2018 IEEE/ACM 2nd International Workshop on Software Correctness for HPC Applications (Correctness)

# Hybrid theorem proving as a lightweight method for verifying numerical software

Alper Altuntas
*National Center for Atmospheric Research*
Boulder, CO
altuntas@ucar.edu

John Baugh
*North Carolina State University*
Raleigh, NC
jwb@ncsu.edu

***Abstract*—Large-scale numerical software requires substantial computer resources that complicate testing and debugging. A single run of a climate model may require many millions of core-hours and terabytes of disk space, making trial-and-error experiments burdensome and time consuming. In this study, we apply hybrid theorem proving from the field of cyber-physical systems to problems in scientific computation, and show how to verify the correctness of discrete updates that appear in the simulation of continuous physical systems. By viewing numerical software as a hybrid system that combines discrete and continuous behavior, test coverage and confidence in findings can be increased. We describe abstraction approaches for modeling numerical software and demonstrate the applicability of the approach in a case study that reproduces undesirable behavior encountered in a parameterization scheme, called the K-profile parameterization, widely used in ocean components of large-scale climate models. We then identify and model a fix in the configuration of the scheme, and verify that the undesired behavior is eliminated for all possible execution sequences. We conclude that hybrid theorem proving is an effective and efficient approach that can be used to verify and reason about properties of large-scale numerical software.**

***Index Terms*—hybrid systems, formal methods, scientific computation, KeYmaera X**

## I. Introduction

The accuracy of physical models depends both on the convergence of numerical methods that approximate continuous physics and on their realization in software, which can be complicated by the discrete computations and state changes that arise from various ad-hoc and empirical modeling considerations. Well-established stability and convergence criteria exist for analyzing finite difference and other numerical schemes, but such approaches offer little guidance when it comes to reasoning about discrete updates to physical parameters, for instance, that may be performed at intermediate points in the computation. As a result, findings can be difficult to validate, and in the absence of test oracles, developers may have to settle for spot checks and plausibility tests based on conservation laws and other principles that are expected to hold [1]. In this study, we draw on the similarities between numerical models of physical processes and cyber-physical systems, which combine discrete and continuous dynamics. By viewing numerical software as a hybrid system [2], we are able to verify some safety properties of interest using tools originally developed for that class of applications.

A cyber-physical system consists of a discrete controller with sensors and actuators that interact with the physical environment. Examples include self-driving cars, air traffic control systems, and industrial robots used in manufacturing applications. Commonly modeled as hybrid systems, they are assumed to have discrete behavior that is instantaneous and intermittent, and continuous behavior that models physical processes, which evolve until the controller intervenes and changes their course instantaneously. Because they are often safety critical, in that human lives and physical well-being depend on them, considerable efforts have been taken to develop tools and approaches to verify their correctness [2]. In this study, we make use of one such tool, KeYmaera X [3], a hybrid theorem prover.

Numerical models may also be viewed as evolving continuously with discrete updates that occur instantaneously and intermittently. An example of such composition may be seen in coastal ocean models, which simulate both the time evolution of water surface elevations and discrete changes in wet-dry states at each grid point, as flood waters advance and recede over land. While water surface elevations can be viewed as evolving continuously, a location on the earth becoming wet or dry is often modeled in software as an instantaneous event, both depending on and affecting the future course of continuously evolving processes. Although the differential equations describing such continuous processes are typically discretized in time and space in a numerical model, they may still be taken to be continuous in a hybrid model, allowing an abstraction from one set of details to achieve better focus on another: the discrete updates taking place at intermediate points in the time evolution of a system.

In hybrid system models [3], discrete and continuous behavior can be captured in *hybrid programs*, which resemble conventional computer programs but may also include the definition of ordinary differential equations (ODEs) with time derivatives. Because we are concerned with the numerical solution of partial differential equations (PDEs) that are discretized in both time and space, we present an abstraction approach that accommodates the use of hybrid models by maintaining the discretization in space but abandoning the discretization in time. Doing so reduces a system of PDEs to a system of ODEs, resulting in a model that evolves continuously in time, with intermittent updates, over a discrete space.

978-1-7281-0226-9/18/$31.00 ©2018 IEEE
DOI 10.1109/Correctness.2018.00005

Since such systems of PDEs have a finite range of influence, as required by the Courant-Friedrichs-Lewy (CFL) condition, the numerical methods employed will be known to have a domain of dependence encompassing the analytical domain of dependence of the PDE. Thus, spatial data dependencies are such that we can work with small, discrete grids and make use of nondeterminism to represent arbitrary states external to them.

Such an approach, for example, may be applied to a numerical model that simulates the one dimensional diffusion of a variable $\lambda$:

$$\frac{\partial \lambda}{\partial t} = \kappa \frac{\partial^2 \lambda}{\partial x^2}$$

where $\kappa$ is the diffusivity and $x$ and $t$ are the space and time variables, respectively. The representation of this PDE in a hybrid verification model may take the following form:

$$\frac{\partial \lambda_i}{\partial t} = \kappa(\lambda_{i-1} - 2\lambda_i + \lambda_{i+1})/(\Delta x)^2$$

where $\Delta x$ is the distance between evenly spaced discrete grid nodes at $i-1$, $i$, and $i+1$. While the spatial discretization is based on a second-order central difference scheme, the time evolution of $\lambda_i$ in this hybrid model remains continuous, in a manner analogous to the method of lines [4], which discretizes all but a single dimension that is left continuous.

