# CICE Documentation

CICE Consortium

Aug 15, 2019

# Contents

CHAPTER 1

# Introduction - CICE

## 1.1 About CICE

CICE is a computationally efficient model for simulating the growth, melting, and movement of polar sea ice. Designed as one component of coupled atmosphere-ocean-land-ice global climate models, today's CICE model is the outcome of more than two decades of effort led by scientists at Los Alamos National Laboratory. The current version of the model has been enhanced greatly through collaborations with members of the community.

CICE has several interacting components: a model of ice dynamics, which predicts the velocity field of the ice pack based on a model of the material strength of the ice; a transport model that describes advection of the areal concentration, ice volumes and other state variables; and a vertical physics package, called "Icepack", which includes mechanical, thermodynamic, and biogeochemical models to compute thickness changes and the internal evolution of the hydrological ice-brine ecosystem. When coupled with other earth system model components, routines external to the CICE model prepare and execute data exchanges with an external "flux coupler".

Icepack is implemented in CICE as a git submodule, and it is documented at https://cice-consortium-icepack.readthedocs.io/en/master/index.html. Development and testing of CICE and Icepack may be done together, but the repositories are independent. This document describes the remainder of the CICE model. The CICE code is available from https://github.com/CICE-Consortium/CICE.

The standard standalone CICE test configuration uses a 3 degree grid with atmospheric data from 1997, available at https://github.com/CICE-Consortium/CICE/wiki/CICE-Input-Data. A 1-degree configuration and data are also available, along with some idealized configurations. The data files are designed only for testing the code, not for use in production runs or as observational data. Please do not publish results based on these data sets.

The CICE model can run serially or in parallel, and the CICE software package includes tests for various configurations. MPI is used for message passing between processors, and OpenMP threading is available.

Major changes with each CICE release (https://github.com/CICE-Consortium/CICE/releases) will be detailed with the included release notes. Enhancements and bug fixes made to CICE since the last numbered release can be found on the CICE wiki (https://github.com/CICE-Consortium/CICE/wiki/CICE-Recent-changes). **Please cite any use of the CICE code.** More information can be found at *Citing the CICE code*.

This document uses the following text conventions: Variable names used in the code are `typewritten`. Subroutine names are given in *italic*. File and directory names are in **boldface**. A comprehensive *Index of primary variables and*

*parameters*, including glossary of symbols with many of their values, appears at the end of this guide.

## 1.2 Quick Start

**Download the model from the CICE-Consortium repository,** https://github.com/CICE-Consortium/CICE

Instructions for working in github with CICE (and Icepack) can be found in the CICE Git and Workflow Guide.

You will probably have to download some inputdata, see the CICE wiki or *Forcing data*.

Software requirements are noted in this *Software Requirements* section.

From your main CICE directory, execute:

```
./cice.setup -c ~/mycase1 -g gx3 -m testmachine -s diag1,thread -p 8x1
cd ~/mycase1
./cice.build
./cice.submit
```

`testmachine` is a generic machine name included with the cice scripts. The local machine name will have to be substituted for `testmachine` and there are working ports for several different machines. However, it may be necessary to port the model to a new machine. See *Porting* for more information about how to port. See *Scripts* for more information about how to use the cice.setup and cice.submit scripts.

Please cite any use of the CICE code. More information can be found at *Citing the CICE code*.

## 1.3 Acknowledgements

This work has been completed through the CICE Consortium and its members with funding through the

- Department of Energy (Los Alamos National Laboratory)
- Department of Defense (Navy)
- Department of Commerce (National Oceanic and Atmospheric Administration)
- National Science Foundation (the National Center for Atmospheric Research)
- Environment and Climate Change Canada.

Special thanks are due to participants from these institutions and many others who contributed to previous versions of CICE or Icepack.

## 1.4 Citing the CICE code

If you use the CICE code, please cite the version you are using with the CICE Digital Object Identifier (DOI):

DOI:10.5281/zenodo.1205674 (https://zenodo.org/record/1205674)

This DOI can be used to cite all CICE versions and the URL will default to the most recent version. However, each released version of CICE will also receive its own, unique DOI that can be used for citations as well.

Please also make the CICE Consortium aware of any publications and model use.

## 1.5 Copyright

# CHAPTER 2

# Science Guide

## 2.1 Coupling With Other Climate Model Components

The sea ice model exchanges information with the other model components via a flux coupler. CICE has been coupled into numerous climate models with a variety of coupling techniques. This document is oriented primarily toward the CESM Flux Coupler [22] from NCAR, the first major climate model to incorporate CICE. The flux coupler was originally intended to gather state variables from the component models, compute fluxes at the model interfaces, and return these fluxes to the component models for use in the next integration period, maintaining conservation of momentum, heat, and fresh water. However, several of these fluxes are now computed in the ice model itself and provided to the flux coupler for distribution to the other components, for two reasons. First, some of the fluxes depend strongly on the state of the ice, and vice versa, implying that an implicit, simultaneous determination of the ice state and the surface fluxes is necessary for consistency and stability. Second, given the various ice types in a single grid cell, it is more efficient for the ice model to determine the net ice characteristics of the grid cell and provide the resulting fluxes, rather than passing several values of the state variables for each cell. These considerations are explained in more detail below.

The fluxes and state variables passed between the sea ice model and the CESM flux coupler are listed in the Icepack documentation. By convention, directional fluxes are positive downward. In CESM, the sea ice model may exchange coupling fluxes using a different grid than the computational grid. This functionality is activated using the namelist variable `gridcpl_file`. Another namelist variable `highfreq`, allows the high-frequency coupling procedure implemented in the Regional Arctic System Model (RASM). In particular, the relative atmosphere-ice velocity ($\vec{U}_a - \vec{u}$) is used instead of the full atmospheric velocity for computing turbulent fluxes in the atmospheric boundary layer.

The ice fraction $a_i$ (aice) is the total fractional ice coverage of a grid cell. That is, in each cell,

$$\begin{array}{rl} a_i = 0 & \text{if there is no ice} \\ a_i = 1 & \text{if there is no open water} \\ 0 < a_i < 1 & \text{if there is both ice and open water,} \end{array}$$

where $a_i$ is the sum of fractional ice areas for each category of ice. The ice fraction is used by the flux coupler to merge fluxes from the ice model with fluxes from the other components. For example, the penetrating shortwave radiation flux, weighted by $a_i$, is combined with the net shortwave radiation flux through ice-free leads, weighted by $(1 - a_i)$, to obtain the net shortwave flux into the ocean over the entire grid cell. The flux coupler requires the fluxes to be divided

by the total ice area so that the ice and land models are treated identically (land also may occupy less than 100% of an atmospheric grid cell). These fluxes are "per unit ice area" rather than "per unit grid cell area."

For CICE run in stand-alone mode (i.e., uncoupled), the AOMIP shortwave and longwave radiation formulas are available in **ice_forcing.F90**. In function *longwave_rosati_miyakoda*, downwelling longwave is computed as

$$F_{lw\downarrow} = \epsilon\sigma T_s^4 - \epsilon\sigma T_a^4(0.39 - 0.05e_a^{1/2})(1 - 0.8f_{cld}) - 4\epsilon\sigma T_a^3(T_s - T_a) \tag{2.1}$$

where the atmospheric vapor pressure (mb) is $e_a = 1000Q_a/(0.622 + 0.378Q_a)$, $\epsilon = 0.97$ is the ocean emissivity, $\sigma$ is the Stephan-Boltzman constant, $f_{cld}$ is the cloud cover fraction, and $T_a$ is the surface air temperature (K). The first term on the right is upwelling longwave due to the mean (merged) ice and ocean surface temperature, $T_s$ (K), and the other terms on the right represent the net longwave radiation patterned after [38].

The downwelling longwave formula of [34] is also available in function *longwave_parkinson_washington*:

$$F_{lw\downarrow} = \epsilon\sigma T_a^4(1 - 0.261\exp\left(-7.77 \times 10^{-4}T_a^2\right)(1 + 0.275f_{cld}) \tag{2.2}$$

The value of $F_{lw\uparrow}$ is different for each ice thickness category, while $F_{lw\downarrow}$ depends on the mean value of surface temperature averaged over all of the thickness categories and open water. The merged ice-ocean temperature in this formula creates a feedback between longwave radiation and sea surface temperature which is unrealistic, resulting in erroneous model sensitivities to radiative changes, e.g. other emissivity values, when run in the stand-alone mode. Although our stand-alone model test configurations are useful for model development purposes, we strongly recommend that scientific conclusions be drawn using the model only when coupled with other earth system components.

The AOMIP shortwave forcing formula (in subroutine *compute_shortwave*) incorporates the cloud fraction and humidity through the atmospheric vapor pressure:

$$F_{sw\downarrow} = \frac{1353\cos^2 Z}{10^{-3}(\cos Z + 2.7)e_a + 1.085\cos Z + 0.1}\left(1 - 0.6f_{cld}^3\right) > 0 \tag{2.3}$$

where $\cos Z$ is the cosine of the solar zenith angle.

Many ice models compute the sea surface slope $\nabla H_\circ$ from geostrophic ocean currents provided by an ocean model or other data source. In our case, the sea surface height $H_\circ$ is a prognostic variable in POP—the flux coupler can provide the surface slope directly, rather than inferring it from the currents. (The option of computing it from the currents is provided in subroutine *dyn_prep2*.) The sea ice model uses the surface layer currents $\vec{U}_w$ to determine the stress between the ocean and the ice, and subsequently the ice velocity $\vec{u}$. This stress, relative to the ice,

$$\vec{\tau}_w = \quad c_w\rho_w\left|\vec{U}_w - \vec{u}\right|\left[\left(\vec{U}_w - \vec{u}\right)\cos\theta + \hat{k} \times \left(\vec{U}_w - \vec{u}\right)\sin\theta\right] \tag{2.4}$$

is then passed to the flux coupler (relative to the ocean) for use by the ocean model. Here, $\theta$ is the turning angle between geostrophic and surface currents, $c_w$ is the ocean drag coefficient, $\rho_w$ is the density of seawater, and $\hat{k}$ is the vertical unit vector. The turning angle is necessary if the top ocean model layers are not able to resolve the Ekman spiral in the boundary layer. If the top layer is sufficiently thin compared to the typical depth of the Ekman spiral, then $\theta = 0$ is a good approximation. Here we assume that the top layer is thin enough.

Please see the Icepack documentation for additional information about atmospheric and oceanic forcing and other data exchanged between the flux coupler and the sea ice model.

## 2.2 Fundamental Variables

The Arctic and Antarctic sea ice packs are mixtures of open water, thin first-year ice, thicker multiyear ice, and thick pressure ridges. The thermodynamic and dynamic properties of the ice pack depend on how much ice lies in each thickness range. Thus the basic problem in sea ice modeling is to describe the evolution of the ice thickness distribution (ITD) in time and space.

The fundamental equation solved by CICE is [44]:

$$\frac{\partial g}{\partial t} = -\nabla \cdot (g\mathbf{u}) - \frac{\partial}{\partial h}(fg) + \psi, \tag{2.5}$$

where $\mathbf{u}$ is the horizontal ice velocity, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$, $f$ is the rate of thermodynamic ice growth, $\psi$ is a ridging redistribution function, and $g$ is the ice thickness distribution function. We define $g(\mathbf{x}, h, t)\, dh$ as the fractional area covered by ice in the thickness range $(h, h + dh)$ at a given time and location.

Additional information about the ITD for CICE can be found in the Icepack documentation.

In addition to the fractional ice area, $a_{in}$, we define the following state variables for each category $n$. In a change from previous CICE versions, we no longer carry snow and ice energy as separate variables; instead they and sea ice salinity are carried as tracers on snow and ice volume.

- $v_{in}$, the ice volume, equal to the product of $a_{in}$ and the ice thickness $h_{in}$.
- $v_{sn}$, the snow volume, equal to the product of $a_{in}$ and the snow thickness $h_{sn}$.
- $e_{ink}$, the internal ice energy in layer $k$, equal to the product of the ice layer volume, $v_{in}/N_i$, and the ice layer enthalpy, $q_{ink}$. Here $N_i$ is the total number of ice layers, with a default value $N_i = 4$, and $q_{ink}$ is the negative of the energy needed to melt a unit volume of ice and raise its temperature to 0 °C. (NOTE: In the current code, $e_i < 0$ and $q_i < 0$ with $e_i = v_i q_i$.)
- $e_{snk}$, the internal snow energy in layer $k$, equal to the product of the snow layer volume, $v_{sn}/N_s$, and the snow layer enthalpy, $q_{snk}$, where $N_s$ is the number of snow layers. (Similarly, $e_s < 0$ in the code.) CICE allows multiple snow layers, but the default value is $N_s = 1$.
- $S_i$, the bulk sea ice salt content in layer $k$, equal to the product of the ice layer volume and the sea ice salinity tracer.
- $T_{sfn}$, the surface temperature.

Since the fractional area is unitless, the volume variables have units of meters (i.e., $m^3$ of ice or snow per $m^2$ of grid cell area), and the energy variables have units of J/$m^2$.

