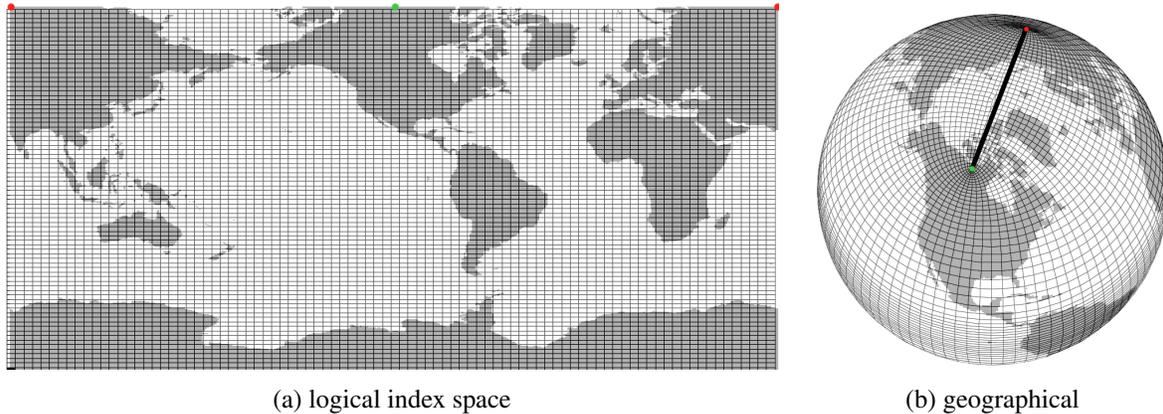


Figure 3: A tripolar grid. These are commonly used in ocean models to avoid singularities in open water by shifting the nominal pole onto land. Panel (a) illustrates the grid in index space, where the northern edge is periodically folded to create the tripolar stitch along the grid lines connecting the two red points (same geographical location) to the green point. Panel (b) displays the same grid in geographical coordinates, with the dark line between the singularity points indicating the tripolar stitch.

of physical and dynamical fields within grid cells must be carefully handled to avoid numerical errors and ensure consistency. Additionally, periodic boundary conditions (e.g., wrapping east–west boundaries or folding the north edge in tripolar grids) must be correctly implemented to maintain continuity. Finally, adaptive approaches, such as adaptive mesh refinement or Arbitrary Lagrangian–Eulerian (ALE) algorithms introduce further challenges.

The decomposition of model grids for parallel computing introduces even greater complexity. The model domain is partitioned into subdomains, each assigned to a processor, with halo exchanges occurring at subdomain boundaries to facilitate interprocessor communication through consistent variable updates. The shift toward GPU-based systems and hybrid parallelism (e.g., combining MPI with OpenMP or GPU directives) adds new demands for code refactoring, data transfer optimization, and careful management of mixed CPU–GPU resources.

3 Current verification and validation practices

Here, we briefly describe current and previous approaches to evaluating software correctness in the climate modeling community. Specifically, we summarize scientific methods based on domain expertise, traditional software techniques such as testing, and novel approaches leveraging formal verification.

3.1 Scientific Methods

Climate scientists often evaluate code changes in a time-consuming and subjective manner that involves comparing new experimental results with control runs and observational data [22]. For example, hind-casting is a common technique that involves running a model simulation from a past state with known conditions (e.g., pre-industrial) through to the present day. The quality of the model output can then be evaluated against available observational data from the recent past. While this approach has been used since the early days of climate modeling, it requires extensive computational resources for the long

simulations, and exhaustively testing the many available model configurations (e.g., grid type, resolution, initial and boundary conditions, numerical method, etc.), is not practical [16]. Further, the manner in which domain experts analyze and compare model data is not standardized and typically dependent on the scientist’s interests and expertise. Another means of evaluating scientific correctness of models is via benchmarks and model intercomparisons. Benchmarks have long been popular tools in the high-performance computing community, and have recently gained traction for evaluating correctness and performance in climate modeling. As an example, the High Performance Climate and Weather (HPCW) benchmark [2] comprises a collection of realistic, near-operational weather and climate workloads, encompasses a number of popular models, and incorporates test cases, verification procedures, and metrics for comparative analysis. Model intercomparison projects are also quite popular and necessary for assessing the relative strengths and weaknesses of various coupled climate models through detailed comparisons. Most notably, modeling centers around the globe participate in the Coupled Model Intercomparison Projects (CMIPs) [1]. Each CMIP, including most recent CMIP phase 6 (CMIP6) [23], contains a variety of experiments designed to evaluate past, present, and future climate states. The CMIP results are particularly useful for quantifying the uncertainty for future climate projections, and understanding factors in differences across model projections leads to improvements in future model development.