A higher level of abstraction may be employed in some cases to reduce the size and therefore improve the tractability of a hybrid model. Instead of including all continuous processes, i.e., the PDEs solved by an actual numerical model, we can instead make use of a generalized system of ODEs that represents only those of particular concern, and their continuous evolution, through which the properties of interest may be verified. Similar abstraction techniques based on compositional reasoning are frequently encountered in model checking applications [5]. What we adopt, then, is a *lightweight* approach to formal methods [6], since we do not attempt to verify an entire software system, but instead identify abstractions and formulate models to answer questions that are difficult to resolve through testing alone.

In the remainder of the paper, we describe our verification approach and abstraction methods, and apply them in a case study that involves a parameterization scheme, called the K-profile parameterization (KPP) [7], which is implemented in an open-source library and widely used in global ocean models to parameterize vertical ocean mixing due to turbulent processes. During a recent effort to incorporate a specific version of the KPP scheme in MOM6 [8], a global ocean model, unrealistic physical behavior was encountered due to a faulty configuration within the KPP scheme. We describe hybrid verification models in KeYmaera X that reproduce the fault and then verify a fix that now appears in production software. We then discuss the applicability of our approach and its relationship to other work, and conclude with some general observations.

## II. Background

### A. Hybrid systems modeling and KeYmaera X

KeYmaera X is an axiomatic theorem prover for verifying safety and liveness properties of hybrid systems [3]. The tool interprets a hybrid verification model and proves properties of interest, sometimes automatically, though manual intervention may be required when pre-defined tactics are incapable of completing a proof. Hybrid verification models in KeYmaera X are stated and verified using a first-order dynamic logic, called *differential dynamic logic*, which incorporates real arithmetic, modal logic operators, discrete and continuous transitions, and control structures [9].

The notation used in KeYmaera X provides a compositional semantics for expressing discrete and continuous transitions as a hybrid program $\alpha$. A differential dynamic logic formula of the form $\phi \rightarrow [\alpha]\psi$, for instance, means that hybrid program $\alpha$, if it begins in a state satisfying formula $\phi$, always satisfies the formula $\psi$, where $\rightarrow$ is logical implication, and $[\cdot]$ is the modal logic operator "always."

Figure 1 shows a formula that models one-dimensional diffusion over a discrete space consisting of three nodes, evenly spaced at $\Delta x = 1$, and with associated scalar variables $\lambda_i$ that are taken to be continuous. It states that the formula $\lambda_1 > (\lambda_0 + \lambda_2)/2$, which appears as both the initial condition and postcondition, is invariant: if it is initially satisfied it will remain satisfied under all executions of the hybrid program in square brackets ($[\,]$).

$$\begin{array}{ll}
\lambda_1 > (\lambda_0 + \lambda_2)/2 \rightarrow & \textit{initial condition} \\
\quad \big[\{ & \textit{execution begins} \\
\qquad \kappa := *; & \textit{discrete assignment} \\
\qquad \{\lambda_1' = \kappa(\lambda_0 - 2\lambda_1 + \lambda_2)\} & \textit{continuous evolution} \\
\quad \}^* \big] & \textit{loop or terminate} \\
\lambda_1 > (\lambda_0 + \lambda_2)/2 & \textit{postcondition}
\end{array}$$

Fig. 1. A differential dynamic logic formula that models continuous, one-dimensional diffusion over a discrete space.

The hybrid program uses curly brackets ($\{\}$) to indicate grouping, and the conventional syntax $a; b$ to indicate the sequential composition of hybrid programs $a$ and $b$. The program includes within it a diffusivity variable $\kappa$ that is updated in a discrete manner: a statement $x := e$ assigns the value of $e$ to variable $x$, and $x := *$ nondeterministically assigns any real value to $x$. The variable $\lambda_1$, in contrast, evolves continuously in time: a statement $\{x' = f(x) \,\&\, Q\}$ defines a differential equation $x' = f(x)$ that evolves within the region described by formula $Q$, if given, for any duration. The values of variables $\lambda_0$ and $\lambda_2$ are set nondeterministically.

An expression $\{a\}^*$ denotes nondeterministic repetition: it repeats program $a$ zero or more times. Intuitively, the code block enclosed by the outer curly brackets in Figure 1 corresponds to a timestepping loop that is executed an arbitrary number of times. At each step, a discrete and nondeterministic

assignment to $\kappa$ is followed by the continuous evolution of the ODE $\lambda_1' = \kappa(\lambda_0 - 2\lambda_1 + \lambda_2)$. When this continuous evolution pauses, the hybrid program may repeat the execution or may terminate.

Unless explicitly stated via an evolution domain constraint $Q$, an ODE in a hybrid program is allowed to evolve for a nondeterministic duration, i.e., for any duration $\Delta t \geq 0$. When modeling an advancing timestep that appears in a numerical model, this behavior leads to an overapproximation, since timesteps in an actual numerical model are constrained by the CFL condition and computational efficiency considerations. However, timesteps of arbitrary sizes will result in a more general model and therefore stronger proofs since any discrete update as the system evolves is part of the analysis.

