A time manager for MPAS

MPAS Development Team

February 18, 2011

Contents

Chapter 1

Summary

In the current MPAS code, time is simply tracked as the number of time steps since the beginning of the simulation; with the time step length also known, the code can multiply the time step length by the number of steps taken to find the elapsed time. However, because there is no calendar date and time associated with the start of the model simulation, there is no concept of 'real time' in a simulation — that is, a simulation has no way of distinguishing whether it is currently, say, January or July.

When performing real-data simulations of the atmosphere (and perhaps of other physical domains, too), certain physics schemes must know the calendar date and time at each step in the model integration. For example, a short-wave radiation scheme needs to know the solar zenith angle, which it computes from the date and time of day. To someone looking at model output, it would also be important to know the date and time for which a particular set of fields is valid; without such information, validating model simulations becomes nearly impossible.

Besides lacking the date and time at any point in a model simulation, the current MPAS time-keeping method provides no unified mechanism for identifying when arbitrarily defined points in the simulation have been reached, at which time some action may be taken. For example, the work of determining whether it is time to perform I/O currently falls on the driver code, which uses the current time step modulo some output interval specified in time steps; adjustments to the model time step are therefore usually accompanied by changes to the output interval. Using a sophisticated, future I/O subsystem in MPAS, we may like to perform I/O on multiple streams (e.g., files) at different intervals specified in natural time units rather than model time steps; in such cases, the simple-minded approach that is currently used becomes cumbersome, especially if the future I/O implementation is to be run-time configurable in terms of the number of streams.

To permit progress on the addition of physics schemes and the re-implementation of the MPAS I/O sub-system in a more general and robust way, this document proposes the addition of a new time management system, which will be general enough to be used throughout the MPAS code. This new system should be useful for tracking the date and time, providing notifications when pre-specified points in time have been reached, and tracking the progress through time of a model simulation (i.e., model time-stepping).

Chapter 2

Requirements

Date last modified: 2011/02/18
Contributors: Michael Duda, Phil Jones

2.1 Requirement: General date/time tracking

The most fundamental requirement for the new MPAS time management system is the ability to track the current date/time through the course of a model simulation.

2.1.1 Set the starting and ending date/time

Recognizing that there is always a finite amount of wallclock time that can be devoted to any simulation, it follows that every simulation is associated with a starting and ending *simulation* date/time. The MPAS time manager is required to maintain these two points explicitly, since their relation to the current simulation time determines whether a simulation has completed or not. In the case of 'cycled' simulations, which in principle could be extended indefinitely given an infinite amount of computer time, the ending date/time represents the end of the simulation cycle currently being run. It must be possible for the user to reset the starting or ending time at any point during the model simulation.

2.1.2 Determine whether the starting or ending time has been reached

Determining whether the ending time has been reached or exceeded is a requirement for knowing when integration should stop in MPAS. For integration backward in time, the same requirement holds, except the role of the ending time is fulfilled by the starting time.

2.1.3 Set the default time increment

An MPAS simulation generally makes use of a fixed, default time step length, and the time manager must be able to record this time step so that the date/time can be incremented without the need to specify the increment explicitly each time step. It must also be possible for the user to reset the default time step at any point during the model simulation. The user must be able to specify this default time increment in a number of ways, including specifying a time interval (in any supported unit - see Units), specifying the number of steps in a given interval (e.g. steps per day), or internally computing a time step based on resolution (useful for benchmarking mode).

2.1.4 Advance the date/time by an arbitrary time increment

In future, we may like to implement the ability in MPAS to vary the time step over the course of a simulation, according to criteria such as the maximum Courant number in the previous time step; therefore, the time manager must be able to advance the date/time by an arbitrary time increment in addition to incrementing by the default time step.

2.1.5 Handle both forward and backward time increments

The ability to 'advance' the date/time by a negative time increment — that is, to step backward through time — may be useful for further model development. For example, the addition of a digital filtering initialization scheme would require the ability to integrate both forward and backward in time, and, therefore, to increment the date/time with both positive and negative increments. The MPAS time manager must support both forward and backward increments.