3.2 Software Methods

Software testing to verify model behavior and detect bugs is critical when developing large and complex simulation codes like ESMs. While climate modeling centers regularly employ various traditional testing approaches such as unit testing, regression testing, and system testing, these approaches should be used to a much larger extent than currently done [4]. Though unfortunately, many tools and technologies developed thus far are not yet suited for large and complex HPC codes like ESMs [27], and this issue leads to the frequent utilization of application-specific custom tests in the climate modeling community to verify model-specific behaviors. We note that a particular challenge for verification of these models is that yielding bitwise reproducible results is often unattainable due to the chaotic nature of these models and the diverse hardware and software environments in which they operate. When bitwise reproducibility is not possible, scientists may be faced with the challenge of determining whether the difference in simulation results is due to an error or simply to the model’s natural variability. And in this case, experts may be more likely to evaluate model results in a time-consuming and subjective manner.

Addressing this correctness question without bitwise reproducibility is somewhat tractable when the difference (e.g., a compiler upgrade, a port or optimization for different hardware, or lack of associativity in floating-point arithmetic) is not expected to change the scientific conclusions, and in such cases a statistical approach can be utilized. In particular, rather than determine whether new results from a non bit-reproducible modification are correct, we address whether the new results are statistically distinguishable from the previous (control) results. An example of such a statistical approach is a technique known as ensemble consistency testing, which is being increasingly utilized (e.g., [7, 43]).

Yet many modifications in the software lifecycle, such as adding a new physical process, are expected to produce “answer-changing” results. Moreover, testing is inherently incomplete; as Dijkstra famously stated, “Testing can be used to show the presence of bugs, but never to show their absence” [20], further underscoring the need for complementary verification techniques beyond traditional testing.

3.3 Formal Specification and Verification

Formal methods [15, 55, 56] are mathematically based techniques for specifying, analyzing, and verifying system properties throughout the development lifecycle. They provide a systematic alternative to ad hoc approaches, promoting correctness, consistency, and completeness. At the specification level, they enable precise descriptions of system behavior without constraining implementation. In design, techniques like data refinement and simulation proofs support structured development. At the implementation level, methods such as Floyd-Hoare logic ensure correctness by using logical assertions to establish that a program meets its specification. Approaches range from theorem proving and model checking to automatic code generation.

Existing tools aim to balance expressiveness and tractability, often employing semantic restrictions to make models easier to reason about and ensure computational feasibility. These tools are typically tailored for specific domains, such as hardware verification, communication protocols, or safety-critical systems, and are optimized to verify the kinds of correctness properties most relevant to those contexts—such as deadlock freedom, synchronization, or logical consistency.

Adapting them for scientific computing introduces additional challenges. Not only must they handle the inherent complexity of large-scale systems, but they must also abstract from or integrate aspects of numerical analysis, such as stability and precision, in their frameworks. Moreover, identifying and targeting bugs most relevant to scientific computing—whether they arise from software structure, parallel execution, or numerical interactions—requires rethinking how existing tools can be extended or repurposed for this domain.

Recent work in applying formal methods to scientific computing has developed along two complementary lines. The first incorporates numerical concerns directly into the reasoning process. Bientinesi et al. [11] use Floyd-Hoare logic to derive and prove the correctness of dense linear algebra algorithms, employing simplified matrix notation to hide indexing details and systematically identify invariants for nested loops. Kellison et al. [32] used deductive verification techniques, rooted in the Coq proof assistant [17] and the Verified Software Toolchain, to prove accuracy and correctness claims of several basic linear algebra routines expressed in C, including sparse matrix algorithms. Siegel et al. [51] present a framework for verifying real, IEEE floating-point, or exact equivalence between sequential and parallel numerical programs, using symbolic expressions and the SPIN model checker. The CIVL Model Checker [52] takes a similar approach to the analysis of scientific C code that uses MPI, OpenMP, CUDA, or Pthreads. Symbolic Taylor Forms—a rigorous methods for tightly bounding floating-point errors—was introduced in [53] and significantly scaled up in [18], with conditional branch instability analyzed in [19]. Mapping to heterogeneous hardware comes with many additional defect-possibilities that are surveyed in [28] with a GPU-specific tool for detecting floating-point exceptions contributed by [37].

Altuntas and Baugh [6] apply hybrid theorem proving to model numerical software as hybrid systems, combining discrete updates and continuous dynamics. Their case study identifies undesirable behavior encountered in ocean components of climate models, demonstrating the potential of this approach to improve confidence in numerical simulations.

The second line separates concerns by abstracting from numerical computations and focusing on the software structure that integrates and manages these computations, emphasizing structural correctness in a variety of applications. Baugh et al. [8] use Alloy to verify equivalence-preserving extensions to ADCIRC, a large-scale hurricane storm surge model used in production, ensuring the soundness of implementation choices. Dyer et al. [21] demonstrate how sparse matrix operations, including multiplication and format conversions, can be modeled and verified using Alloy, relying on bounded analysis to ensure structural refinement. Benavides et al. [9] explore refinement checking in high-performance com-

puting applications, using Alloy to model single program, multiple data (SPMD) parallelism in Coarray Fortran, with an example of an iterative Jacobi solver for Laplace’s equation.