The three terms on the right-hand side of Equation (2.5) describe three kinds of sea ice transport: (1) horizontal transport in $(x, y)$ space; (2) transport in thickness space $h$ due to thermodynamic growth and melting; and (3) transport in thickness space $h$ due to ridging and other mechanical processes. We solve the equation by operator splitting in three stages, with two of the three terms on the right set to zero in each stage. We compute horizontal transport using the incremental remapping scheme of [7] as adapted for sea ice by [28]; this scheme is discussed in Section *Horizontal Transport*. Ice is transported in thickness space using the remapping scheme of [27]. The mechanical redistribution scheme, based on [44], [39], [12], [8], and [29] is outlined in the Icepack Documentation. To solve the horizontal transport and ridging equations, we need the ice velocity $\mathbf{u}$, and to compute transport in thickness space, we must know the the ice growth rate $f$ in each thickness category. We use the elastic-viscous-plastic (EVP) ice dynamics scheme of [16], as modified by [5], [15], [17] and [18], or a new elastic-anisotropic-plastic model [49][47][45] to find the velocity, as described in Section *Dynamics*. Finally, we use a thermodynamic model to compute $f$. The order in which these computations are performed in the code itself was chosen so that quantities sent to the coupler are consistent with each other and as up-to-date as possible. The Delta-Eddington radiative scheme computes albedo and shortwave components simultaneously, and in order to have the most up-to-date values available for the coupler at the end of the timestep, the order of radiation calculations is shifted. Albedo and shortwave components are computed after the ice state has been modified by both thermodynamics and dynamics, so that they are consistent with the ice area and thickness at the end of the step when sent to the coupler. However, they are computed using the downwelling shortwave from the beginning of the timestep. Rather than recompute the albedo and shortwave components at the beginning of the next timestep using new values of the downwelling shortwave forcing, the shortwave components computed at the end of the last timestep are scaled for the new forcing.

## 2.3 Tracers

The basic conservation equations for ice area fraction $a_{in}$, ice volume $v_{in}$, and snow volume $v_{sn}$ for each thickness category $n$ are

$$\frac{\partial}{\partial t}(a_{in}) + \nabla \cdot (a_{in}\mathbf{u}) = 0, \tag{2.6}$$

$$\frac{\partial v_{in}}{\partial t} + \nabla \cdot (v_{in}\mathbf{u}) = 0, \tag{2.7}$$

$$\frac{\partial v_{sn}}{\partial t} + \nabla \cdot (v_{sn}\mathbf{u}) = 0. \tag{2.8}$$

The ice and snow volumes can be written equivalently in terms of tracers, ice thickness $h_{in}$ and snow depth $h_{sn}$:

$$\frac{\partial h_{in}a_{in}}{\partial t} + \nabla \cdot (h_{in}a_{in}\mathbf{u}) = 0, \tag{2.9}$$

$$\frac{\partial h_{sn}a_{in}}{\partial t} + \nabla \cdot (h_{sn}a_{in}\mathbf{u}) = 0. \tag{2.10}$$

Although we maintain ice and snow volume instead of the thicknesses as state variables in CICE, the tracer form is used for volume transport (section *Horizontal Transport*). There are many other tracers available, whose values are contained in the `trcrn` array. Their transport equations typically have one of the following three forms

$$\frac{\partial (a_{in}T_n)}{\partial t} + \nabla \cdot (a_{in}T_n\mathbf{u}) = 0, \tag{2.11}$$

$$\frac{\partial (v_{in}T_n)}{\partial t} + \nabla \cdot (v_{in}T_n\mathbf{u}) = 0, \tag{2.12}$$

$$\frac{\partial (v_{sn}T_n)}{\partial t} + \nabla \cdot (v_{sn}T_n\mathbf{u}) = 0. \tag{2.13}$$

Equation (2.11) describes the transport of surface temperature, whereas Equation (2.12) and Equation (2.13) describe the transport of ice and snow enthalpy, salt, and passive tracers such as volume-weighted ice age and snow age. Each tracer field is given an integer index, `trcr_depend`, which has the value 0, 1, or 2 depending on whether the appropriate conservation equation is Equation (2.11), Equation (2.12), or Equation (2.13), respectively. The total number of tracers is $N_{tr} \geq 1$. Table *Tracers* provides an overview of available tracers, including the namelist flags that turn them on and off, and their indices in the tracer arrays. If any of the three explicit pond schemes is on, then `tr_pond` is true. Biogeochemistry tracers can be defined in the skeletal layer, dependent on the ice area fraction, or through the full depth of snow and ice, in which case they utilize the bio grid and can depend on the brine fraction or the ice volume, if the brine fraction is not in use.

Table 1: *Tracer flags and indices*

| flag | num tracers | dependency | index (CICE grid) | index (bio grid) |
|---|---|---|---|---|
| default | 1 | aice | nt_Tsfc=1 | |
| default | 1 | vice | nt_qice | |
| default | 1 | vsno | nt_qsno | |
| default | 1 | vice | nt_sice | |
| tr_iage | 1 | vice | nt_iage | |
| tr_FY | 1 | aice | nt_FY | |
| tr_lvl | 2 | aice | nt_alvl | |
| | | vice | nt_vlvl | |
| tr_pond_cesm | 2 | aice | nt_apnd | |
| | | apnd | nt_vpnd | |

Continued on next page

Table 1 – continued from previous page

| flag | num tracers | dependency | index (CICE grid) | index (bio grid) |
|---|---|---|---|---|
| tr_pond_lvl | 3 | aice | nt_apnd | |
| | | apnd | nt_vpnd | |
| | | apnd | nt_ipnd | |
| tr_pond_topo | 3 | aice | nt_apnd | |
| | | apnd | nt_vpnd | |
| | | apnd | nt_ipnd | |
| tr_aero | n_aero | vice, vsno | nt_aero | |
| tr_brine | | vice | nt_fbri | |
| solve_zsal | n_trzs | fbri or (a,v)ice | nt_bgc_S | |
| tr_bgc_N | n_algae | fbri or (a,v)ice | nt_bgc_N | nlt_bgc_N |
| tr_bgc_Nit | | fbri or (a,v)ice | nt_bgc_Nit | nlt_bgc_Nit |
| tr_bgc_C | n_doc | fbri or (a,v)ice | nt_bgc_DOC | nlt_bgc_DOC |
| | n_dic | fbri or (a,v)ice | nt_bgc_DIC | nlt_bgc_DIC |
| tr_bgc_chl | n_algae | fbri or (a,v)ice | nt_bgc_chl | nlt_bgc_chl |
| tr_bgc_Am | | fbri or (a,v)ice | nt_bgc_Am | nlt_bgc_Am |
| tr_bgc_Sil | | fbri or (a,v)ice | nt_bgc_Sil | nlt_bgc_Sil |
| tr_bgc_DMS | | fbri or (a,v)ice | nt_bgc_DMSPp | nlt_bgc_DMSPd |
| | | fbri or (a,v)ice | nt_bgc_DMSPd | nlt_bgc_DMSPd |
| | | fbri or (a,v)ice | nt_bgc_DMS | nlt_bgc_DMS |
| tr_bgc_PON | | fbri or (a,v)ice | nt_bgc_PON | nlt_bgc_PON |
| tr_bgc_DON | | fbri or (a,v)ice | nt_bgc_DON | nlt_bgc_DON |
| tr_bgc_Fe | n_fed | fbri or (a,v)ice | nt_bgc_Fed | nlt_bgc_Fed |
| | n_fep | fbri or (a,v)ice | nt_bgc_Fep | nlt_bgc_Fep |
| tr_bgc_hum | | fbri or (a,v)ice | nt_bgc_hum | nlt_bgc_hum |
| tr_zaero | n_zaero | fbri or (a,v)ice | nt_zaero | nlt_zaero |
| | 1 | fbri | nt_zbgc_frac | |

Users may add any number of additional tracers that are transported conservatively, provided that the dependency `trcr_depend` is defined appropriately. See Section *Adding Tracers* for guidance on adding tracers.

Please see the Icepack documentation for additional information about tracers that depend on other tracers, age of the ice, aerosols, brine height, and the sea ice ecosystem.

## 2.4 Horizontal Transport

We wish to solve the continuity or transport equation (Equation (2.6)) for the fractional ice area in each thickness category $n$. Equation (2.6) describes the conservation of ice area under horizontal transport. It is obtained from Equation (2.5) by discretizing $g$ and neglecting the second and third terms on the right-hand side, which are treated separately (As described in the Icepack Documentation).

There are similar conservation equations for ice volume (Equation (2.7)), snow volume (Equation (2.8)), ice energy and snow energy:

$$\frac{\partial e_{ink}}{\partial t} + \nabla \cdot (e_{ink}\mathbf{u}) = 0, \tag{2.14}$$

$$\frac{\partial e_{snk}}{\partial t} + \nabla \cdot (e_{snk}\mathbf{u}) = 0. \tag{2.15}$$

By default, ice and snow are assumed to have constant densities, so that volume conservation is equivalent to mass conservation. Variable-density ice and snow layers can be transported conservatively by defining tracers corresponding to ice and snow density, as explained in the introductory comments in **ice_transport_remap.F90**. Prognostic equations for ice and/or snow density may be included in future model versions but have not yet been implemented.

Two transport schemes are available: upwind and the incremental remapping scheme of [7] as modified for sea ice by [28]. The remapping scheme has several desirable features:

- It conserves the quantity being transported (area, volume, or energy).
- It is non-oscillatory; that is, it does not create spurious ripples in the transported fields.
- It preserves tracer monotonicity. That is, it does not create new extrema in the thickness and enthalpy fields; the values at time $m+1$ are bounded by the values at time $m$.
- It is second-order accurate in space and therefore is much less diffusive than first-order schemes (e.g., upwind). The accuracy may be reduced locally to first order to preserve monotonicity.
- It is efficient for large numbers of categories or tracers. Much of the work is geometrical and is performed only once per grid cell instead of being repeated for each quantity being transported.

The time step is limited by the requirement that trajectories projected backward from grid cell corners are confined to the four surrounding cells; this is what is meant by incremental remapping as opposed to general remapping. This requirement leads to a CFL-like condition,

$$\frac{\max|\mathbf{u}|\Delta t}{\Delta x} \leq 1.$$

For highly divergent velocity fields the maximum time step must be reduced by a factor of two to ensure that trajectories do not cross. However, ice velocity fields in climate models usually have small divergences per time step relative to the grid size.

The remapping algorithm can be summarized as follows:

1. Given mean values of the ice area and tracer fields in each grid cell, construct linear approximations of these fields. Limit the field gradients to preserve monotonicity.
2. Given ice velocities at grid cell corners, identify departure regions for the fluxes across each cell edge. Divide these departure regions into triangles and compute the coordinates of the triangle vertices.
3. Integrate the area and tracer fields over the departure triangles to obtain the area, volume, and energy transported across each cell edge.
4. Given these transports, update the state variables.

Since all scalar fields are transported by the same velocity field, step (2) is done only once per time step. The other three steps are repeated for each field in each thickness category. These steps are described below.

After the transport calculation, the sum of ice and open water areas within a grid cell may not add up to 1. The mechanical deformation parameterization in Icepack corrects this issue by ridging the ice and creating open water such that the ice and open water areas again add up to 1.

### 2.4.1 Reconstructing area and tracer fields

First, using the known values of the state variables, the ice area and tracer fields are reconstructed in each grid cell as linear functions of $x$ and $y$. For each field we compute the value at the cell center (i.e., at the origin of a 2D Cartesian coordinate system defined for that grid cell), along with gradients in the $x$ and $y$ directions. The gradients are limited to preserve monotonicity. When integrated over a grid cell, the reconstructed fields must have mean values equal to the known state variables, denoted by $\bar{a}$ for fractional area, $\tilde{h}$ for thickness, and $\hat{q}$ for enthalpy. The mean values are not, in general, equal to the values at the cell center. For example, the mean ice area must equal the value at the centroid, which may not lie at the cell center.

Consider first the fractional ice area, the analog to fluid density $\rho$ in [7]. For each thickness category we construct a field $a(\mathbf{r})$ whose mean is $\bar{a}$, where $\mathbf{r} = (x, y)$ is the position vector relative to the cell center. That is, we require

$$\int_A a\, dA = \bar{a}\, A, \tag{2.16}$$

where $A = \int_A dA$ is the grid cell area. Equation (2.16) is satisfied if $a(\mathbf{r})$ has the form

$$a(\mathbf{r}) = \bar{a} + \alpha_a \langle \nabla a \rangle \cdot (\mathbf{r} - \bar{\mathbf{r}}), \tag{2.17}$$

where $\langle \nabla a \rangle$ is a centered estimate of the area gradient within the cell, $\alpha_a$ is a limiting coefficient that enforces monotonicity, and $\bar{\mathbf{r}}$ is the cell centroid:

$$\bar{\mathbf{r}} = \frac{1}{A} \int_A \mathbf{r}\, dA.$$

It follows from Equation (2.17) that the ice area at the cell center ($\mathbf{r} = 0$) is

$$a_c = \bar{a} - a_x \overline{x} - a_y \overline{y},$$

where $a_x = \alpha_a(\partial a/\partial x)$ and $a_y = \alpha_a(\partial a/\partial y)$ are the limited gradients in the $x$ and $y$ directions, respectively, and the components of $\bar{\mathbf{r}}$, $\overline{x} = \int_A x\, dA/A$ and $\overline{y} = \int_A y\, dA/A$, are evaluated using the triangle integration formulas described in Section *Integrating fields*. These means, along with higher-order means such as $\overline{x^2}$, $\overline{xy}$, and $\overline{y^2}$, are computed once and stored.