While formulas as simple as the one in Figure 1 can be proved automatically, more sophisticated ones may require manual user interaction to complete the proof. Custom proof tactics can be developed in KeYmaera X via a Hilbert-style deductive system. As basic building blocks for developing them, KeYmaera X provides a collection of fundamental axioms, sequent calculus proof rules for loop invariants, a real arithmetic solver, substitutions, symbolic solutions of systems of ODEs, and others. Moreover, KeYmaera X provides powerful differential proof rules, such as differential invariants, that allow users to reason about differential equations without having to solve them. In an iterative manner, users can employ these building blocks or let KeYmaera X attempt to find a proof tactic automatically. A full description of the approach can be found in the KeYmaera X documentation [3], [9], [10].

### B. Global Ocean Modeling

Earth system models, such as CESM [11], that couple atmosphere, ocean, land, ice, and other components, are used to simulate the past and future of Earth's climate. Global ocean models included in such coupled systems solve the three dimensional primitive equations describing ocean dynamics to determine time histories of the prognostic variables, that is, the pressure, density, potential temperature, and salinity, and the velocity components in three dimensions. The primitive equations are discretized in time and space using finite difference approximations. Figure 2 shows an example of a coarse-resolution ocean model grid. The grid is also discretized in the vertical dimension, where the number of vertical levels varies from 3 in the shallowest regions to 60 in the deepest, and the grid cell thicknesses gradually increase as the depth increases.

As for time discretization, the POP2 model [13], for example, adopts a three-time-level leapfrog scheme for stepping forward in time. At every timestep, the discretized primitive continuity equations are solved to obtain values of the prognostic variables, and the KPP scheme, described in the next section, is executed to obtain boundary layer depth and diffusivities within the ocean boundary layer.

It is typical in geophysical models to carry out the computations on staggered grids where different prognostic variables reside on different points within a grid cell. Figure 3 shows the placement of prognostic ocean variables on a staggered grid

Fig. 2. A coarse-resolution global ocean grid [12].

called the Arakawa B grid. The diffusivities computed by the KPP scheme, not shown in the figure, are placed on the top and bottom interfaces of grid cells, like the vertical velocity component $w$.

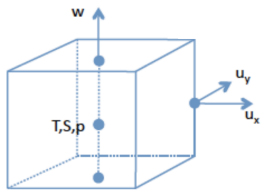


Fig. 3. Placement of prognostic quantities in a POP2 grid cell, where $T$, $S$, and $p$ are temperature, salinity, and pressure, respectively, $u_x$ and $u_y$ are horizontal velocity components, and $w$ is the vertical velocity component [13].

### C. The KPP scheme

The cell sizes of workhorse ocean model grids used in most climate models are on the order $1^\circ \times 1^\circ$ horizontally. The cell thicknesses are about 10 meters near the ocean surface and 250 meters near the ocean bottom in the deepest regions. These horizontal and vertical grid resolutions are insufficient to resolve all processes of concern, so subgrid-scale processes are incorporated in these models via various parameterization approaches. One example of such subgrid-scale processes is ocean mixing due to vertical turbulent fluxes in the boundary layer.

The continuous evolution of a scalar quantity $\lambda$, such as temperature or salinity, over a vertical water column is formulated as follows:

$$\frac{\partial \overline{\lambda}}{\partial t} = \frac{\partial}{\partial z}(\overline{w'\lambda'} + \overline{w}\,\overline{\lambda})$$

where $w$ is the vertical velocity, and so $\overline{w}\,\overline{\lambda}$ is the vertical flux of $\lambda$ resolved by the model and is computed as an advective process, whereas $\overline{w'\lambda'}$ is the unresolved (subgrid-scale) turbulent vertical flux, which is parameterized in the numerical model as a diffusive process [7]:

$$\overline{w'\lambda'} = -K_\lambda(\frac{\partial\overline{\lambda}}{\partial z} + \gamma_\lambda)$$

where $K_\lambda$ is the diffusivity and $\gamma_\lambda$ is the nonlocal transport term for the scalar $\lambda$. The diffusivity $K_\lambda$ is the primary variable of concern of the test case presented in this paper, whereas $\gamma_\lambda$ is out of scope and is taken to be zero.

The CVMix library [7], which implements the KPP scheme, is an open-source library for parameterizing vertical turbulent mixing in ocean models. In addition to the KPP scheme, the CVMix library provides parameterizations for various other mixing processes. Among these processes, the KPP scheme is the one that brings about the greatest algorithmic complexity, and it is our focus in this study as it illustrates both continuous and instantaneous behavior.

Unlike the remaining parameterizations in CVMix that prescribe vertical turbulent mixing in the ocean interior, the KPP scheme is concerned with the vertical mixing only within the ocean boundary layer (OBL) that resides above the ocean interior and below the ocean-atmosphere boundary. The OBL is where the ocean is directly influenced by the atmospheric processes and solar radiation. Therefore, its treatment is highly critical for coupled climate models. At every timestep, the KPP scheme first computes the depth of the OBL, and then computes $K_\lambda$, i.e., the diffusivities at the interfaces of cells within a water column. Figure 4 shows a time history of the global mean of computed OBL depths over a 300-year period from a recent CESM simulation, and Figure 5 shows spatially the 20-year mean OBL depths computed between the model years 281 and 300 of the same simulation.