4 Towards a Relational Spatio-Temporal Logic

In this section, we introduce our abstraction approaches and perspectives for effectively specifying key aspects of ESMs. Our aim is to establish a foundation for a relational spatio-temporal logic, i.e., a modeling framework that enables ESM developers to specify critical structures and behaviors of interest within ESMs, reason about their interactions, and formally verify key correctness properties.

4.1 Model Coupling and Concurrency

Concurrency in Earth System Models (ESMs) occurs across multiple levels: from inter-component and inter-process interactions to GPU thread management. These instances of concurrency within ESMs present unique challenges but also offer opportunities for formal specification, particularly due to the modular structure of most ESMs, where concurrency typically occurs between well-defined interfaces. For example, at the intercomponent level, the thin "model cap" layers found in popular models like CESM, along with the coupler, provide natural boundaries for defining assumptions and guarantees.

As such, we propose representing concurrency via compositional reasoning, e.g., assume-guarantee reasoning [25, 30]. This methodology has been successfully applied in other contexts [24, 49], and is similarly well-suited for our purposes since explicitly specifying every component of a coupled ESM is impractical. Instead, a targeted strategy similar to lightweight formal methods is both necessary and achievable. Within our specification framework, we therefore propose a higher-level analysis and representation that emphasizes data dependencies and the sequencing of events.

A practical example of this approach can be utilized in reasoning about the run sequencing of model components within CESM and similar models. Until recently, run sequencing was implemented through hard-coded subroutine calls. More recently, configuration input files, known as NUOPC run sequencing files [46], have been used to specify run sequencing in a more configurable manner, marking significant progress. However, there is still room for improvement: the rationale behind certain run sequencing configurations is sometimes unclear. By explicitly defining dependencies rather than merely specifying the resulting sequence, run sequencing can be automatically generated using techniques such as SAT solvers, thereby facilitating the optimization of concurrency, all while preserving the integrity of the scientific results.

As for process-level concurrency, which is extensively studied in the formal methods community, we propose leveraging existing tools such as CIVL to establish equivalence between sequential and parallel numerical programs [51]. Similarly, for GPU parallelism, tools such as Faial [38] and HiRace [31] can be employed to detect data races, thus helping enhance correctness at the thread level.

4.2 Continuous and Instantaneous Processes

Our approach to effectively specifying and reasoning about continuous and instantaneous behaviors draws inspiration from methodologies in the cyber-physical systems (CPS) domain. In CPS, modelers abstract the inherently continuous real world by treating certain rapidly occurring events as instantaneous, e.g., modeling the bouncing of a ball [48].

In contrast, numerical modeling software, such as Earth System Models (ESMs), are fundamentally discrete. However, some processes, such as those simulated by dynamical cores, evolve smoothly over

time. We therefore adopt a similar abstraction in the opposite direction: by assuming continuity for these smoothly evolving processes, we can focus on their interactions and facilitate efficient reasoning. Instead of attempting to incorporate the discrete computation of partial differential equations (PDEs) directly, which becomes intractable with even a small number of timesteps and grid points, we propose modeling PDEs as continuous in time but discrete in space. This abstraction, commonly used in numerical analysis, is known as the method of lines, where a PDE system is transformed into a system of ordinary differential equations (ODEs). By doing so, we enable the use of formal verification techniques designed for continuous systems, such as those developed in the CPS community.

We list several alternative methods for specifying PDEs as ODEs and reasoning about their behavior: One such method is *Hybrid Programming* along with *Differential Dynamic Logic* [48], which allows for the specification of differential invariants to reason about continuous behavior without explicitly solving the ODE system, as demonstrated in [6]. Another possible approach is *symbolic solution* of ODEs, particularly when closed-form solutions exist. In cases where closed-form solutions are unavailable, *numerical solutions* of ODEs, as is done in the case study in the next section, may be employed. Additionally, to increase confidence in numerical solutions, *probabilistic and stochastic approaches* can be used. These include numerically computing ODEs multiple times with precision-level perturbations and running multiple ensembles, inspired by ensemble consistency testing [7], or employing stochastic methods commonly used in the CPS domain for reachability analysis [13]. Finally, when dealing with original PDEs that are too complex for direct integration into a formal framework, simpler *surrogate ODEs* (e.g., linear shallow water equations instead of nonlinear momentum and continuity equations) or *nondeterministic representations* of continuous behavior can be utilized.

By employing one or a combination of these methods, we can effectively model the continuous time evolution of ESMs while maintaining computational tractability and facilitating formal reasoning about interactions and system behavior.