Next consider the ice and snow thickness and enthalpy fields. Thickness is analogous to the tracer concentration $T$ in [7], but there is no analog in [7] to the enthalpy. The reconstructed ice or snow thickness $h(\mathbf{r})$ and enthalpy $q(\mathbf{r})$ must satisfy

$$\int_A a\, h\, dA = \bar{a}\, \tilde{h}\, A, \tag{2.18}$$

$$\int_A a\, h\, q\, dA = \bar{a}\, \tilde{h}\, \hat{q}\, A, \tag{2.19}$$

where $\tilde{h} = h(\tilde{\mathbf{r}})$ is the thickness at the center of ice area, and $\hat{q} = q(\hat{\mathbf{r}})$ is the enthalpy at the center of ice or snow volume. Equations (2.18) and (2.19) are satisfied when $h(\mathbf{r})$ and $q(\mathbf{r})$ are given by

$$h(\mathbf{r}) = \tilde{h} + \alpha_h \langle \nabla h \rangle \cdot (\mathbf{r} - \tilde{\mathbf{r}}), \tag{2.20}$$

$$q(\mathbf{r}) = \hat{q} + \alpha_q \langle \nabla q \rangle \cdot (\mathbf{r} - \hat{\mathbf{r}}), \tag{2.21}$$

where $\alpha_h$ and $\alpha_q$ are limiting coefficients. The center of ice area, $\tilde{\mathbf{r}}$, and the center of ice or snow volume, $\hat{\mathbf{r}}$, are given by

$$\tilde{\mathbf{r}} = \frac{1}{\bar{a}\, A} \int_A a\, \mathbf{r}\, dA,$$

$$\hat{\mathbf{r}} = \frac{1}{\bar{a}\, \tilde{h}\, A} \int_A a\, h\, \mathbf{r}\, dA.$$

Evaluating the integrals, we find that the components of $\tilde{\mathbf{r}}$ are

$$\tilde{x} = \frac{a_c \overline{x} + a_x \overline{x^2} + a_y \overline{xy}}{\bar{a}},$$

$$\tilde{y} = \frac{a_c \overline{y} + a_x \overline{xy} + a_y \overline{y^2}}{\bar{a}},$$

and the components of $\hat{\mathbf{r}}$ are

$$\hat{x} = \frac{c_1 \overline{x} + c_2 \overline{x^2} + c_3 \overline{xy} + c_4 \overline{x^3} + c_5 \overline{x^2 y} + c_6 \overline{xy^2}}{\bar{a}\, \tilde{h}},$$

$$\hat{y} = \frac{c_1 \overline{y} + c_2 \overline{xy} + c_3 \overline{y^2} + c_4 \overline{x^2 y} + c_5 \overline{xy^2} + c_6 \overline{y^3}}{\bar{a}\, \tilde{h}},$$

where

$$\begin{aligned} c_1 &\equiv a_c h_c, \\ c_2 &\equiv a_c h_x + a_x h_c, \\ c_3 &\equiv a_c h_y + a_y h_c, \\ c_4 &\equiv a_x h_x, \\ c_5 &\equiv a_x h_y + a_y h_x, \\ c_6 &\equiv a_y h_y. \end{aligned}$$

From Equation (2.20) and Equation (2.21), the thickness and enthalpy at the cell center are given by

$$h_c = \tilde{h} - h_x \tilde{x} - h_y \tilde{y},$$

$$q_c = \hat{q} - q_x \hat{x} - q_y \hat{y},$$

where $h_x$, $h_y$, $q_x$ and $q_y$ are the limited gradients of thickness and enthalpy. The surface temperature is treated the same way as ice or snow thickness, but it has no associated enthalpy. Tracers obeying conservation equations of the form Equation (2.12) and Equation (2.13) are treated in analogy to ice and snow enthalpy, respectively.

We preserve monotonicity by van Leer limiting. If $\bar{\phi}(i,j)$ denotes the mean value of some field in grid cell $(i,j)$, we first compute centered gradients of $\bar{\phi}$ in the $x$ and $y$ directions, then check whether these gradients give values of $\phi$ within cell $(i,j)$ that lie outside the range of $\bar{\phi}$ in the cell and its eight neighbors. Let $\bar{\phi}_{\max}$ and $\bar{\phi}_{\min}$ be the maximum and minimum values of $\bar{\phi}$ over the cell and its neighbors, and let $\phi_{\max}$ and $\phi_{\min}$ be the maximum and minimum values of the reconstructed $\phi$ within the cell. Since the reconstruction is linear, $\phi_{\max}$ and $\phi_{\min}$ are located at cell corners. If $\phi_{\max} > \bar{\phi}_{\max}$ or $\phi_{\min} < \bar{\phi}_{\min}$, we multiply the unlimited gradient by $\alpha = \min(\alpha_{\max}, \alpha_{\min})$, where

$$\alpha_{\max} = (\bar{\phi}_{\max} - \bar{\phi})/(\phi_{\max} - \bar{\phi}),$$

$$\alpha_{\min} = (\bar{\phi}_{\min} - \bar{\phi})/(\phi_{\min} - \bar{\phi}).$$

Otherwise the gradient need not be limited.

Earlier versions of CICE (through v3.14) computed gradients in physical space. Starting in v4.0, gradients are computed in a scaled space in which each grid cell has sides of unit length. The origin is at the cell center, and the four vertices are located at (0.5, 0.5), (-0.5,0.5),(-0.5, -0.5) and (0.5, -0.5). In this coordinate system, several of the above grid-cell-mean quantities vanish (because they are odd functions of x and/or y), but they have been retained in the code for generality.

### 2.4.2 Locating departure triangles

The method for locating departure triangles is discussed in detail by [7]. The basic idea is illustrated in *Departure Region*, which shows a shaded quadrilateral departure region whose contents are transported to the target or home grid cell, labeled $H$. The neighboring grid cells are labeled by compass directions: $NW$, $N$, $NE$, $W$, and $E$. The four vectors point along the velocity field at the cell corners, and the departure region is formed by joining the starting points of these vectors. Instead of integrating over the entire departure region, it is convenient to compute fluxes across cell edges. We identify departure regions for the north and east edges of each cell, which are also the south and west edges of neighboring cells. Consider the north edge of the home cell, across which there are fluxes from the neighboring $NW$ and $N$ cells. The contributing region from the $NW$ cell is a triangle with vertices $abc$, and that from the $N$ cell is a quadrilateral that can be divided into two triangles with vertices $acd$ and $ade$. Focusing on triangle $abc$, we first determine the coordinates of vertices $b$ and $c$ relative to the cell corner (vertex $a$), using Euclidean geometry to find vertex $c$. Then we translate the three vertices to a coordinate system centered in the $NW$ cell. This translation is needed in order to integrate fields (Section *Integrating fields*) in the coordinate system where they have been reconstructed (Section *Reconstructing area and tracer fields*). Repeating this process for the north and east edges of each grid cell, we compute the vertices of all the departure triangles associated with each cell edge.

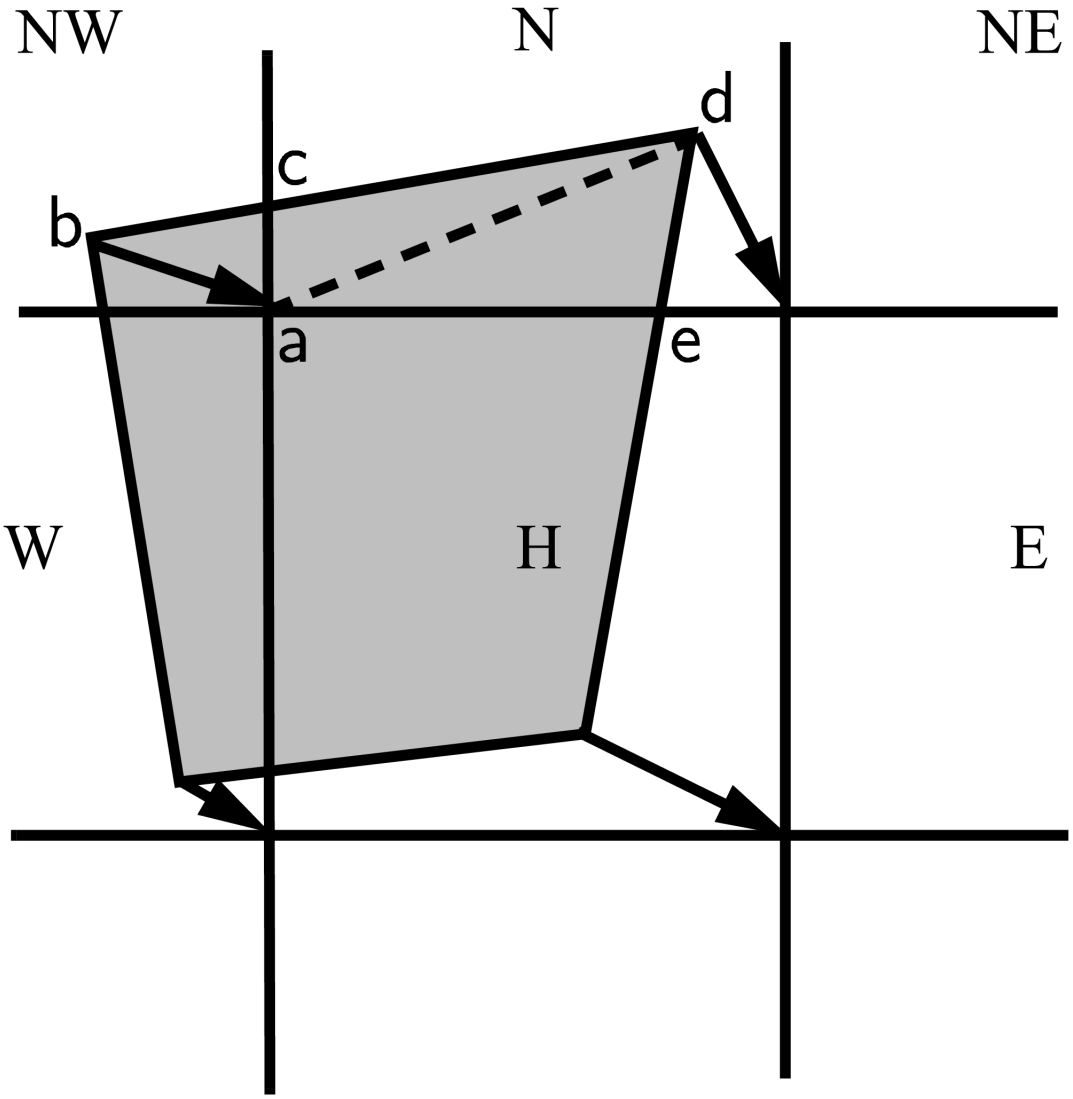


Fig. 1: Departure Region

Figure *Departure Region* shows that in incremental remapping, conserved quantities are remapped from the shaded departure region, a quadrilateral formed by connecting the backward trajectories from the four cell corners, to the grid cell labeled $H$. The region fluxed across the north edge of cell $H$ consists of a triangle ($abc$) in the $NW$ cell and a quadrilateral (two triangles, $acd$ and $ade$) in the $N$ cell.

Figure *Triangles*, reproduced from [7], shows all possible triangles that can contribute fluxes across the north edge of a grid cell. There are 20 triangles, which can be organized into five groups of four mutually exclusive triangles as shown in *Triangular Contributions*. In this table, $(x_1, y_1)$ and $(x_2, y_2)$ are the Cartesian coordinates of the departure points relative to the northwest and northeast cell corners, respectively. The departure points are joined by a straight line that intersects the west edge at $(0, y_a)$ relative to the northwest corner and intersects the east edge at $(0, y_b)$ relative to the northeast corner. The east cell triangles and selecting conditions are identical except for a rotation through 90 degrees.