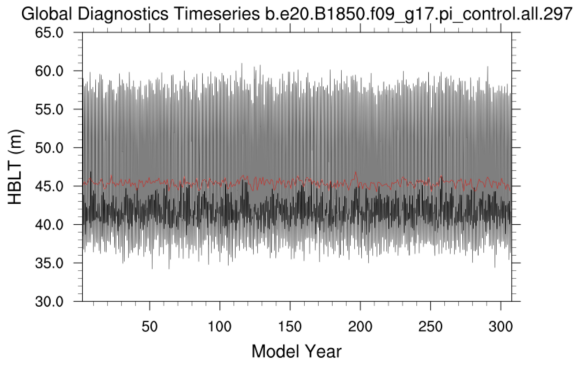


Fig. 4. Time history of the global average of computed OBL depths. The black line corresponds to the monthly mean, and the red line corresponds to the yearly mean over the entire globe.

The formula for the diffusivity $K_\lambda$ on a cell interface at depth $d$ within the OBL is given by [14]:

$$K_\lambda = h \cdot w_\lambda(\sigma) \cdot G_\lambda(\sigma)$$

where $h$ is the OBL depth, $w_\lambda$ is the vertical turbulent velocity scale, whose details are omitted for brevity, $G$ is a cubic shape function, and $\sigma$ is the normalized depth within the OBL, which

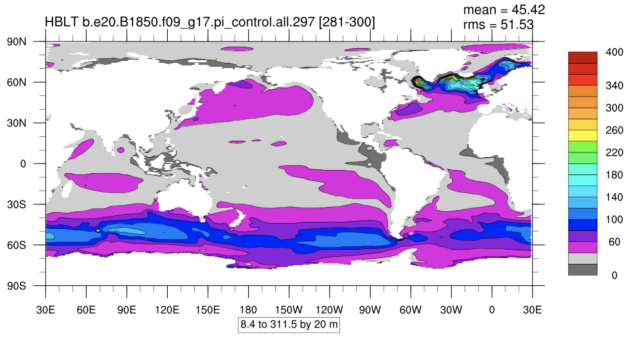


Fig. 5. 20-year mean OBL depths between model years 281 and 300.

is set to $d/h$, and so is 0 at the ocean surface and 1 at the OBL base. The vertical shape function $G$ has the following form [14]:

$$G_\lambda(\sigma) = a_0 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3$$

The coefficients $a_0$ and $a_1$ are set to 0 and 1, respectively, due to constraints arising from the surface boundary conditions (at $\sigma = 0$). The remaining two coefficients are determined according to conditions at the OBL base (at $\sigma = 1$). In order to ensure a smooth transition from the OBL to the ocean interior, the KPP scheme matches the diffusivities and their gradients to the diffusivities and the gradients computed by the other mixing schemes of CVMix, i.e., the interior mixing schemes, at $\sigma = 1$. For convective (unstable) forcing conditions, the matching is accomplished by setting $a_2$ and $a_3$ as follows [14]:

$$a_2 = -2 + 3\,\nu/(h \cdot w_\lambda(1)) - \partial_z\nu/w_\lambda(1)$$
$$a_3 = 1 - 2\,\nu/(h \cdot w_\lambda(1)) + \partial_z\nu/w_\lambda(1)$$

where $\nu$ and $\partial_z\nu$ are the diffusivities and their gradients, respectively, at the OBL base computed by the interior mixing schemes.

## III. Test Case: The KPP scheme and its matching algorithm

Developers of the ocean component of CESM have recently incorporated a specific version of the KPP scheme into MOM6, the ocean component for future versions of CESM. While testing this newly incorporated mixing scheme, they encountered unrealistic physical behavior: negative diffusivities in regions subject to convective forcing, i.e., surface cooling.[1] After testing and debugging attempts that took several days, the source was found to be in the configuration of the matching algorithm of the KPP scheme. In this section, we develop a hybrid verification model of the KPP scheme and detect this

[1]In regions where the thermal expansion coefficient is positive, surface cooling increases the density and so reduces the buoyancy of surface water, which results in convection. This type of surface forcing is called *unstable buoyancy forcing,* a process that deepens the ocean boundary layer and enhances vertical mixing.

behavior. We then apply a fix in the matching algorithm and confirm that it eliminates it.

### A. Modeling discrete spaces

In practice, grids on which earth system models operate contain millions of cells and span the entire globe. If we are to verify properties of the simulation then clearly the approach must be sound for grids of arbitrary size. Hybrid verification models, however, may contain only a limited number of variables before they become intractable. We therefore make use of small, abstract grids in our models and employ nondeterminism as needed to draw general conclusions.

In line with this perspective, we model a single water column in our hybrid model, as shown in Figure 6 (a). The depth $D$ of the water column is nondeterministically assigned an arbitrary positive value, and the distance $z_w$ from the ocean bottom to a cell interface, the only one explicitly represented, is nondeterministically assigned a value between 0 and $D$. The water column abstraction is intended to model an arbitrary water column within the ocean model, together with an arbitrary cell interface within the water column. We observe with respect to the KPP scheme that both discrete and continuous processes over a water column are independent of the states of other water columns. Therefore, a property shown to hold on this interface holds on any cell interface within a grid of arbitrary size.