Reals vs. Floating-Point Arithmetic Floating-point arithmetic introduces a range of issues that can compromise model correctness, including rounding errors, precision limitations, and numerical instability. However, maintaining continuity in time often necessitates working with real numbers rather than floating-point representations. This requirement is particularly relevant for methods such as differential dynamic logic and symbolic solutions. While reasoning about floating-point arithmetic is crucial, especially when feasible, it often introduces significant complexity and computational overhead that can impede formal verification efforts. Although the issues related to floating-point arithmetic are critical and remain active areas of research, by abstracting away from the intricacies of floating-point computations, we can focus on higher-level aspects of the model, such as the correctness of algorithms, data dependencies, and the formal verification of interactions between components. This abstraction enables a clearer and more manageable framework for reasoning about system behavior, facilitating the identification and resolution of logical and structural issues that are often overshadowed by numerical concerns.

Instantaneous events: Specifying discrete algorithms, such as parameterization routines and computational processes, is relatively straightforward when integrating them into an abstract model. This is because they map naturally onto discrete transitions, which are the core building blocks of most formal methods. However, there is a crucial distinction: ESM developers typically utilize procedural programming paradigms, whereas formal specification relies on declarative paradigms. Bridging this gap is essential. Towards that end, one approach may be to adopt intermediate representations, for example, by using state transition operators as implemented in tools like Alloy 6.

4.3 Mesh representations

In ESM codebases, grids are typically represented as *structures of arrays*, where each horizontal grid property (e.g., the coordinates, areas, angles, etc.) is stored in two-dimensional arrays. This also applies to the physical quantities incident on grid cells (e.g., temperature, density, fluxes, etc.). The indices i and j , then, correspond to the logical row and column identifiers that locate each cell (or node) in the global model domain, abstracting away the underlying spherical or regional map projection.

In our abstraction approach, we propose maintaining this logical indexing-based representation. However, we argue for the specification of a minimal grid: as small as the local domain of dependence [6]. In practice, this often means focusing on a single grid cell (or node) and its immediate neighbors in each dimension, sufficient to capture the local PDE stencil or parameterization dependencies. This is justified because both discrete and continuous processes generally have a very limited domain of dependence. In the case of continuous processes, this is particularly true since hyperbolic PDEs (e.g., those governing fluid flow) have a finite propagation speed. The Courant-Friedrichs-Lewy (CFL) condition restricts how far information can travel in a single timestep, ensuring that any local update only depends on a limited set of neighboring cells. Consequently, spatial data dependencies do not extend arbitrarily beyond those nearby cells, so abstracting the global domain into a small grid region plus nondeterministic external inputs allows compositional reasoning without enumerating the entire grid.

Representing staggering, i.e., the placement of quantities on different locations of a grid cell (e.g., cell center, edges, or corners), can be based on viewing grids as a Hasse diagram, as was done by Knepley et al. for parallel mesh and data distribution, load balancing, and overlap generation [33], or similarly via relational logic [8]. These approaches facilitate reasoning about adaptive algorithms like AMR, parallel operations such as halo exchanges, east-west wrapping, and other specialized boundary handling mechanisms that maintain grid continuity, as well as other grid-related algorithms essential for ensuring computational accuracy and efficiency.

5 A Case study

The hybrid model studied in [6] is known as the K-profile parameterization (KPP) scheme. This model, which is available online [3], involves control iterating a finite, though unspecified, number of times. At each iteration, there is a sequence of discrete, instantaneous updates to the state, followed by the continuous evolution of the state for some non-zero, finite amount of time. The continuous evolution is specified by an ordinary differential equation. In this case, the equation has the form $y' = -y$, where y' is the time-derivative of variable y . KeYmaera X [26] was used to find a defect in this scheme in which a variable K , which should always be positive, in fact becomes negative. The defect was repaired by modifying the function that updates K , and KeYmaera was used to construct a proof, in differential dynamic logic, that K remains positive and other invariants hold.

Here we describe a more lightweight modeling approach for the same problem. We wrote a model of the hybrid program in CIVL-C, the language of the CIVL Model Checker (Fig. 4). CIVL-C is an intermediate verification language with syntax and semantics that are similar to other imperative programming languages. The model checker analyzes a CIVL-C model using symbolic execution, i.e., the values assigned to variables are symbolic expressions rather than concrete numbers.