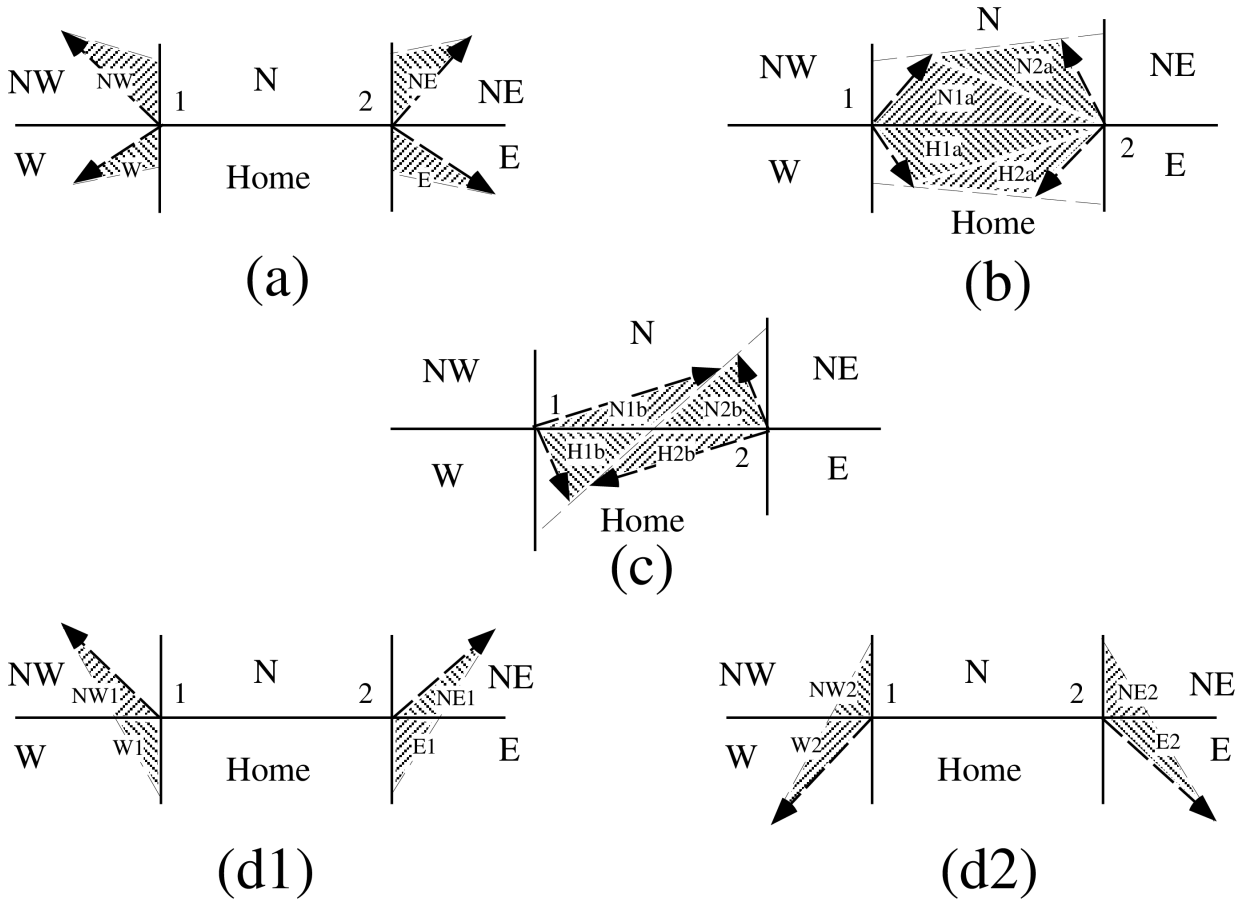


Fig. 2: Triangles

Table *Triangular Contributions* show the evaluation of contributions from the 20 triangles across the north cell edge. The coordinates $x_1$, $x_2$, $y_1$, $y_2$, $y_a$, and $y_b$ are defined in the text. We define $\tilde{y}_1 = y_1$ if $x_1 > 0$, else $\tilde{y}_1 = y_a$. Similarly, $\tilde{y}_2 = y_2$ if $x_2 < 0$, else $\tilde{y}_2 = y_b$.

Table 2: Triangular Contributions

| Triangle group | Triangle label | Selecting logical condition | |
|---|---|---|---|
| 1 | NW | $y_a > 0$ and $y_1 \geq 0$ and $x_1 < 0$ | |
| | NW1 | $y_a < 0$ and $y_1 \geq 0$ and $x_1 < 0$ | |
| | W | $y_a < 0$ and $y_1 < 0$ and $x_1 < 0$ | |
| | W2 | $y_a > 0$ and $y_1 < 0$ and $x_1 < 0$ | |
| | | | |
| 2 | NE | $y_b > 0$ and $y_2 \geq 0$ and $x_2 > 0$ | |
| | NE1 | $y_b < 0$ and $y_2 \geq 0$ and $x_2 > 0$ | |
| | E | $y_b < 0$ and $y_2 < 0$ and $x_2 > 0$ | |
| | E2 | $y_b > 0$ and $y_2 < 0$ and $x_2 > 0$ | |
| | | | |
| 3 | W1 | $y_a < 0$ and $y_1 \geq 0$ and $x_1 < 0$ | |
| | NW2 | $y_a > 0$ and $y_1 < 0$ and $x_1 < 0$ | |
| | E1 | $y_b < 0$ and $y_2 \geq 0$ and $x_2 > 0$ | |
| | NE2 | $y_b > 0$ and $y_2 < 0$ and $x_2 > 0$ | |
| | | | |
| 4 | H1a | $y_a y_b \geq 0$ and $y_a + y_b < 0$ | |
| | N1a | $y_a y_b \geq 0$ and $y_a + y_b > 0$ | |
| | H1b | $y_a y_b < 0$ and $\tilde{y}_1 < 0$ | |
| | N1b | $y_a y_b < 0$ and $\tilde{y}_1 > 0$ | |
| | | | |
| 5 | H2a | $y_a y_b \geq 0$ and $y_a + y_b < 0$ | |
| | N2a | $y_a y_b \geq 0$ and $y_a + y_b > 0$ | |
| | H2b | $y_a y_b < 0$ and $\tilde{y}_2 < 0$ | |
| | N2b | $y_a y_b < 0$ and $\tilde{y}_2 > 0$ | |
| | | | |

This scheme was originally designed for rectangular grids. Grid cells in CICE actually lie on the surface of a sphere and must be projected onto a plane. The projection used in CICE maps each grid cell to a square with sides of unit length. Departure triangles across a given cell edge are computed in a coordinate system whose origin lies at the midpoint of the edge and whose vertices are at (-0.5, 0) and (0.5, 0). Intersection points are computed assuming Cartesian geometry with cell edges meeting at right angles. Let CL and CR denote the left and right vertices, which are joined by line CLR. Similarly, let DL and DR denote the departure points, which are joined by line DLR. Also, let IL and IR denote the intersection points (0, $y_a$) and (0, $y_b$) respectively, and let IC = ($x_c$, 0) denote the intersection of CLR and DLR. It can be shown that $y_a$, $y_b$, and $x_c$ are given by

$$\begin{aligned} y_a = & \quad \frac{x_{CL}(y_{DM} - y_{DL}) + x_{DM} y_{DL} - x_{DL} y_{DM}}{x_{DM} - x_{DL},} \\ y_b = & \quad \frac{x_{CR}(y_{DR} - y_{DM}) - x_{DM} y_{DR} + x_{DR} y_{DM}}{x_{DR} - x_{DM},} \\ x_c = & \quad x_{DL} - y_{DL} \left( \frac{x_{DR} - x_{DL}}{y_{DR} - y_{DL}} \right) \end{aligned}$$

Each departure triangle is defined by three of the seven points (CL, CR, DL, DR, IL, IR, IC).

Given a 2D velocity field **u**, the divergence $\nabla \cdot \mathbf{u}$ in a given grid cell can be computed from the local velocities and written in terms of fluxes across each cell edge:

$$\nabla \cdot \mathbf{u} = \frac{1}{A} \left[ \left( \frac{u_{NE} + u_{SE}}{2} \right) L_E + \left( \frac{u_{NW} + u_{SW}}{2} \right) L_W + \left( \frac{u_{NE} + u_{NW}}{2} \right) L_N + \left( \frac{u_{SE} + u_{SW}}{2} \right) L_S \right], \tag{2.22}$$

where $L$ is an edge length and the indices $N, S, E, W$ denote compass directions. Equation (2.22) is equivalent to the divergence computed in the EVP dynamics (Section *Dynamics*). In general, the fluxes in this expression are not equal to those implied by the above scheme for locating departure regions. For some applications it may be desirable to prescribe the divergence by prescribing the area of the departure region for each edge. This can be done in CICE 4.0 by setting *l_fixed_area* = true in **ice_transport_driver.F90** and passing the prescribed departure areas (*edgearea_e* and *edgearea_n*) into the remapping routine. An extra triangle is then constructed for each departure region to ensure that the total area is equal to the prescribed value. This idea was suggested and first implemented by Mats Bentsen of the Nansen Environmental and Remote Sensing Center (Norway), who applied an earlier version of the CICE remapping scheme to an ocean model. The implementation in CICE v4.0 is somewhat more general, allowing for departure regions lying on both sides of a cell edge. The extra triangle is constrained to lie in one but not both of the grid cells that share the edge. Since this option has yet to be fully tested in CICE, the current default is *l_fixed_area* = false.

We made one other change in the scheme of [7] for locating triangles. In their paper, departure points are defined by projecting cell corner velocities directly backward. That is,

$$\mathbf{x_D} = -\mathbf{u}\,\Delta t, \tag{2.23}$$

where $\mathbf{x}_D$ is the location of the departure point relative to the cell corner and $\mathbf{u}$ is the velocity at the corner. This approximation is only first-order accurate. Accuracy can be improved by estimating the velocity at the midpoint of the trajectory.

### 2.4.3 Integrating fields

Next, we integrate the reconstructed fields over the departure triangles to find the total area, volume, and energy transported across each cell edge. Area transports are easy to compute since the area is linear in $x$ and $y$. Given a triangle with vertices $\mathbf{x_i} = (x_i, y_i)$, $i \in \{1, 2, 3\}$, the triangle area is

$$A_T = \frac{1}{2}\left|(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)\right|.$$

The integral $F_a$ of any linear function $f(\mathbf{r})$ over a triangle is given by

$$F_a = A_T f(\mathbf{x_0}), \tag{2.24}$$

where $\mathbf{x}_0 = (x_0, y_0)$ is the triangle midpoint,

$$\mathbf{x}_0 = \frac{1}{3}\sum_{i=1}^{3}\mathbf{x}_i.$$

To compute the area transport, we evaluate the area at the midpoint,

$$a(\mathbf{x}_0) = a_c + a_x x_0 + a_y y_0,$$

and multiply by $A_T$. By convention, northward and eastward transport is positive, while southward and westward transport is negative.

Equation (2.24) cannot be used for volume transport, because the reconstructed volumes are quadratic functions of position. (They are products of two linear functions, area and thickness.) The integral of a quadratic polynomial over a triangle requires function evaluations at three points,

$$F_h = \frac{A_T}{3}\sum_{i=1}^{3} f\left(\mathbf{x}_i'\right), \tag{2.25}$$

where $\mathbf{x}_i' = (\mathbf{x}_0 + \mathbf{x}_i)/2$ are points lying halfway between the midpoint and the three vertices. [7] use this formula to compute transports of the product $\rho T$, which is analogous to ice volume. Equation (2.25) does not work for ice and

snow energies, which are cubic functions—products of area, thickness, and enthalpy. Integrals of a cubic polynomial over a triangle can be evaluated using a four-point formula [42]:

$$F_q = A_T \left[ -\frac{9}{16} f(\mathbf{x}_0) + \frac{25}{48} \sum_{i=1}^{3} f(\mathbf{x}_i'') \right] \tag{2.26}$$

where $\mathbf{x_i}'' = (3\mathbf{x}_0 + 2\mathbf{x}_i)/5$. To evaluate functions at specific points, we must compute many products of the form $a(\mathbf{x})\,h(\mathbf{x})$ and $a(\mathbf{x})\,h(\mathbf{x})\,q(\mathbf{x})$, where each term in the product is the sum of a cell-center value and two displacement terms. In the code, the computation is sped up by storing some sums that are used repeatedly.

### 2.4.4 Updating state variables

Finally, we compute new values of the state variables in each ice category and grid cell. The new fractional ice areas $a'_{in}(i,j)$ are given by

$$a'_{in}(i,j) = a_{in}(i,j) + \frac{F_{aE}(i-1,j) - F_{aE}(i,j) + F_{aN}(i,j-1) - F_{aN}(i,j)}{A(i,j)} \tag{2.27}$$

where $F_{aE}(i,j)$ and $F_{aN}(i,j)$ are the area transports across the east and north edges, respectively, of cell $(i,j)$, and $A(i,j)$ is the grid cell area. All transports added to one cell are subtracted from a neighboring cell; thus Equation (2.27) conserves total ice area.

The new ice volumes and energies are computed analogously. New thicknesses are given by the ratio of volume to area, and enthalpies by the ratio of energy to volume. Tracer monotonicity is ensured because

$$h' = \frac{\int_A a\,h\,dA}{\int_A a\,dA},$$

$$q' = \frac{\int_A a\,h\,q\,dA}{\int_A a\,h\,dA},$$

where $h'$ and $q'$ are the new-time thickness and enthalpy, given by integrating the old-time ice area, volume, and energy over a Lagrangian departure region with area $A$. That is, the new-time thickness and enthalpy are weighted averages over old-time values, with non-negative weights $a$ and $ah$. Thus the new-time values must lie between the maximum and minimum of the old-time values.