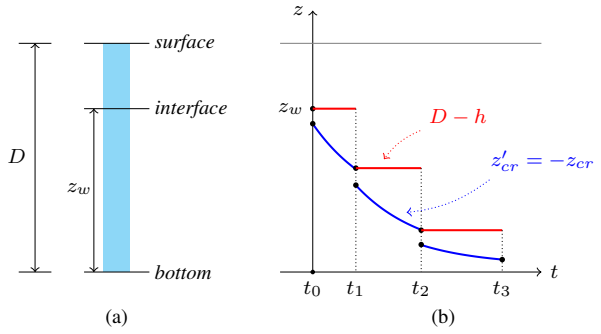


Fig. 6. Abstract representation of a grid cell interface incident on an arbitrary water column, where $D$ is the depth of the water column, $z_w$ is the distance between the ocean bottom and the cell interface, $z_{cr}$ is the distance between the ocean bottom ($z = 0$) and the critical depth, and $h$ is the boundary layer depth. While $D$ and $z_w$ are fixed (yet nondeterministically chosen) coordinates, $z_{cr}$ and $h$ change over time $t$.

### B. The KeYmaera X model of the KPP scheme

Following the abstraction approach laid out in the introduction, we now develop a KeYmaera X model of the KPP scheme. The main body of the model, shown in Figure 7, begins with an antecedent that defines initial conditions, which nondeterministically set abstract grid properties and the state of the interface. The hybrid program then follows with timestepping computations, enclosed by the outer curly brackets, that are repeated an arbitrary number of times. Finally, to ensure realistic physical behavior, the postcondition $K > 0$ asserts that diffusivities at the interface $z = z_w$ must be positive valued: any execution sequence that violates it corresponds to a counterexample that may arise in the actual numerical model.

```
initialConditions() →
  [{
    computeBLD;        // discrete updates
    computeNu;
    computeK;
    {z'_cr = −z_cr}    // continuous system
  }*]
K > 0
```

Fig. 7. Main body of the KeYmaera X model of the KPP scheme. The native KeYmaera X syntax is occasionally replaced with mathematical symbols in this figure and in the remainder of this paper for readability.

The timestepping computations are once again divided into discrete and continuous parts. First, discrete updates contained in the hybrid programs *computeBLD*, *computeNu*, and *computeK* exert instantaneous changes on program variables. Then, the continuous system is expressed as an ODE, $z'_{cr} = -z_{cr}$, where $z'_{cr}$ denotes the time derivative of the abstract variable $z_{cr}$. A timestep begins with the discrete updates and continues on with the evolution of the ODE for an arbitrary duration. Once it pauses, the program either terminates or advances to the next timestep. The continuous system and discrete updates are described below, in turn.

*a) Continuous system:* We first examine the continuous constituent of the hybrid program, which describes the time evolution of the abstract variable $z_{cr}$, the depth at which a physical property called the Bulk Richardson number is equal to a predetermined critical value. In our hybrid program, $z_{cr}$ is a manifestation of the continuous processes, e.g., the evolution of the buoyancy profile determined by the temperature and salinity at each level, the vertical shear, and the ocean surface forcing conditions. By incorporating only the continuous evolution of $z_{cr}$ as an exponential decay, as shown in Figure 6 (b), we abstract from these complex quantities and their evolutions. This abstraction, i.e., the exponential decay of $z_{cr}$, is in agreement with actual model results when the water column is subject to convective forcing.

Another variable, whose evolution is shown in Figure 6 (b), is the OBL depth, $h$, which is updated discretely at the beginning of every timestep by setting its value to $D - z_{cr}$ within the program named *computeBLD*. While the abstract variable $z_{cr}$ may be thought of as the embodiment of the underlying continuous processes left out of the verification model, e.g., the evolution of the temperature, salinity, surface forcing, and shear, the variable $h$ is the *discrete* OBL depth that is computed by sampling $z_{cr}$ intermittently. The initial values of $z_{cr}$ and $h$ are set to $z_w$ and $D - z_w$, respectively, via the *initialConditions* predicate. Note that the continuous evolution of $z_{cr}$ depends not only on the aforementioned continuous processes but also on $h$. Therefore, every time

$h$ is updated discretely, $z_{cr}$ is updated accordingly, resulting in discontinuities in the continuous evolution of $z_{cr}$ at the beginning of each timestep.

*b) Discrete updates:* The definition of program *computeK*,[2] shown below, enforces the matching condition by first computing shape function coefficients $a_2$ and $a_3$ and then computing the diffusivity variable $K$ at the cell interface:

$$
\begin{aligned}
&\textit{HP computeK} ::= \{ \\
&\quad a_2 := -2 + 3 * \nu/(h * w()) + \partial\nu/w(); \\
&\quad a_3 := 1 - \nu/(h * w()) - \partial\nu/w(); \\
&\quad K := h * w() * G(\sigma, a_2, a_3); \\
&\}.
\end{aligned}
$$

where $h$ is the discrete OBL depth determined by *computeBLD*, which samples the continuous variable $z_{cr}$, $\nu$ and $\partial\nu$ are variables set by *computeNu* that correspond to the interior diffusivity and the gradient of the interior diffusivity at the base of the OBL, respectively, $w$ is a function that returns the vertical turbulent velocity scale value at the interface $z = z_w$, and $G$ is the value of the shape function at $z = z_w$. We omit full definitions of the *initialConditions* predicate and the programs *computeBLD* and *computeNu* for brevity. However, the complete hybrid model is available in an online repository [15].

### C. The KeYmaera X proof tactic of the KPP scheme

Having developed a hybrid verification model of the KPP scheme, we manually construct a custom proof tactic using the KeYmaera X user interface. In this paper, we describe some of the key steps in the process, and again refer the interested reader to our online repository [15] for further details of the proof tactics employed.