There is no way to express the continuous evolution of a quantity, governed by an ODE, in CIVL-C. Instead, we have modeled this by a simple first order scheme to approximate the result. This involves a loop in which time is incremented by Δt at each iteration; the number of iterations is chosen nondeterministically (lines 66–70).

```

1 $input int N; // number of outer iterations
2 $input int M; // max number inner iterations
3 $input double dt; // delta_t
4 $assume(0<dt && dt<1);
5 $input double zw, D, w;
6 $assume(D>0 && D>zw && zw>0 && w>0);
7 double t=0.0; // time
8 double nu, dnu, h, sigma, alpha, zCr, K, a2, a3;
9
10 double G(double sigma, double a2, double a3) {
11     return sigma +
12         a2*$pow(sigma,2) + a3*$pow(sigma,3);
13 }
14
15 void computeNu(void) {
16     $havoc(&nu);
17     $assume(nu>0);
18     $havoc(&dnu);
19 }
20
21 void computeBLD(void) {
22     h = D - zCr;
23     sigma = (D - zw)/h;
24     $havoc(&alpha);
25     $assume(0<alpha && alpha<1);
26     zCr = alpha*zCr;
27 }
28
29 void computeK(void) {
30     a2 = -2 + 3*nu/(h*w) + dnu/w;
31     a3 = 1 - nu/(h*w) - dnu/w;
32     K = h*w * G(sigma, a2, a3);
33 }
34
35 void invariant(void) {
36     $assert(K>0);
37     $assert(zw>=zCr);
38     $assert(zCr>0);
39 }
40 void initialConditions() {
41     $havoc(&nu);
42     $havoc(&dnu);
43     $havoc(&h);
44     $havoc(&sigma);
45     $havoc(&alpha);
46     $havoc(&zCr);
47     $havoc(&K);
48     $havoc(&a2);
49     $havoc(&a3);
50     $assume(0<nu && K>0);
51     $assume(zCr == zw);
52 }
53
54 void printState(void) { ... }
55
56 int main(void) {
57     printState();
58     initialConditions();
59     for (int i=0; i<N; i++) {
60         printState();
61         invariant();
62         computeBLD();
63         computeNu();
64         computeK();
65         // zCr'=-zCr ...
66         int m = $choose_int(M);
67         for (int j=0; j<m; j++) {
68             t += dt;
69             zCr += -zCr*dt;
70         }
71     }
72     printState();
73     invariant();
74 }

```

Figure 4: CIVL-C model of the KPP defect fix.

Using this model, we are able to find the same defect reported in the earlier work. This manifests as a violation of the assertion $K > 0$ (line 36) found with $N = 2$ and $M = 1$ in less one second. The model checker also prints the symbolic values of all variables that lead to the violation. When function `computeK` is replaced with the corrected version, no violations are found. For $N = M = 3$, verification takes 1.5 seconds. This time includes 31 calls to the automated theorem prover Z3 [44], which easily discharges the assertions. The CIVL approach provides strong evidence for the correctness of the fix, though not as strong as a formal proof. Nevertheless, it was straightforward to write the model and the verification process itself is fully automated.

Although tools like KeYmaera X provide the additional rigor of formal proofs, lightweight tools such as CIVL can play a critical role in the early stages of the design and implementation process. The ability to quickly find defects and receive automated feedback helps developers iterate rapidly and refine their models without the overhead of constructing formal proofs at each step. Once these preliminary checks confirm the absence of obvious errors, investing additional effort in proof tools like KeYmaera X becomes more strategic and meaningful.

Finally, CIVL’s support for modeling concurrency, a fundamental feature of many scientific codes, highlights a complementary strength compared to tools like KeYmaera X, which focus on formal proofs but lack concurrency support. While concurrency was not required for this specific problem, this capability, combined with CIVL’s lightweight and automated approach, underscores the broader potential for lightweight formal methods to assist developers of ocean models in debugging and analyzing their algorithms effectively.

6 Conclusions

Formal methods offer a promising path toward improving the reliability and trustworthiness of ESMs. Given the complexity of ESMs and their reliance on bitwise reproducibility and manual analysis, formal specification and verification can serve as a valuable complementary approach. By defining explicit correctness properties and systematically checking them, formal methods can bolster current practices, reduce subjectivity in model assessments, and help detect subtle software or algorithmic defects that might otherwise go unnoticed.

Lightweight formal methods, in particular, like those employed by CIVL offer a practical and scalable approach to verifying numerical software. CIVL’s syntax and semantics closely resemble imperative programming languages, making it accessible to scientific software developers while enabling automated verification. Ability to rapidly detect defects and provide automated feedback allows developers to iterate efficiently without the overhead of constructing full formal proofs at every step. Our case study, for instance, demonstrated how CIVL can uncover subtle defects in ocean model parameterization schemes and validate corrections efficiently. Expanding the use of these lightweight verification techniques in ESMs, particularly for parallelism, floating-point precision, and complex data dependencies, could significantly enhance model reliability.

Our work highlights several essential features of ESMs that are well suited for formal reasoning, including component coupling and concurrency, the interplay of continuous and discrete processes, and the intricate spatial discretization considerations. We have proposed abstraction approaches for these complexities to allow for more systematic verification of model components. Although our initial exploration demonstrates the feasibility of using formal methods, we recognize that substantial effort is needed to create tools and abstractions that climate scientists can readily adopt.

A crucial next step involves addressing correctness in large-scale GPU simulations, such as those implemented with AMReX, A software framework for massively parallel, block-structured AMR applications [58]. While AMReX provides essential capabilities like domain periodicity, implementing specialized boundary handling, such as for tripolar grids, introduces significant concurrency and correctness challenges. Formal specification can help ensure that new routines, parameterization logic, and hardware adaptations maintain critical invariants, such as stable boundary operations, through automated verification. Moreover, formal methods can proactively address potential data races in concurrent execution and mitigate numerical inconsistencies that may arise from compiler variations or hardware differences [42], improving the robustness and reproducibility of large-scale climate simulations.