## 2.5 Dynamics

There are now different rheologies available in the CICE code. The elastic-viscous-plastic (EVP) model represents a modification of the standard viscous-plastic (VP) model for sea ice dynamics [11]. The elastic-anisotropic-plastic (EAP) model, on the other hand, explicitly accounts for the observed sub-continuum anisotropy of the sea ice cover [49][47]. If *kdyn* = 1 in the namelist then the EVP rheology is used (module **ice_dyn_evp.F90**), while *kdyn* = 2 is associated with the EAP rheology (**ice_dyn_eap.F90**). At times scales associated with the wind forcing, the EVP model reduces to the VP model while the EAP model reduces to the anisotropic rheology described in detail in [49][45]. At shorter time scales the adjustment process takes place in both models by a numerically more efficient elastic wave mechanism. While retaining the essential physics, this elastic wave modification leads to a fully explicit numerical scheme which greatly improves the model's computational efficiency.

The EVP sea ice dynamics model is thoroughly documented in [16], [15], [17] and [18] and the EAP dynamics in [45]. Simulation results and performance of the EVP and EAP models have been compared with the VP model and with each other in realistic simulations of the Arctic respectively in [20] and [45]. Here we summarize the equations and direct the reader to the above references for details. The numerical implementation in this code release is that of [17] and [18], with revisions to the numerical solver as in [4]. The implementation of the EAP sea ice dynamics into CICE is described in detail in [45].

### 2.5.1 Momentum

The force balance per unit area in the ice pack is given by a two-dimensional momentum equation [11], obtained by integrating the 3D equation through the thickness of the ice in the vertical direction:

$$m\frac{\partial \mathbf{u}}{\partial t} = \nabla \cdot \sigma + \vec{\tau}_a + \vec{\tau}_w + \vec{\tau}_b - \hat{k} \times mf\mathbf{u} - mg\nabla H_\circ, \tag{2.28}$$

where $m$ is the combined mass of ice and snow per unit area and $\vec{\tau}_a$ and $\vec{\tau}_w$ are wind and ocean stresses, respectively. The term $\vec{\tau}_b$ is a seabed stress (also referred to as basal stress) that represents the grounding of pressure ridges in shallow water [24]. The mechanical properties of the ice are represented by the internal stress tensor $\sigma_{ij}$. The other two terms on the right hand side are stresses due to Coriolis effects and the sea surface slope. The parameterization for the wind and ice–ocean stress terms must contain the ice concentration as a multiplicative factor to be consistent with the formal theory of free drift in low ice concentration regions. A careful explanation of the issue and its continuum solution is provided in [18] and [5].

The momentum equation is discretized in time as follows, for the classic EVP approach. First, for clarity, the two components of Equation (2.28) are

$$\begin{aligned} m\frac{\partial u}{\partial t} = &\frac{\partial \sigma_{1j}}{\partial x_j} + \tau_{ax} + a_i c_w \rho_w \left|\mathbf{U}_w - \mathbf{u}\right| \left[(U_w - u)\cos\theta - (V_w - v)\sin\theta\right] - C_b u + mfv - mg\frac{\partial H_\circ}{\partial x}, \\ m\frac{\partial v}{\partial t} = &\frac{\partial \sigma_{2j}}{\partial x_j} + \tau_{ay} + a_i c_w \rho_w \left|\mathbf{U}_w - \mathbf{u}\right| \left[(U_w - u)\sin\theta + (V_w - v)\cos\theta\right] - C_b v - mfu - mg\frac{\partial H_\circ}{\partial y}. \end{aligned}$$

In the code, $\texttt{vrel} = a_i c_w \rho_w \left|\mathbf{U}_w - \mathbf{u}^k\right|$ and $C_b = T_b \left(\sqrt{(u^k)^2 + (v^k)^2} + u_0\right)^{-1}$, where $k$ denotes the subcycling step. The following equations illustrate the time discretization and define some of the other variables used in the code.

$$\underbrace{\left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b\right)}_{\texttt{cca}} u^{k+1} - \underbrace{(mf + \texttt{vrel}\sin\theta)}_{\texttt{ccb}} v^{k+1} = \underbrace{\frac{\partial \sigma_{1j}^{k+1}}{\partial x_j}}_{\texttt{strintx}} + \underbrace{\tau_{ax} - mg\frac{\partial H_\circ}{\partial x}}_{\texttt{forcex}} + \texttt{vrel}\underbrace{(U_w\cos\theta - V_w\sin\theta)}_{\texttt{waterx}} + \frac{m}{\Delta t_e}u^k, \tag{2.29}$$

$$\underbrace{(mf + \texttt{vrel}\sin\theta)}_{\texttt{ccb}} u^{k+1} + \underbrace{\left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b\right)}_{\texttt{cca}} v^{k+1} = \underbrace{\frac{\partial \sigma_{2j}^{k+1}}{\partial x_j}}_{\texttt{strinty}} + \underbrace{\tau_{ay} - mg\frac{\partial H_\circ}{\partial y}}_{\texttt{forcey}} + \texttt{vrel}\underbrace{(U_w\sin\theta + V_w\cos\theta)}_{\texttt{watery}} + \frac{m}{\Delta t_e}v^k, \tag{2.30}$$

and vrel·waterx(y) = taux(y).

We solve this system of equations analytically for $u^{k+1}$ and $v^{k+1}$. Define

$$\hat{u} = F_u + \tau_{ax} - mg\frac{\partial H_\circ}{\partial x} + \texttt{vrel}\,(U_w\cos\theta - V_w\sin\theta) + \frac{m}{\Delta t_e}u^k \tag{2.31}$$

$$\hat{v} = F_v + \tau_{ay} - mg\frac{\partial H_\circ}{\partial y} + \texttt{vrel}\,(U_w\sin\theta + V_w\cos\theta) + \frac{m}{\Delta t_e}v^k, \tag{2.32}$$

where $\mathbf{F} = \nabla \cdot \sigma^{k+1}$. Then

$$\begin{aligned} \left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b\right) u^{k+1} - (mf + \texttt{vrel}\sin\theta)\, v^{k+1} = &\quad \hat{u} \\ (mf + \texttt{vrel}\sin\theta)\, u^{k+1} + \left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b\right) v^{k+1} = &\quad \hat{v}. \end{aligned}$$

Solving simultaneously for $u^{k+1}$ and $v^{k+1}$,

$$\begin{aligned} u^{k+1} &= \frac{a\hat{u} + b\hat{v}}{a^2 + b^2} \\ v^{k+1} &= \frac{a\hat{v} - b\hat{u}}{a^2 + b^2}, \end{aligned}$$

where

$$a = \frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b \tag{2.33}$$

$$b = mf + \texttt{vrel}\sin\theta. \tag{2.34}$$

When the subcycling is finished for each (thermodynamic) time step, the ice–ocean stress must be constructed from *taux(y)* and the terms containing *vrel* on the left hand side of the equations.

The Hibler-Bryan form for the ice-ocean stress [13] is included in **ice_dyn_shared.F90** but is currently commented out, pending further testing.

### 2.5.2 Seabed stress

The parameterization for the seabed stress is described in [24]. The components of the basal seabed stress are $\tau_{bx} = C_b u$ and $\tau_{by} = C_b v$, where $C_b$ is a coefficient expressed as

$$C_b = k_2 \max[0, (h_u - h_{cu})] e^{-\alpha_b * (1 - a_u)} (\sqrt{u^2 + v^2} + u_0)^{-1}, \tag{2.35}$$

where $k_2$ determines the maximum seabed stress that can be sustained by the grounded parameterized ridge(s), $u_0$ is a small residual velocity and $\alpha_b = 20$ is a parameter to ensure that the seabed stress quickly drops when the ice concentration is smaller than 1. In the code, $k_2 \max[0, (h_u - h_{cu})] e^{-\alpha_b * (1 - a_u)}$ is defined as $T_b$. The quantities $h_u$, $a_u$ and $h_{cu}$ are calculated at the 'u' point based on local ice conditions (surrounding tracer points). They are respectively given by

$$h_u = \max[v_i(i,j), v_i(i+1,j), v_i(i,j+1), v_i(i+1,j+1)], \tag{2.36}$$

$$a_u = \max[a_i(i,j), a_i(i+1,j), a_i(i,j+1), a_i(i+1,j+1)]. \tag{2.37}$$

$$h_{cu} = a_u h_{wu} / k_1, \tag{2.38}$$

where the $a_i$ and $v_i$ are the total ice concentrations and ice volumes around the $u$ point $i, j$ and $k_1$ is a parameter that defines the critical ice thickness $h_{cu}$ at which the parameterized ridge(s) reaches the seafloor for a water depth $h_{wu} = \min[h_w(i,j), h_w(i+1,j), h_w(i,j+1), h_w(i+1,j+1)]$. Given the formulation of $C_b$ in equation (2.35), the seabed stress components are non-zero only when $h_u > h_{cu}$.

The maximum seabed stress depends on the weigth of the ridge above hydrostatic balance and the value of $k_2$. It is, however, the parameter $k_1$ that has the most notable impact on the simulated extent of landfast ice. The value of $k_1$ can be changed at runtime using the namelist variable `k1`. The grounding scheme can be turned on or off using the namelist logical basalstress.

Note that the user must provide a bathymetry field for using this grounding scheme. It is suggested to have a bathymetry field with water depths larger than 5 m that represents well shallow water regions such as the Laptev Sea and the East Siberian Sea. To prevent unrealistic grounding, $T_b$ is set to zero when $h_{wu}$ is larger than 30 m. This maximum value is chosen based on observations of large keels in the Arctic Ocean [1].

### 2.5.3 Internal stress

For convenience we formulate the stress tensor $\sigma$ in terms of $\sigma_1 = \sigma_{11} + \sigma_{22}$, $\sigma_2 = \sigma_{11} - \sigma_{22}$, and introduce the divergence, $D_D$, and the horizontal tension and shearing strain rates, $D_T$ and $D_S$ respectively.

CICE now outputs the internal ice pressure which is an important field to support navigation in ice-infested water. The internal ice pressure ($sigP$) is the average of the normal stresses multiplied by $-1$ and is therefore simply equal to $-\sigma_1/2$.

*Elastic-Viscous-Plastic*

In the EVP model the internal stress tensor is determined from a regularized version of the VP constitutive law. Following the approach of [2] (see also [24]), the elliptical yield curve can be modified such that the ice has isotropic tensile strength. The tensile strength $T_p$ is expressed as a fraction of the ice strength $P$, that is $T_p = k_t P$ where $k_t$ should be set to a value between 0 and 1 (this can be changed at runtime with the namelist parameter `Ktens`). The constitutive law is therefore

$$\frac{1}{E}\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2\zeta} + \frac{P_R(1-k_t)}{2\zeta} = D_D, \tag{2.39}$$

$$\frac{1}{E}\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2}{2\eta} = D_T, \tag{2.40}$$

$$\frac{1}{E}\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2\eta} = \frac{1}{2}D_S, \tag{2.41}$$

where

$$D_D = \dot{\epsilon}_{11} + \dot{\epsilon}_{22},$$

$$D_T = \dot{\epsilon}_{11} - \dot{\epsilon}_{22},$$

$$D_S = 2\dot{\epsilon}_{12},$$

$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),$$

$$\zeta = \frac{P(1+k_t)}{2\Delta},$$

$$\eta = \frac{P(1+k_t)}{2\Delta e^2},$$

$$\Delta = \left[D_D^2 + \frac{1}{e^2}\left(D_T^2 + D_S^2\right)\right]^{1/2},$$

and $P_R$ is a "replacement pressure" (see [9], for example), which serves to prevent residual ice motion due to spatial variations of $P$ when the rates of strain are exactly zero. The ice strength $P$ is a function of the ice thickness and concentration as it is described in the Icepack Documentation. The parameteter $e$ is the ratio of the major and minor axes of the elliptical yield curve, also called the ellipse aspect ratio. It can be changed using the namelist parameter `e_ratio`.

Viscosities are updated during the subcycling, so that the entire dynamics component is subcycled within the time step, and the elastic parameter $E$ is defined in terms of a damping timescale $T$ for elastic waves, $\Delta t_e < T < \Delta t$, as

$$E = \frac{\zeta}{T},$$

where $T = E_\circ \Delta t$ and $E_\circ$ (eyc) is a tunable parameter less than one. Including the modification proposed by [4] for equations (2.40) and (2.41) in order to improve numerical convergence, the stress equations become

$$\begin{aligned}
\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P_R(1-k_t)}{2T} &= \frac{P(1+k_t)}{2T\Delta}D_D, \\
\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2}{2T} &= \frac{P(1+k_t)}{2Te^2\Delta}D_T, \\
\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2T} &= \frac{P(1+k_t)}{4Te^2\Delta}D_S.
\end{aligned}$$

Once discretized in time, these last three equations are written as

$$\begin{aligned}
\frac{(\sigma_1^{k+1}-\sigma_1^k)}{\Delta t_e}+\frac{\sigma_1^{k+1}}{2T}+\frac{P_R^k(1-k_t)}{2T} &= \frac{P(1+k_t)}{2T\Delta^k}D_D^k,\\
\frac{(\sigma_2^{k+1}-\sigma_2^k)}{\Delta t_e}+\frac{\sigma_2^{k+1}}{2T} &= \frac{P(1+k_t)}{2Te^2\Delta^k}D_T^k,\\
\frac{(\sigma_{12}^{k+1}-\sigma_{12}^k)}{\Delta t_e}+\frac{\sigma_{12}^{k+1}}{2T} &= \frac{P(1+k_t)}{4Te^2\Delta^k}D_S^k,
\end{aligned} \tag{2.42}$$

where $k$ denotes again the subcycling step. All coefficients on the left-hand side are constant except for $P_R$. This modification compensates for the decreased efficiency of including the viscosity terms in the subcycling. (Note that the viscosities do not appear explicitly.) Choices of the parameters used to define $E$, $T$ and $\Delta t_e$ are discussed in Sections *Revised approach* and *Choosing an appropriate time step*.