The first step in constructing a custom proof tactic for our program is to identify an invariant for the timestepping loop, which will be needed to make the proofs go through. The inference rule in KeYmaera X is shown below [16]:

$$\frac{\Gamma \vdash J, \Delta \qquad J \vdash [\alpha]J \qquad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

where we seek an invariant $J$ for loop $\alpha^*$ that is true initially ($\Gamma \vdash J, \Delta$), is preserved by the loop body ($J \vdash [\alpha]J$), and entails the postcondition ($J \vdash P$), thereby allowing us to conclude that $\Gamma \vdash [\alpha^*]P, \Delta$ holds.

The rule's conclusion matches the main body of the model, shown in Figure 7, in the obvious way: $\Gamma$ corresponds to the *initialConditions* predicate, which sets the abstract grid properties and the state of the interface, $\alpha$ is the body of the timestep loop combining discrete updates and the continuous system, and $P$ is the postcondition $K > 0$.

We try a loop invariant that is the conjunction of the postcondition $K > 0$ and an intuitive bound on the continuous evolution, $0 < z_{cr} \leq z_w$. The first and the third sequents of the rule's premise are easily shown to hold. The third, for example, reduces to $K > 0 \land 0 < z_{cr} \leq z_w \vdash K > 0$, which is sound. Proving the second sequent, $J \vdash [\alpha]J$, is more involved. After a series of fundamental axioms and sequent calculus proof rules are introduced, we end up with two sequents as our new premise:

$$
\begin{aligned}
\Theta \vdash [z'_{cr} = -z_{cr}]\,(0 < z_{cr} \land z_{cr} \leq z_w) \\
\land\ \Theta \vdash [z'_{cr} = -z_{cr}]\,(\mathcal{K} > 0)
\end{aligned}
\tag{1}
$$

where $\Theta$ is the conjunction of the initial conditions $\Gamma$ and additional expressions resulting from substituting and moving the discrete updates to the antecedent using axioms and rules such as the composition, assignment, and implication rules, and $\mathcal{K}$ is a real arithmetic expression obtained by similarly substituting the associated variables, excluding $z_{cr}$, in the formula of the diffusivity $\kappa$. Elimination of $z_{cr}$ from the formula $\mathcal{K}$ is ultimately made possible by the loop invariant.

The first sequent of the premise is where the proof tactic development process leads us to prove that the continuous evaluation of $z_{cr}$ is indeed bounded by the domain constraint prescribed in the loop invariant. Proving it justifies the substitutions and assumptions made to obtain the second sequent of the premise, and so is critical. First, we make use of the *boxAnd* tactic [17] to separate the conjunct into two components, $0 < z_{cr}$ and $z_{cr} \leq z_w$ . To prove the first, i.e., $\Theta \vdash [z'_{cr} = -z_{cr}]\,(0 < z_{cr})$, for example, we make use of the differential ghost proof rule where we add to the system an auxiliary continuous variable $y$ and a postcondition $y^2 z_{cr} = 1$. Since the new postcondition entails $z_{cr} > 0$, proving it suffices to conclude that the sequent in question is closed soundly. For that, we make use of the differential invariant proof rule which successfully closes the branch, where we (deliberately) define the continuous evolution of the auxiliary variable as $y' = 0.5y$. The differential invariant proof rule allows modelers to reason about ODEs without having to solve them and instead by proving that a formula $p$ always holds after arbitrary evolutions of the ODEs based on their right-hand sides, e.g., $-z_{cr}$ and $0.5y$ [18].

After similarly proving the second component of the first conjunct of Equation 1, we conclude that the assumptions made to obtain the second conjunct, i.e., $\Theta \vdash [z'_{cr} = -z_{cr}]\,(\mathcal{K} > 0)$, are sound. To prove that this is sound as well, we eliminate the ODE using the differential weakening proof rule, since the postcondition $\mathcal{K} > 0$ does not involve the continuous variable $z_{cr}$, and so its correctness is not affected by the ODE. We then make use of the real arithmetic operator in an attempt to close this final branch of the proof tactic. Doing so leads KeYmaera X to conclude that the premise evaluates to *false*, which indicates that there exists an execution sequence leading to non-positive diffusivities at the cell interface $z = z_w$.

### D. The fixed matching condition

Upon inspection of the counterexample generated by KeYmaera X, we conclude that negative diffusivities occur due to the matching condition that matches the diffusivities $\kappa$

[2]The syntax *HP inc* ::= $\{x := x + 1\}$ defines a program abbreviation *inc* that is expanded to $x := x + 1$ everywhere it appears, similar to an inline function.

and $\nu$ and their gradients, $\partial\kappa$ and $\partial\nu$, at $z = D - h$, as described in Section II-C. It is observed in the counterexample that when the OBL base falls well below $z_w$, and when the interior diffusivities have a negative and steep gradient $\partial\nu$, the matching of the gradients forces $\kappa$ at the cell interface $z = z_w$ to be negative.