Moving forward, bridging the gap between formal methods research and climate modeling practice will require interdisciplinary collaboration. By developing tools that integrate seamlessly into existing workflows, we can enable broader adoption of specification and verification techniques. Our long-term vision is to establish formal methods as a complementary component of climate model development, enhancing both scientific rigor and predictive reliability.

References

- [1] *CMIP - The Coupled Model Intercomparison Project*. <https://wcrp-cmip.org/cmip-overview/>. Accessed: 2025-01-20.
- [2] *HPCW - The High Performance Climate & Weather Benchmark*. <https://www.esiwace.eu/services/hpcw>. Accessed: 2025-01-20.
- [3] Alper Altuntas (2018): *KeYmaera X models of the KPP scheme*. Available at <https://github.com/alperaltuntas/keymaera-kpp>. Accessed: 2025-03-20.
- [4] Alper Altuntas, Allison Baker, Ilene Carpenter, Brian Dobbins, Michael Duda, Dorit Hammerling, Thomas Hauser & Karsten Peters-von Gehlen (2024): *Report on the 2023 Workshop on Correctness and Reproducibility for Climate and Weather Software*. Technical Report NCAR/TN-582+PROC, National Center for Atmospheric Research, doi:10.5065/0534-mc88.
- [5] Alper Altuntas & John Baugh (2017): *Verifying concurrency in an adaptive ocean circulation model*. In: *Proceedings of the First International Workshop on Software Correctness for HPC Applications*, pp. 1–7.
- [6] Alper Altuntas & John Baugh (2018): *Hybrid theorem proving as a lightweight method for verifying numerical software*. In: *2018 IEEE/ACM 2nd International Workshop on Software Correctness for HPC Applications (Correctness)*, IEEE, pp. 1–8.
- [7] AH Baker, DM Hammerling, MN Levy, H Xu, JM Dennis, BE Eaton, J Edwards, C Hannay, SA Mickelson, RB Neale et al. (2015): *A new ensemble-based consistency test for the Community Earth System Model (pyCECT v1. 0)*. *Geoscientific Model Development* 8(9), pp. 2829–2840.
- [8] John Baugh & Alper Altuntas (2018): *Formal methods and finite element analysis of hurricane storm surge: A case study in software verification*. *Science of Computer Programming* 158, pp. 100–121.
- [9] Juan Benavides, John Baugh & Ganesh Gopalakrishnan (2023): *An HPC practitioner’s workbench for formal refinement checking*. In C. Mendis & L. Rauchwerger, editors: *Languages and Compilers for Parallel Computing, LCPC 2022*, Springer, Cham, pp. 64–72, doi:10.1007/978-3-031-31445-2_5.
- [10] Douglas I Benn, Charles R Warren & Ruth H Mottram (2007): *Calving processes and the dynamics of calving glaciers*. *Earth-Science Reviews* 82(3-4), pp. 143–179.
- [11] Paolo Bientinesi, John A Gunnels, Margaret E Myers, Enrique S Quintana-Ortí & Robert A Geijn (2005): *The science of deriving dense linear algebra algorithms*. *ACM Transactions on Mathematical Software (TOMS)* 31(1), pp. 1–26.
- [12] Niklas Boers (2021): *Observation-based early-warning signals for a collapse of the Atlantic Meridional Overturning Circulation*. *Nature Climate Change* 11(8), pp. 680–688.
- [13] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay & Zheng Wang (2012): *UPPAAL-SMC: Statistical model checking for priced timed automata*. *arXiv preprint arXiv:1207.1272*.
- [14] Chih-Chieh Chen, Jadwiga H Richter, Walker R Lee, Douglas G MacMartin & Ben Kravitz (2024): *Rethinking the susceptibility-based strategy for marine cloud brightening climate intervention: Experiment with CESM2 and its implications*. *Geophysical Research Letters* 51(10), p. e2024GL108860.
- [15] Edmund M Clarke & Jeannette M Wing (1996): *Formal methods: State of the art and future directions*. *ACM Computing Surveys (CSUR)* 28(4), pp. 626–643.
- [16] T. Clune & R. Rood (2011): *Software Testing and Verification in Climate Model Development*. *IEEE Software* 28(6), pp. 49–55, doi:http://dx.doi.org/10.1109/MS.2011.117.
- [17] *The Coq proof assistant*. Available at <http://coq.inria.fr/>.
- [18] Arnab Das, Ian Briggs, Ganesh Gopalakrishnan, Sriram Krishnamoorthy & Pavel Panchekha (2020): *Scalable yet Rigorous Floating-Point Error Analysis*. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’20*, IEEE Press.