The bilinear discretization used for the stress terms $\partial\sigma_{ij}/\partial x_j$ in the momentum equation is now used, which enabled the discrete equations to be derived from the continuous equations written in curvilinear coordinates. In this manner, metric terms associated with the curvature of the grid are incorporated into the discretization explicitly. Details pertaining to the spatial discretization are found in [17].

*Elastic-Anisotropic-Plastic*

In the EAP model the internal stress tensor is related to the geometrical properties and orientation of underlying virtual diamond shaped floes (see *Diamond-shaped floes*). In contrast to the isotropic EVP rheology, the anisotropic plastic yield curve within the EAP rheology depends on the relative orientation of the diamond shaped floes (unit vector $\mathbf{r}$ in *Diamond-shaped floes*), with respect to the principal direction of the deformation rate (not shown). Local anisotropy of the sea ice cover is accounted for by an additional prognostic variable, the structure tensor $\mathbf{A}$ defined by

$$\mathbf{A}=\int_{\mathbb{S}}\vartheta(\mathbf{r})\mathbf{r}\mathbf{r}d\mathbf{r}.$$

where $\mathbb{S}$ is a unit-radius circle; $\mathbf{A}$ is a unit trace, 2×2 matrix. From now on we shall describe the orientational distribution of floes using the structure tensor. For simplicity we take the probability density function $\vartheta(\mathbf{r})$ to be Gaussian, $\vartheta(z)=\omega_1\exp(-\omega_2 z^2)$, where $z$ is the ice floe inclination with respect to the axis $x_1$ of preferential alignment of ice floes (see *Diamond-shaped floes*), $\vartheta(z)$ is periodic with period $\pi$, and the positive coefficients $\omega_1$ and $\omega_2$ are calculated to ensure normalization of $\vartheta(z)$, i.e. $\int_0^{2\pi}\vartheta(z)dz=1$. The ratio of the principal components of $\mathbf{A}$, $A_1/A_2$, are derived from the phenomenological evolution equation for the structure tensor $\mathbf{A}$,

$$\frac{D\mathbf{A}}{Dt}=\mathbf{F}_{iso}(\mathbf{A})+\mathbf{F}_{frac}(\mathbf{A},\boldsymbol{\sigma}), \tag{2.43}$$

where $t$ is the time, and $D/Dt$ is the co-rotational time derivative accounting for advection and rigid body rotation ($D\mathbf{A}/Dt=d\mathbf{A}/dt-\mathbf{W}\cdot\mathbf{A}-\mathbf{A}\cdot\mathbf{W}^T$) with $\mathbf{W}$ being the vorticity tensor. $\mathbf{F}_{iso}$ is a function that accounts for a variety of processes (thermal cracking, melting, freezing together of floes) that contribute to a more isotropic nature to the ice cover. $\mathbf{F}_{frac}$ is a function determining the ice floe re-orientation due to fracture, and explicitly depends upon sea ice stress (but not its magnitude). Following [49], based on laboratory experiments by [40] we consider four failure mechanisms for the Arctic sea ice cover. These are determined by the ratio of the principal values of the sea ice stress $\sigma_1$ and $\sigma_2$: (i) under biaxial tension, fractures form across the perpendicular principal axes and therefore counteract any apparent redistribution of the floe orientation; (ii) if only one of the principal stresses is compressive, failure occurs through axial splitting along the compression direction; (iii) under biaxial compression with a low confinement ratio, ($\sigma_1/\sigma_2<R$), sea ice fails Coulombically through formation of slip lines delineating new ice floes oriented along the largest compressive stress; and finally (iv) under biaxial compression with a large confinement ratio, ($\sigma_1/\sigma_2\geq R$), the ice is expected to fail along both principal directions so that the cumulative directional effect balances to zero.

Figure *Diamond-shaped floes* shows geometry of interlocking diamond-shaped floes (taken from [49]). $\phi$ is half of the acute angle of the diamonds. $L$ is the edge length. $\boldsymbol{n}_1$, $\boldsymbol{n}_2$ and $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$ are respectively the normal and tangential unit vectors along the diamond edges. $\mathbf{v}=L\boldsymbol{\tau}_2\cdot\dot{\boldsymbol{\epsilon}}$ is the relative velocity between the two floes connected by the vector $L\boldsymbol{\tau}_2$. $\mathbf{r}$ is the unit vector along the main diagonal of the diamond. Note that the diamonds illustrated here represent

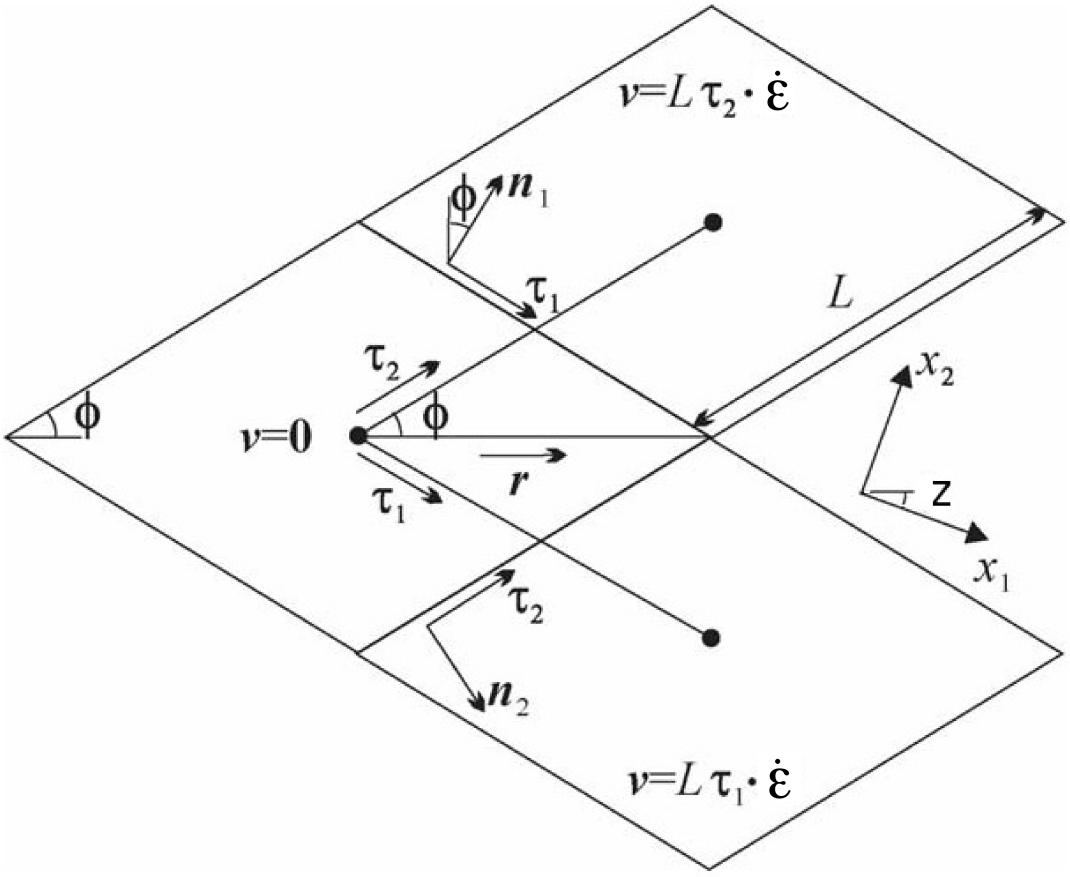


Fig. 3: Diamond-shaped floes

one possible realisation of all possible orientations. The angle $z$ represents the rotation of the diamonds' main axis relative to their preferential orientation along the axis $x_1$.

The new anisotropic rheology requires solving the evolution Equation (2.43) for the structure tensor in addition to the momentum and stress equations. The evolution equation for $\mathbf{A}$ is solved within the EVP subcycling loop, and consistently with the momentum and stress evolution equations, we neglect the advection term for the structure tensor. Equation (2.43) then reduces to the system of two equations:

$$\begin{aligned}\frac{\partial A_{11}}{\partial t} &= -k_t\left(A_{11}-\frac{1}{2}\right)+M_{11},\\ \frac{\partial A_{12}}{\partial t} &= -k_t A_{12}+M_{12},\end{aligned}$$

where the first terms on the right hand side correspond to the isotropic contribution, $F_{iso}$, and $M_{11}$ and $M_{12}$ are the components of the term $F_{frac}$ in Equation (2.43) that are given in [49] and [45]. These evolution equations are discretized semi-implicitly in time. The degree of anisotropy is measured by the largest eigenvalue ($A_1$) of this tensor ($A_2 = 1 - A_1$). $A_1 = 1$ corresponds to perfectly aligned floes and $A_1 = 0.5$ to a uniform distribution of floe orientation. Note that while we have specified the aspect ratio of the diamond floes, through prescribing $\phi$, we make no assumption about the size of the diamonds so that formally the theory is scale invariant.

As described in greater detail in [49], the internal ice stress for a single orientation of the ice floes can be calculated explicitly and decomposed, for an average ice thickness $h$, into its ridging (r) and sliding (s) contributions

$$\boldsymbol{\sigma}^b(\mathbf{r},h) = P_r(h)\boldsymbol{\sigma}_r^b(\mathbf{r}) + P_s(h)\boldsymbol{\sigma}_s^b(\mathbf{r}), \tag{2.44}$$

where $P_r$ and $P_s$ are the ridging and sliding strengths and the ridging and sliding stresses are functions of the angle $\theta = \arctan(\dot{\epsilon}_{II}/\dot{\epsilon}_I)$, the angle $y$ between the major principal axis of the strain rate tensor (not shown) and the structure tensor ($x_1$ axis in *Diamond-shaped floes*, and the angle $z$ defined in *Diamond-shaped floes*. In the stress expressions above the underlying floes are assumed parallel, but in a continuum-scale sea ice region the floes can possess different orientations in different places and we take the mean sea ice stress over a collection of floes to be given by the average

$$\boldsymbol{\sigma}^{EAP}(h) = P_r(h)\int_{\mathbb{S}} \vartheta(\mathbf{r})\left[\boldsymbol{\sigma}_r^b(\mathbf{r}) + k\boldsymbol{\sigma}_s^b(\mathbf{r})\right] d\mathbf{r} \tag{2.45}$$

where we have introduced the friction parameter $k = P_s/P_r$ and where we identify the ridging ice strength $P_r(h)$ with the strength $P$ described in section 1 and used within the EVP framework.

As is the case for the EVP rheology, elasticity is included in the EAP description not to describe any physical effect, but to make use of the efficient, explicit numerical algorithm used to solve the full sea ice momentum balance. We use the analogous EAP stress equations,

$$\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} = \frac{\sigma_1^{EAP}}{2T}, \tag{2.46}$$

$$\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2}{2T} = \frac{\sigma_2^{EAP}}{2T}, \tag{2.47}$$

$$\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2T} = \frac{\sigma_{12}^{EAP}}{2T}, \tag{2.48}$$

where the anisotropic stress $\boldsymbol{\sigma}^{EAP}$ is defined in a look-up table for the current values of strain rate and structure tensor. The look-up table is constructed by computing the stress (normalized by the strength) from Equations (2.46)–(2.48) for discrete values of the largest eigenvalue of the structure tensor, $\frac{1}{2} \le A_1 \le 1$, the angle $0 \le \theta \le 2\pi$, and the angle $-\pi/2 \le y \le \pi/2$ between the major principal axis of the strain rate tensor and the structure tensor [45]. The updated stress, after the elastic relaxation, is then passed to the momentum equation and the sea ice velocities are updated in the usual manner within the subcycling loop of the EVP rheology. The structure tensor evolution equations are solved implicitly at the same frequency, $\Delta t_e$, as the ice velocities and internal stresses. Finally, to be coherent with our new rheology we compute the area loss rate due to ridging as $|\dot{\boldsymbol{\epsilon}}|\alpha_r(\theta)$, with $\alpha_r(\theta)$ and $\alpha_s(\theta)$ given by [48],

$$\alpha_r(\theta) = \frac{\sigma_{ij}^r\dot{\epsilon}_{ij}}{P_r|\dot{\boldsymbol{\epsilon}}|}, \qquad \alpha_s(\theta) = \frac{\sigma_{ij}^s\dot{\epsilon}_{ij}}{P_s|\dot{\boldsymbol{\epsilon}}|}.$$

Both ridging rate and sea ice strength are computed in the outer loop of the dynamics.