2.1.6 Perform time computation with no round-off using exact arithmetic

The ability of the time manager to track time using exact arithmetic is necessary so that, for example, events take place at the proper time, and not at the time step before or after due to rounding errors in the date/time computation. Simulations of up to 100,000 years without roundoff accumulation are required.

2.2 Requirement: Notifications

Upcoming developments in MPAS will require the ability to determine when pre-specified points in the model simulation have been reached. For example, we may want to update tendencies from physics parameterizations periodically at an interval longer than the time step. Another planned development that would require the ability to perform actions at pre-determined points is in the I/O system, where we may want to read (in the case of boundary conditions, for example) or write fields to any of several I/O streams periodically. In general, we require the time manager to have the ability to provide notifications to code when specified points in time have been reached.

2.2.1 Add notifications

Clearly, the ability of the user to specify the points in time where notifications are to be provided is required of the time manager. It must be possible to add notifications at any point in program execution, so that not all notifications need to be added at initialization time.

2.2.2 One-time notification

The time manager will support notifications that only ring once for a particular time.

2.2.3 Recurring notification

The time manager must support the setting of notifications which will recur with a fixed period, since some events (e.g., boundary updates, diagnostics, I/O) occur periodically.

2.2.4 Query for notifications

The user must be able to query whether a notification or alarm is "ringing". (Previous text here specified a bit too much of the design).

2.2.5 Clear a notification

In order that a particular notification not be provided by the time manager in subsequent queries, it must be possible to clear a notification. Clearing an instance of a periodically recurring notification will only clear that particular instance.

2.2.6 Remove a notification

For both periodic and 'one-off' notifications, it must be possible to remove those notifications from the time manager to prevent them from being provided in future queries for notifications.

2.3 Time Intervals

2.3.1 Adding time intervals

The user must be able to specify other time intervals. These are often required for time-averaging and other diagnostics.

2.3.2 Computing time intervals

Given two time instances, the time manager must be able to compute the elapsed time between them in any supported unit. Also, as above for the default time increment (time step), the user may wish to compute an interval that can enforce commensurate time intervals (e.g. an interval that will guarantee a number of steps within a specified interval).

2.3.3 Incrementing time intervals

The user should be able to increment alternative time intervals.

2.4 Calendars and Units

The time manager must be able to track time in a variety of units and calendars.

2.4.1 Units

The time manager must support time intervals and time instants in units of hours, minutes, seconds, days, months and years (and fractions thereof). The underlying representation can be in arbitrary units (to support the roundoff requirements above), but user queries and arguments must support these units.

2.4.2 Unit Conversion

A set of utilities must be available to convert units for time-related quantities (e.g. days to seconds, etc.). Some conversions may require knowledge of the calendar choice.

2.4.3 Calendars

The time manager must, at minimum, support a Gregorian calendar. Other calendars that may need to be supported are 360-day (30-day equal month) calendars or Julian Day calendars.

2.4.4 Leap Years

The user must be able to specify whether to turn on/off leap years. For example, no-leap-year is needed for normal year or other climatological forcing simulations while other forcing choices will need leap years.

2.4.5 Mid-month computation

Often, for ocean simulations, monthly forcing is used and is applied (or interpolated) based on the middle of the month. This will be calendar dependent.

2.5 Formatting

For input, output and tagging, the time manager should support dates or intervals in standard formats like yyyy-mm-dd or hhhh.mm.ss.

2.6 Real time

Mark had requirements for interacting with a real time and querying system time, but that might be better captured in a different doc that would include timers or other system queries?

Chapter 3

Design

3.1 Design Solution: XXX

Date last modified: 2011/01/05
Contributors: (add your name to this list if it does not appear)

TBD

Chapter 4

Proposed Plan for Implementation

4.1 Implementation: XXX

Date last modified: 2011/01/05
Contributors: (add your name to this list if it does not appear)

TBD

Chapter 5

Testing and Validation

5.1 Testing and Validation: XXX

Date last modified: 2011/01/05
Contributors: (add your name to this list if it does not appear)

TBD