- [19] Arnab Das, Tanmay Tirpankar, Ganesh Gopalakrishnan & Sriram Krishnamoorthy (2021): *Robustness Analysis of Loop-Free Floating-Point Programs via Symbolic Automatic Differentiation*. In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 481–491, doi:10.1109/Cluster48925.2021.00055.
- [20] Edsger Wybe Dijkstra et al. (1970): *Notes on structured programming*.
- [21] Tristan Dyer, Alper Altuntas & John Baugh (2019): *Bounded verification of sparse matrix computations*. In: *Proceedings of the Third International Workshop on Software Correctness for HPC Applications, Correctness'19*, IEEE/ACM, pp. 36–43.
- [22] Steve M. Easterbrook & Timothy C. Johns (2009): *Engineering the Software for Understanding Climate Change*. *Computing in Science and Engineering* 11(6), pp. 65–74, doi:http://doi.ieeecomputersociety.org/10.1109/MCSE.2009.193.
- [23] V. Eyring, S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer & K. E. Taylor (2016): *Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization*. *Geoscientific Model Development* 9(5), pp. 1937–1958, doi:10.5194/gmd-9-1937-2016. Available at <https://gmd.copernicus.org/articles/9/1937/2016/>.
- [24] Chuchu Fan, Bolun Qi, Sayan Mitra & Mahesh Viswanathan (2017): *DryVR: Data-driven verification and compositional reasoning for automotive systems*. In: *International Conference on Computer Aided Verification*, Springer, pp. 441–461.
- [25] Cormac Flanagan & Shaz Qadeer (2003): *Thread-Modular Model Checking*. In Thomas Ball & Sriram K. Rajamani, editors: *Model Checking Software*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 213–224.
- [26] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp & André Platzer (2015): *KeYmaera X: An axiomatic tactical theorem prover for hybrid systems*. In: *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, Springer, pp. 527–538.
- [27] Maya Gokhale, Ganesh Gopalakrishnan, Jackson Mayo, Santosh Nagarakatte, Cindy Rubio-González & Stephen F. Siegel (2023): *Report of the DOE/NSF Workshop on Correctness in Scientific Computing, June 2023, Orlando, FL*. arXiv:2312.15640.
- [28] Ganesh Gopalakrishnan, Ignacio Laguna, Ang Li, Pavel Panchekha, Cindy Rubio-González & Zachary Tatlock (2021): *Guarding Numerics Amidst Rising Heterogeneity*. In: *2021 IEEE/ACM 5th International Workshop on Software Correctness for HPC Applications*, pp. 9–15, doi:10.1109/Correctness54621.2021.00007.
- [29] Stephen M Griffies, Alistair Adcroft & Robert W Hallberg (2020): *A primer on the vertical Lagrangian-remap method in ocean models based on finite volume generalized vertical coordinates*. *Journal of Advances in Modeling Earth Systems* 12(10), p. e2019MS001954.
- [30] Ian J. Hayes & Cliff B. Jones (2018): *A Guide to Rely/Guarantee Thinking*. In Jonathan P. Bowen, Zhiming Liu & Zili Zhang, editors: *Engineering Trustworthy Software Systems*, Springer International Publishing, Cham, pp. 1–38.
- [31] John Jacobson, Martin Burtcher & Ganesh Gopalakrishnan (2024): *HiRace: Accurate and Fast Data Race Checking for GPU Programs*. In: *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 1–14.
- [32] Ariel E. Kellison, Andrew W. Appel, Mohit Tekriwal & David Bindel (2023): *LAProof: A Library of Formal Proofs of Accuracy and Correctness for Linear Algebra Programs*. In: *2023 IEEE 30th Symposium on Computer Arithmetic (ARITH)*, pp. 36–43, doi:10.1109/ARITH58626.2023.00021.
- [33] Matthew G Knepley, Michael Lange & Gerard J Gorman (2015): *Unstructured overlapping mesh distribution in parallel*. arXiv preprint arXiv:1506.06194.
- [34] Leslie Lamport (1978): *Time, clocks, and the ordering of events in a distributed system*. In: *Communications of the ACM*, p. 21 (7): 558.
- [35] William G Large, James C McWilliams & Scott C Doney (1994): *Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization*. *Reviews of geophysics* 32(4), pp. 363–403.