### 2.5.4 Revised approach

The revised EVP approach is based on a pseudo-time iterative scheme [25], [4], [23]. By construction, the revised EVP approach should lead to the VP solution (given the right numerical parameters and a sufficiently large number of iterations). To do so, the inertial term is formulated such that it matches the backward Euler approach of implicit solvers and there is an additional term for the pseudo-time iteration. Hence, with the revised approach, the discretized momentum equations (2.29) and (2.30) become

$$\frac{\beta^*(u^{k+1}-u^k)}{\Delta t_e}+\frac{m(u^{k+1}-u^n)}{\Delta t}+(\texttt{vrel}\cos\theta+C_b)u^{k+1}-(mf+\texttt{vrel}\sin\theta)v^{k+1}=\frac{\partial\sigma_{1j}^{k+1}}{\partial x_j}+\tau_{ax}-mg\frac{\partial H_\circ}{\partial x}+\texttt{vrel}(U_w\cos\theta \tag{2.49}$$

$$\frac{\beta^*(v^{k+1}-v^k)}{\Delta t_e}+\frac{m(v^{k+1}-v^n)}{\Delta t}+(\texttt{vrel}\cos\theta+C_b)v^{k+1}+(mf+\texttt{vrel}\sin\theta)u^{k+1}=\frac{\partial\sigma_{2j}^{k+1}}{\partial x_j}+\tau_{ay}-mg\frac{\partial H_\circ}{\partial y}+\texttt{vrel}(U_w\sin\theta \tag{2.50}$$

where $\beta^*$ is a numerical parameter and $u^n, v^n$ are the components of the previous time level solution. With $\beta = \beta^*\Delta t\,(m\Delta t_e)^{-1}$ [4], these equations can be written as

$$\underbrace{\left((\beta+1)\frac{m}{\Delta t}+\texttt{vrel}\cos\theta\ +C_b\right)}_{\texttt{cca}}u^{k+1}-\underbrace{(mf+\texttt{vrel}\sin\theta)}_{\texttt{ccb}}v^{k+1}=\underbrace{\frac{\partial\sigma_{1j}^{k+1}}{\partial x_j}}_{\texttt{strintx}}+\underbrace{\tau_{ax}-mg\frac{\partial H_\circ}{\partial x}}_{\texttt{forcex}}+\texttt{vrel}\underbrace{(U_w\cos\theta-V_w\sin\theta)}_{\texttt{waterx}}+\frac{m}{\Delta t}(\beta \tag{2.51}$$

$$\underbrace{(mf+\texttt{vrel}\sin\theta)}_{\texttt{ccb}}u^{k+1}+\underbrace{\left((\beta+1)\frac{m}{\Delta t}+\texttt{vrel}\cos\theta+C_b\right)}_{\texttt{cca}}v^{k+1}=\underbrace{\frac{\partial\sigma_{2j}^{k+1}}{\partial x_j}}_{\texttt{strinty}}+\underbrace{\tau_{ay}-mg\frac{\partial H_\circ}{\partial y}}_{\texttt{forcey}}+\texttt{vrel}\underbrace{(U_w\sin\theta+V_w\cos\theta)}_{\texttt{watery}}+\frac{m}{\Delta t}(\beta v \tag{2.52}$$

At this point, the solutions $u^{k+1}$ and $v^{k+1}$ are obtained in the same manner as for the standard EVP approach (see equations (2.31) to (2.34)).

Introducing another numerical parameter $\alpha = 2T\Delta t_e^{-1}$ [4], the stress equations in (2.42) become

$$\begin{aligned}\alpha(\sigma_1^{k+1}-\sigma_1^k)+\sigma_1^k+P_R^k(1-k_t)&=\frac{P(1+k_t)}{\Delta^k}D_D^k,\\ \alpha(\sigma_2^{k+1}-\sigma_2^k)+\sigma_2^k&=\frac{P(1+k_t)}{e^2\Delta^k}D_T^k,\\ \alpha(\sigma_{12}^{k+1}-\sigma_{12}^k)+\sigma_{12}^k&=\frac{P(1+k_t)}{2e^2\Delta^k}D_S^k,\end{aligned}$$

where as opposed to the classic EVP, the second term in each equation is at iteration $k$ [4]. Also, as opposed to the classic EVP, $\Delta t_e$ times the number of subcycles (or iterations) does not need to be equal to the advective time step $\Delta t$. Finally, as with the classic EVP approach, the stresses are initialized using the previous time level values. The revised EVP is activated by setting the namelist parameter *revised_evp* = true. In the code $\alpha = arlx$ and $\beta = brlx$. The values of $arlx$ and $brlx$ can be set in the namelist. It is recommended to use large values of these parameters and to set $arlx = brlx$ [23].

CHAPTER 3

# User Guide

## 3.1 Implementation

CICE is written in FORTRAN90 and runs on platforms using UNIX, LINUX, and other operating systems. The code is based on a two-dimensional horizontal orthogonal grid that is broken into two-dimensional horizontal blocks and parallelized over blocks with MPI and OpenMP threads. The code also includes some optimizations for vector architectures.

CICE consists of source code under the **cicecore/** directory that supports model dynamics and top-level control. The column physics source code is under the **icepack/** directory and this is implemented as a submodule in github from a separate repository (CICE) There is also a **configuration/** directory that includes scripts for configuring CICE cases.

### 3.1.1 Directory structure

The present code distribution includes source code and scripts. Forcing data is available from the ftp site. The directory structure of CICE is as follows

**LICENSE.pdf** license for using and sharing the code

**DistributionPolicy.pdf** policy for using and sharing the code

**README.md** basic information and pointers

**icepack/** the Icepack module. The icepack subdirectory includes Icepack specific scripts, drivers, and documentation. CICE only uses the columnphysics source code under **icepack/columnphysics/**.

**cicecore/** CICE source code

**cicecore/cicedynB/** routines associated with the dynamics core

**cicecore/driver/** top-level CICE drivers and coupling layers

**cicecore/shared/** CICE source code that is independent of the dynamical core

**cicecore/version.txt** file that indicates the CICE model version.

**configuration/scripts/** support scripts, see *Scripts*

**doc/** documentation

**cice.setup** main CICE script for creating cases

A case (compile) directory is created upon initial execution of the script **cice.setup** at the user-specified location provided after the -c flag. Executing the command `./cice.setup -h` provides helpful information for this tool.

### 3.1.2 Grid, boundary conditions and masks

The spatial discretization is specialized for a generalized orthogonal B-grid as in [32] or [41]. The ice and snow area, volume and energy are given at the center of the cell, velocity is defined at the corners, and the internal ice stress tensor takes four different values within a grid cell; bilinear approximations are used for the stress tensor and the ice velocity across the cell, as described in [17]. This tends to avoid the grid decoupling problems associated with the B-grid. EVP is available on the C-grid through the MITgcm code distribution, http://mitgcm.org/viewvc/MITgcm/MITgcm/pkg/seaice/.

Since ice thickness and thermodynamic variables such as temperature are given in the center of each cell, the grid cells are referred to as "T cells." We also occasionally refer to "U cells," which are centered on the northeast corner of the corresponding T cells and have velocity in the center of each. The velocity components are aligned along grid lines.

The user has several choices of grid routines: *popgrid* reads grid lengths and other parameters for a nonuniform grid (including tripole and regional grids), and *rectgrid* creates a regular rectangular grid. The input files **global_gx3.grid** and **global_gx3.kmt** contain the $\langle 3^\circ \rangle$ POP grid and land mask; **global_gx1.grid** and **global_gx1.kmt** contain the $\langle 1^\circ \rangle$ grid and land mask, and **global_tx1.grid** and **global_tx1.kmt** contain the $\langle 1^\circ \rangle$ POP tripole grid and land mask. These are binary unformatted, direct access, Big Endian files.

In CESM, the sea ice model may exchange coupling fluxes using a different grid than the computational grid. This functionality is activated using the namelist variable `gridcpl\_file`.

#### Grid domains and blocks

In general, the global gridded domain is `nx\_global` × `ny\_global`, while the subdomains used in the block distribution are `nx\_block` × `ny\_block`. The physical portion of a subdomain is indexed as [`ilo:ihi`, `jlo:jhi`], with nghost "ghost" or "halo" cells outside the domain used for boundary conditions. These parameters are illustrated in *Grid parameters* in one dimension. The routines *global_scatter* and *global_gather* distribute information from the global domain to the local domains and back, respectively. If MPI is not being used for grid decomposition in the ice model, these routines simply adjust the indexing on the global domain to the single, local domain index coordinates. Although we recommend that the user choose the local domains so that the global domain is evenly divided, if this is not possible then the furthest east and/or north blocks will contain nonphysical points ("padding"). These points are excluded from the computation domain and have little effect on model performance.

Figure *Grid parameters* shows the grid parameters for a sample one-dimensional, 20-cell global domain decomposed into four local subdomains. Each local domain has one ghost (halo) cell on each side, and the physical portion of the local domains are labeled `ilo:ihi`. The parameter `nx\_block` is the total number of cells in the local domain, including ghost cells, and the same numbering system is applied to each of the four subdomains.

The user sets the `NTASKS` and `NTHRDS` settings in **cice.settings** and chooses a block size `block\_size\_x` × `block\_size\_y`, `max\_blocks`, and decomposition information `distribution\_type`, `processor\_shape`, and `distribution\_type` in **ice_in**. That information is used to determine how the blocks are distributed across the processors, and how the processors are distributed across the grid domain. Recommended combinations of these parameters for best performance are given in Section *Performance*. The script **cice.setup** computes some default decompositions and layouts but the user can overwrite the defaults by manually changing the values in *ice_in*. At runtime, the model will print decomposition information to the log file, and if the block size or max blocks is inconsistent with the task and thread size, the model will abort. The code will also print a warning if the maximum number of blocks is too large. Although this is not fatal, it does use extra memory. If `max\_blocks` is set to -1, the code will compute a `max\_blocks` on the fly.

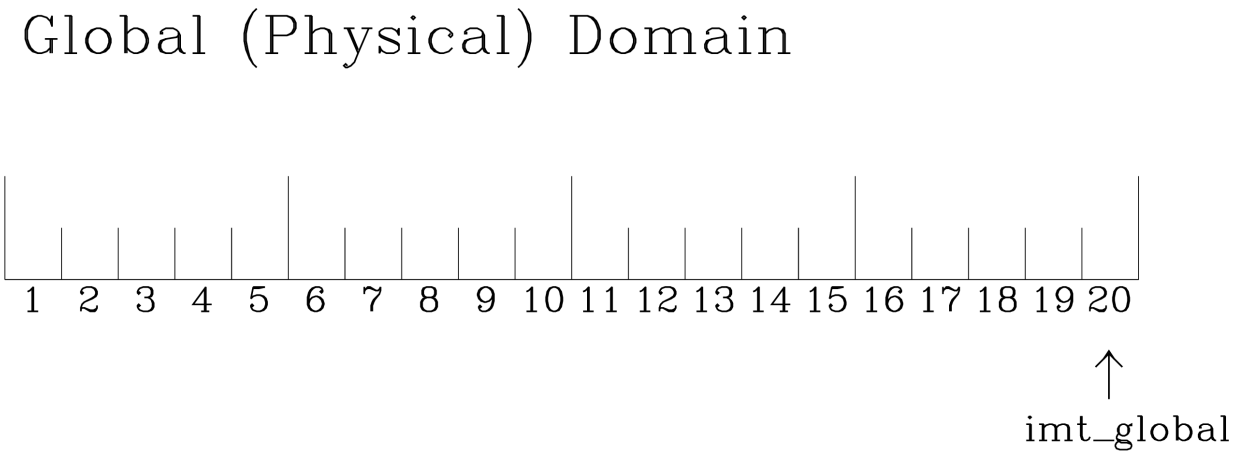


Fig. 1: Grid parameters

A loop at the end of routine *create_blocks* in module **ice_blocks.F90** will print the locations for all of the blocks on the global grid if dbug is set to be true. Likewise, a similar loop at the end of routine *create_local_block_ids* in module **ice_distribution.F90** will print the processor and local block number for each block. With this information, the grid decomposition into processors and blocks can be ascertained. The dbug flag must be manually set in the code in each case (independently of the dbug flag in **ice_in**), as there may be hundreds or thousands of blocks to print and this information should be needed only rarely. This information is much easier to look at using a debugger such as Totalview. There is also an output field that can be activated in *icefields_nml*, `f\_blkmask`, that prints out the variable `blkmask` to the history file and which labels the blocks in the grid decomposition according to `blkmask = my\_task + iblk/100`.