To prevent the case where the diffusivities in the OBL may be negative, we modify the computation of the coefficients $a_2$ and $a_3$ so that when the interior diffusivity gradient $\partial\nu$ is negative at the OBL base, only the diffusivities $\kappa$ and $\nu$ are matched by the scheme and not their gradients. The modified *computeK* definition is as follows, where $?Q$ is the conditional operator that instructs KeYmaera X to take into account the execution sequences of a particular conditional branch only when $Q$ is true.

$$
\begin{aligned}
&\textit{HP computeK} ::= \{ \\
&\quad \{ \ ?\, \partial\nu < 0; \\
&\qquad\quad a_2 := -2 + 3 * \nu/(h * w()); \\
&\qquad\quad a_3 := 1 - \nu/(h * w()); \\
&\quad \cup \\
&\quad ?\, \partial\nu \geq 0; \\
&\qquad\quad a_2 := -2 + 3 * \nu/(h * w()) + \partial\nu/w(); \\
&\qquad\quad a_3 := 1 - \nu/(h * w()) - \partial\nu/w(); \\
&\quad \} \\
&\quad K := h * w() * G(\sigma, a_2, a_3); \\
&\}.
\end{aligned}
$$

Following again the development process summarized in Section III-C, we are now able to prove that the postcondition $K > 0$ always holds for any possible execution of discrete updates to the continuous system, for an arbitrary number of timesteps.

Both the discovery of the problem we address, and the application of the fix, were made prior to performing this study, but because testing is incomplete, the model presented here was developed to provide additional coverage and to increase confidence that a corner case is not being missed. A comparison of the relative effort between the two activities is worth noting. With respect to numerical simulation and testing, the debugging process took several days and thousands of core-hours. An initial version of the KeYmaera X model, on the other hand, took less than a day to put together and consists of about 50 lines of code. The theorem proving process for the final version of the model took about ten to fifteen minutes to perform, a majority of which was spent providing user input to steer the proof tactic.

## IV. Discussion and Related Work

The verification approach outlined in this paper can be viewed as a lightweight approach to formal methods since there is *partiality in modeling:* we target specific properties that can be checked with the aid of verification tools from the field of cyber-physical systems. Here we consider such an approach in the context of other possible concerns, and compare it with several other approaches in the literature that address different aspects of program correctness.

*a) Numerical concerns:* Our primary focus is on the correctness of discrete, instantaneous updates in the solution of PDEs, essentially extracting and working with a system of piecewise continuous ODEs. In doing so, we recognize that the corresponding numerical model is merely an approximation of the analytical solution, whose quality depends on timestep size and various other numerical issues, including truncation errors.

Abstracting from them yields an approach that does a lot of heavy lifting. The test coverage possible in the temporal dimension far exceeds what might be obtained in a verification approach using finite-difference based time discretizations and symbolic model checking, for instance. Doing so would require, we suspect, large finite state models and implausibly long traces.[3]

In addition, it may be possible to address some numerical concerns in hybrid verification models by altering the continuous system in a way that reflects the behavior of the numerical model. For example, if numerical dispersion is of concern, the ODE or its solution may be altered to exhibit the dispersive behavior.

*b) Extracting models from code:* Another aspect of the approach outlined here is that we manually extract verification models from code, but automated approaches have the potential to ensure that the checks being performed are sound and, because they are automated, to make formal verification approaches more accessible to a wider community.

Siegel et al. [19], for instance, present a framework that tests small numerical programs for *real equivalence*, meaning that one program can be transformed into the other using the identities of real numbers. The approach is based on creating a sequential program that serves as a specification and then using it as the measure against which an implementation is compared, such as a more complex MPI-based parallel program. Equivalence checking is performed by building up symbolic expressions in both programs and comparing them using the SPIN model checker.

Nevertheless, modeling would seem to confer the same kind of benefits for software, a complex artifact, as it does throughout the sciences and engineering. Instead of attempting to verify large-scale numerical codebases *en masse*, one instead addresses the particular sources of complexity and areas of concern using appropriate tools. And because scientists and engineers are accustomed to working with models, there is some potential for acceptance, particularly as the tools mature and become easier to use.

*c) State-based formal methods:* The structure and behavior of scientific programs constitute a kind of complexity that goes beyond just numerical concerns. Using state-based methods [20], software is described by what constitutes a state and how and when transitions are performed, often at a much higher level than code. By employing notions like predicate abstraction and abstraction refinement, one can

[3]Though such an approach might offer complementary benefits.

separate concerns and apply such approaches to numerical software and its fundamental structures of data and control.

As an example, in prior work [21] we look at an ocean circulation model used in production and an extension made to it that offers substantial performance gains. To explore implementation choices and to ensure soundness of the extension, we use Alloy, which has tool support and an automatic form of analysis performed within a bounded scope using a SAT solver. In another study [22], we present a model checking approach for verifying the types of concurrency found in coupled earth models. By modeling read-write behavior and the timestamps associated with updates, race-free phasing arrangements can be generated, thereby preventing data from either being overwritten too soon or becoming stale.

## V. Conclusions

The computational cost of climate simulations is substantial: a single run may require millions of core-hours and produce terabytes of data to analyze. Debugging such programs through testing alone, therefore, poses its own set of practical challenges. Based on sampling, testing is well-known to be incomplete, motivating the development of new, supplemental approaches that can offer some assurance in the absence of exhaustive testing.

In this study, we describe our experience with a hybrid theorem prover as a lightweight verification tool for reasoning about discrete state changes in numerical software. Our abstraction approach is based on representing the time evolution of physical processes as piecewise continuous ODEs, and incorporating those discrete aspects in hybrid programs, similar to conventional computer programs. The approach is demonstrated in the context of large-scale numerical software, a coupled earth system model used in production, and applied to a newly incorporated mixing scheme to overcome some of the limitations associated with testing. As such, it represents a novel use of a tool designed for a different application domain, cyber-physical systems, together with a new modeling approach that views scientific software as a hybrid system.