- [36] David Lawrence (2024): *CESM3: Plans, Progress, and Timelines*. Presentation slides from the CESM workshop, June 10, 2024. Boulder, CO.
- [37] Xinyi Li, Ignacio Laguna, Katarzyna Swirydowicz, Bo Fang, Ang Li & Ganesh Gopalakrishnan (2023): *Design and Evaluation of GPU-FPX: A Low-Overhead tool for Floating-Point Exception Detection in NVIDIA GPUs*. In: *ACM HPDC 2023 (the 32nd International Symposium on High-Performance Parallel and Distributed Computing)*.
- [38] Dennis Liew, Tiago Cogumbreiro & Julien Lange (2024): *Sound and Partially-Complete Static Analysis of Data-Races in GPU Programs*. *Proc. ACM Program. Lang.* 8(OOPSLA2), pp. 2434–2461, doi:10.1145/3689797. Available at <https://doi.org/10.1145/3689797>.
- [39] Yangang Liu, Man-Kong Yau, Shin-ichiro Shima, Chunsong Lu & Sisi Chen (2023): *Parameterization and explicit modeling of cloud microphysics: Approaches, challenges, and future directions*. *Advances in Atmospheric Sciences* 40(5), pp. 747–790.
- [40] V. Masson-Delmotte, P. Zhai, A. Pirani, S. L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M. I. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J. B. R. Matthews, T. K. Maycock, T. Waterfield, O. Yeleki, R. Yu & B. Zhou, editors (2021): *Climate Change 2021: The Physical Science Basis*. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, doi:10.1017/9781009157896. Available at <https://www.ipcc.ch/report/ar6/wg1/>.
- [41] Dimitris Menemenlis, Jean-Michel Campin, Patrick Heimbach, Chris Hill, Tong Lee, An Nguyen, Michael Schodlok & Hong Zhang (2008): *ECCO2: High resolution global ocean and sea ice data synthesis*. *Mercator Ocean Quarterly Newsletter* 31(October), pp. 13–21.
- [42] Dolores Miao, Ignacio Laguna & Cindy Rubio-González (2024): *Input Range Generation for Compiler-Induced Numerical Inconsistencies*. In Kenji Kise, Valentina Salapura, Murali Annavaram & Ana Lucia Varbanescu, editors: *Proceedings of the 38th ACM International Conference on Supercomputing, ICS 2024, Kyoto, Japan, June 4-7, 2024*, ACM, pp. 201–212, doi:10.1145/3650200.3656618. Available at <https://doi.org/10.1145/3650200.3656618>.
- [43] D. J. Milroy, A. H. Baker, D. M. Hammerling & E. R. Jessup (2018): *Nine time steps: ultra-fast statistical consistency testing of the Community Earth System Model (pyCECT v3.0)*. *Geoscientific Model Development* 11(2), pp. 697–711, doi:10.5194/gmd-11-697-2018. Available at <https://www.geosci-model-dev.net/11/697/2018/>.
- [44] Leonardo de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In Ramakrishnan & Rehof [50], pp. 337–340, doi:10.1007/978-3-540-78800-3_24.
- [45] NASA Scientific Visualization Studio (2012): *Gulf Stream Sea Surface Currents and Temperatures*. Available at <https://svs.gsfc.nasa.gov/3913>. Accessed: January 17, 2025.
- [46] NUOPC Team (2024): *NUOPC Layer Documentation*. Available at <https://earthsystemmodeling.org/nuopc/>. Accessed: January 17, 2025.
- [47] Robert Pincus & Bjorn Stevens (2013): *Paths to accuracy for radiation parameterizations in atmospheric models*. *Journal of Advances in Modeling Earth Systems* 5(2), pp. 225–233.
- [48] André Platzer (2018): *Logical foundations of cyber-physical systems*. 662, Springer.
- [49] Pavan Rajagopal, James B Dabney & Julia M Badger (2023): *Using Assume-Guarantee Contracts for Operational Verification of Autonomous Spacecraft*. In: *16th Annual Flight Software Workshop*.
- [50] C. R. Ramakrishnan & Jakob Rehof, editors (2008): *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science 4963*, Springer.
- [51] Stephen F. Siegel, Anastasia Mironova, George S. Avrunin & Lori A. Clarke (2008): *Combining symbolic execution with model checking to verify parallel numerical programs*. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 17(2), pp. 10:1–10:34.

- [52] Stephen F. Siegel, Manchun Zheng, Ziqing Luo, Timothy K. Zirkel, Andre V. Marianiello, John G. Edenhofner, Matthew B. Dwyer & Michael S. Rogers (2015): *CIVL: The Concurrency Intermediate Verification Language*. In: *SC15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, New York, doi:10.1145/2807591.2807635. Article no. 61, pages 1–12.
- [53] Alexey Solovyev, Marek S. Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamaric & Ganesh Gopalakrishnan (2019): *Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions*. *ACM Trans. Program. Lang. Syst.* 41(1), pp. 2:1–2:39, doi:10.1145/3230733. Available at <https://doi.org/10.1145/3230733>.
- [54] John Thuburn (2008): *Some conservation issues for the dynamical cores of NWP and climate models*. *Journal of Computational Physics* 227(7), pp. 3715–3730.
- [55] J.M. Wing (1990): *A specifier's introduction to formal methods*. *Computer* 23(9), pp. 8–22.
- [56] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui & John Fitzgerald (2009): *Formal methods: Practice and experience*. *ACM Computing Surveys (CSUR)* 41(4), pp. 1–36.
- [57] Carl Wunsch (2006): *Discrete inverse and state estimation problems: with geophysical fluid applications*. Cambridge University Press.
- [58] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves et al. (2019): *AMReX: a framework for block-structured adaptive mesh refinement*. *The Journal of Open Source Software* 4(37), p. 1370.