We consider the application of such a tool in a relatively uncharted domain, scientific computation, where there is little community experience in working with formal methods. Our approach is lightweight in the sense that there is partiality in modeling: it is directed toward particular aspects of numerical software. We imagine that a single approach is unlikely to meet every need, and that developers will instead benefit from having multiple tools in their toolboxes, so they can choose the right one for the job.

## Acknowledgment

We thank Bill Large for his comments on our continuous and discrete representation of the KPP scheme. We also thank Gustavo Marques for his efforts in debugging the KPP matching configuration in MOM6. We acknowledge computing support from Information Systems Group of the Climate and Global Dynamics Laboratory at NCAR, sponsored by the National Science Foundation and other agencies.

## References

[1] T. Storer, "Bridging the chasm: A survey of software engineering practice in scientific programming," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 47:1–47:32, 2017.

[2] L. Doyen, G. Frehse, G. J. Pappas, and A. Platzer, "Verification of hybrid systems," in *Handbook of Model Checking*. Springer, 2018, pp. 1047–1110.

[3] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *International Conference on Automated Deduction*. Springer, 2015, pp. 527–538.

[4] A. Zafarullah, "Application of the method of lines to parabolic partial differential equations with error estimates," *Journal of the ACM*, vol. 17, no. 2, pp. 294–302, Apr. 1970.

[5] S. Berezin, S. Campos, and E. M. Clarke, "Compositional reasoning in model checking," in *Compositionality: The Significant Difference*. Springer, 1998, pp. 81–102.

[6] D. Jackson and J. Wing, "Lightweight formal methods," *IEEE Computer*, vol. 29, pp. 22–23, 1996.

[7] S. M. Griffies, M. Levy, A. J. Adcroft, G. Danabasoglu, R. W. Hallberg, D. Jacobsen, W. Large, and T. Ringler, "Theory and numerics of the community ocean vertical mixing (CVMIX) project," [Retrieved from https://github.com/CVMix/CVMix-description/blob/master/cvmix.pdf], Tech. Rep., 2015.

[8] S. M. Griffies, "Elements of the Modular Ocean Model (MOM)," NOAA/Geophysical Fluid Dynamics Laboratory, Tech. Rep. 7, GFDL Ocean Group, 2012.

[9] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[10] ——, "A complete uniform substitution calculus for differential dynamic logic," *Journal of Automated Reasoning*, vol. 59, no. 2, pp. 219–265, 2017.

[11] J. W. Hurrell, M. M. Holland, P. R. Gent, S. Ghan, J. E. Kay, P. J. Kushner, J.-F. Lamarque, W. G. Large, D. Lawrence, K. Lindsay *et al.*, "The community earth system model: a framework for collaborative research," *Bulletin of the American Meteorological Society*, vol. 94, no. 9, pp. 1339–1360, 2013.

[12] M. Vertenstein, T. Craig, A. Middleton, D. Feddema, and C. Fischer, "CESM1. 0.4 user's guide," The National Center for Atmospheric Research, Tech. Rep., 2011.

[13] R. Smith, P. Jones, B. Briegleb, F. Bryan, G. Danabasoglu, J. Dennis, J. Dukowicz, C. Eden, B. Fox-Kemper, P. Gent *et al.*, "The parallel ocean program (POP) reference manual ocean component of the community climate system model (CCSM) and community earth system model (CESM)," *Rep. LAUR-01853*, vol. 141, pp. 1–140, 2010.

[14] W. G. Large, J. C. McWilliams, and S. C. Doney, "Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization," *Reviews of Geophysics*, vol. 32, no. 4, pp. 363–403, 1994.

[15] A. Altuntas, "KeYmaera X models and proof tactics of the KPP scheme," https://github.com/alperaltuntas/keymaera-kpp, 2018.

[16] A. Platzer, *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

[17] N. Fulton, S. Mitsch, B. Bohrer, and A. Platzer, "Bellerophon: Tactical theorem proving for hybrid systems," in *International Conference on Interactive Theorem Proving*. Springer, 2017, pp. 207–224.

[18] A. Platzer, "A uniform substitution calculus for differential dynamic logic," in *International Conference on Automated Deduction*. Springer, 2015, pp. 467–481.

[19] S. F. Siegel, A. Mironova, G. S. Avrunin, and L. A. Clarke, "Combining symbolic execution with model checking to verify parallel numerical programs," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 17, no. 2, pp. 10:1–10:34, 2008.

[20] J. Baugh and T. Dyer, "State-based formal methods in scientific computation," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 6th International Conference, ABZ 2018*. Springer, 2018, pp. 392–396, Lecture Notes in Computer Science 10817.

[21] J. Baugh and A. Altuntas, "Formal methods and finite element analysis of hurricane storm surge: A case study in software verification," *Science of Computer Programming*, vol. 158, pp. 100–121, 2018.

[22] A. Altuntas and J. Baugh, "Verifying concurrency in an adaptive ocean circulation model," in *Proceedings of the First International Workshop on Software Correctness for HPC Applications, Correctness'17*. ACM, 2017, pp. 1–7.