### Tripole grids

The tripole grid is a device for constructing a global grid with a normal south pole and southern boundary condition, which avoids placing a physical boundary or grid singularity in the Arctic Ocean. Instead of a single north pole, it has two "poles" in the north, both located on land, with a line of grid points between them. This line of points is called the "fold," and it is the "top row" of the physical grid. One pole is at the left-hand end of the top row, and the other is in the middle of the row. The grid is constructed by "folding" the top row, so that the left-hand half and the right-hand half of it coincide. Two choices for constructing the tripole grid are available. The one first introduced to CICE is called "U-fold", which means that the poles and the grid cells between them are U cells on the grid. Alternatively the poles and the cells between them can be grid T cells, making a "T-fold." Both of these options are also supported by the OPA/NEMO ocean model, which calls the U-fold an "f-fold" (because it uses the Arakawa C-grid in which U cells are on T-rows). The choice of tripole grid is given by the namelist variable `ns\_boundary\_type`, 'tripole' for the U-fold and 'tripoleT' for the T-fold grid.

In the U-fold tripole grid, the poles have U-index `nx_global`$/2$ and `nx\_global` on the top U-row of the physical grid, and points with U-index i and `nx_global` $-$ `i` are coincident. Let the fold have U-row index $n$ on the global grid; this will also be the T-row index of the T-row to the south of the fold. There are ghost (halo) T- and U-rows to the north, beyond the fold, on the logical grid. The point with index i along the ghost T-row of index $n+1$ physically coincides with point `nx_global` $-$ `i` $+1$ on the T-row of index $n$. The ghost U-row of index $n+1$ physically coincides with the U-row of index $n-1$.

In the T-fold tripole grid, the poles have T-index 1 and and `nx_global`$/2+1$ on the top T-row of the physical grid, and points with T-index i and `nx_global` $-$ `i` $+2$ are coincident. Let the fold have T-row index $n$ on the global grid. It is usual for the northernmost row of the physical domain to be a U-row, but in the case of the T-fold, the U-row of index $n$ is "beyond" the fold; although it is not a ghost row, it is not physically independent, because it coincides with U-row $n-1$, and it therefore has to be treated like a ghost row. Points i on U-row $n$ coincides with `nx_global` $-$ `i` $+1$ on U-row $n-1$. There are still ghost T- and U-rows $n+1$ to the north of U-row $n$. Ghost T-row $n+1$ coincides with T-row $n-1$, and ghost U-row $n+1$ coincides with U-row $n-2$.

The tripole grid thus requires two special kinds of treatment for certain rows, arranged by the halo-update routines. First, within rows along the fold, coincident points must always have the same value. This is achieved by averaging them in pairs. Second, values for ghost rows and the "quasi-ghost" U-row on the T-fold grid are reflected copies of the coincident physical rows. Both operations involve the tripole buffer, which is used to assemble the data for the affected rows. Special treatment is also required in the scattering routine, and when computing global sums one of each pair of coincident points has to be excluded.

### Vertical Grids

The sea ice physics described in a single column or grid cell is contained in the Icepack submodule, which can be run independently of the CICE model. Icepack includes a vertical grid for the physics and a "bio-grid" for biogeochemistry, described in the Icepack Documentation. History variables available for column output are ice and snow temperature, Tinz and Tsnz, and the ice salinity profile, Sinz. These variables also include thickness category as a fourth dimension.

### Boundary conditions

Much of the infrastructure used in CICE, including the boundary routines, is adopted from POP. The boundary routines perform boundary communications among processors when MPI is in use and among blocks whenever there is more than one block per processor.

Open/cyclic boundary conditions are the default in CICE; the physical domain can still be closed using the land mask. In our bipolar, displaced-pole grids, one row of grid cells along the north and south boundaries is located on land, and along east/west domain boundaries not masked by land, periodic conditions wrap the domain around the globe. CICE can be run on regional grids with open boundary conditions; except for variables describing grid lengths, non-land halo cells along the grid edge must be filled by restoring them to specified values. The namelist variable `restore\_ice` turns this functionality on and off; the restoring timescale `trestore` may be used (it is also used for restoring ocean sea surface temperature in stand-alone ice runs). This implementation is only intended to provide the "hooks" for a more sophisticated treatment; the rectangular grid option can be used to test this configuration. The 'displaced_pole' grid option should not be used unless the regional grid contains land all along the north and south boundaries. The current form of the boundary condition routines does not allow Neumann boundary conditions, which must be set explicitly. This has been done in an unreleased branch of the code; contact Elizabeth for more information.

For exact restarts using restoring, set `restart\_ext` = true in namelist to use the extended-grid subroutines.

On tripole grids, the order of operations used for calculating elements of the stress tensor can differ on either side of the fold, leading to round-off differences. Although restarts using the extended grid routines are exact for a given run, the solution will differ from another run in which restarts are written at different times. For this reason, explicit halo updates of the stress tensor are implemented for the tripole grid, both within the dynamics calculation and for restarts. This has not been implemented yet for tripoleT grids, pending further testing.

### Masks

A land mask hm ($M_h$) is specified in the cell centers, with 0 representing land and 1 representing ocean cells. A corresponding mask uvm ($M_u$) for velocity and other corner quantities is given by

$$M_u(i,j) = \min\{M_h(l),\, l = (i,j),\, (i+1,j),\, (i,j+1),\, (i+1,j+1)\}.$$

The logical masks `tmask` and `umask` (which correspond to the real masks `hm` and `uvm`, respectively) are useful in conditional statements.

In addition to the land masks, two other masks are implemented in *dyn_prep* in order to reduce the dynamics component's work on a global grid. At each time step the logical masks `ice\_tmask` and `ice\_umask` are determined from the current ice extent, such that they have the value "true" wherever ice exists. They also include a border of cells around the ice pack for numerical purposes. These masks are used in the dynamics component to prevent unnecessary calculations on grid points where there is no ice. They are not used in the thermodynamics component, so that ice may form in previously ice-free cells. Like the land masks `hm` and `uvm`, the ice extent masks `ice\_tmask` and `ice\_umask` are for T cells and U cells, respectively.

Improved parallel performance may result from utilizing halo masks for boundary updates of the full ice state, incremental remapping transport, or for EVP or EAP dynamics. These options are accessed through the logical namelist flags `maskhalo\_bound`, `maskhalo\_remap`, and `maskhalo\_dyn`, respectively. Only the halo cells containing needed information are communicated.

Two additional masks are created for the user's convenience: `lmask\_n` and `lmask\_s` can be used to compute or write data only for the northern or southern hemispheres, respectively. Special constants (`spval` and `spval\_dbl`, each equal to $10^{30}$) are used to indicate land points in the history files and diagnostics.

### Performance

Namelist options (*domain_nml*) provide considerable flexibility for finding efficient processor and block configuration. Some of these choices are illustrated in *Distribution options*. Users have control of many aspects of the decomposition such as the block size (`block\_size\_x`, `block\_size\_y`), the `distribution\_type`, the `distribution\_wght`, the `distribution\_wght\_file` (when `distribution\_type` = `wghtfile`), and the `processor\_shape` (when `distribution\_type` = `cartesian`).

The user specifies the total number of tasks and threads in **cice.settings** and the block size and decompostion in the namelist file. The main trades offs are the relative efficiency of large square blocks versus model internal load balance as CICE computation cost is very small for ice-free blocks. Smaller, more numerous blocks provides an opportunity for better load balance by allocating each processor both ice-covered and ice-free blocks. But smaller, more numerous blocks becomes less efficient due to MPI communication associated with halo updates. In practice, blocks should probably not have fewer than about 8 to 10 grid cells in each direction, and more square blocks tend to optimize the volume-to-surface ratio important for communication cost. Often 3 to 8 blocks per processor provide the decompositions flexiblity to create reasonable load balance configurations.

The `distribution\_type` options allow standard cartesian distributions of blocks, redistribution via a 'rake' algorithm for improved load balancing across processors, and redistribution based on space-filling curves. There are also additional distribution types ('roundrobin,' 'sectrobin,' 'sectcart', and 'spiralcenter') that support alternative decompositions and also allow more flexibility in the number of processors used. Finally, there is a 'wghtfile' decomposition that generates a decomposition based on weights specified in an input file.

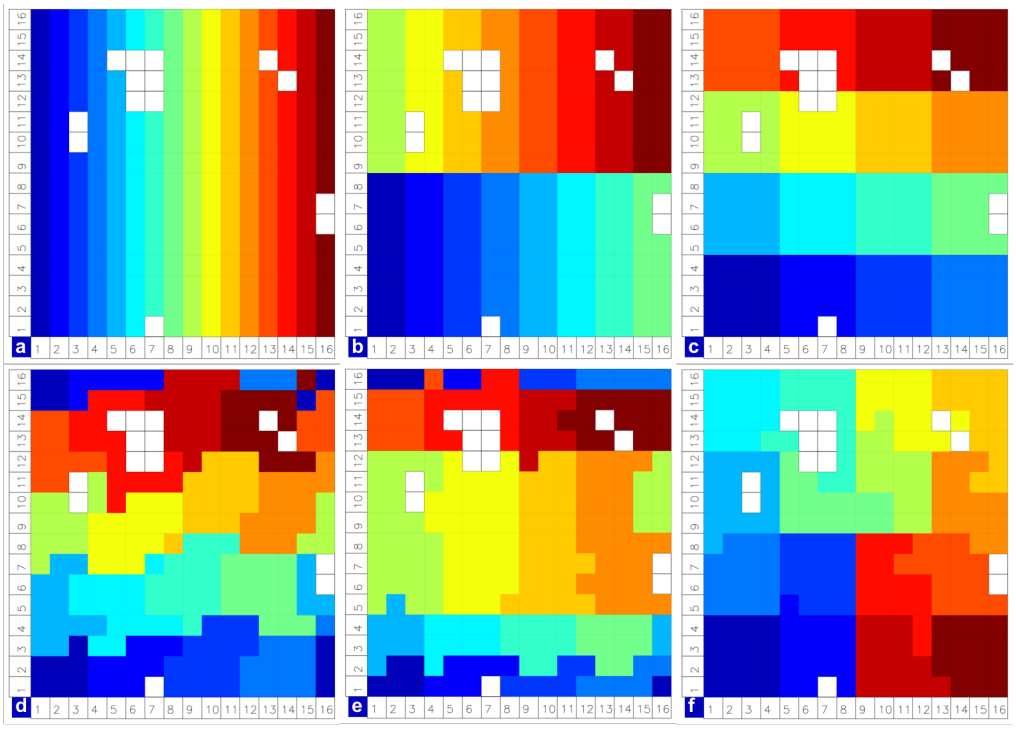


Fig. 2: Distribution options

Figure *Distribution options* shows distribution of 256 blocks across 16 processors, represented by colors, on the gx1 grid: (a) cartesian, slenderX1, (b) cartesian, slenderX2, (c) cartesian, square-ice (square-pop is equivalent here), (d) rake with block weighting, (e) rake with latitude weighting, (f) spacecurve. Each block consists of 20x24 grid cells, and white blocks consist entirely of land cells.

Figure *Decomposition options* shows sample decompositions for (a) spiral center and (b) wghtfile for an Arctic polar grid. (c) is the weight field in the input file use to drive the decompostion in (b).

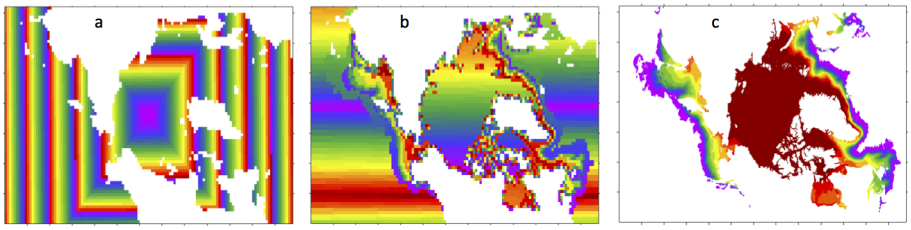


Fig. 3: Decomposition options

`processor\_shape` is used with the `distribution\_type` cartesian option, and it allocates blocks to processors in various groupings such as tall, thin processor domains (`slenderX1` or `slenderX2`, often better for sea ice simulations on global grids where nearly all of the work is at the top and bottom of the grid with little to do in between) and close-to-square domains (`square-pop` or `square-ice`), which maximize the volume to surface ratio (and therefore on-processor computations to message passing, if there were ice in every grid cell). In cases where the number of processors is not a perfect square (4, 9, 16...), the `processor\_shape` namelist variable allows the user to choose how the processors are arranged. Here again, it is better in the sea ice model to have more processors in x than in y, for example, 8 processors arranged 4x2 (`square-ice`) rather than 2x4 (`square-pop`). The latter option is offered for direct-communication compatibility with POP, in which this is the default.

`distribution\_wght` chooses how the work-per-block estimates are weighted. The 'block' option is the default in POP and it weights each block equally. This is useful in POP which always has work in each block and is written with a lot of array syntax requiring calculations over entire blocks (whether or not land is present). This option is provided in CICE as well for direct-communication compatibility with POP. The 'latitude' option weights the blocks based on latitude and the number of ocean grid cells they contain. Many of the non-cartesian decompositions support automatic land block elimination and provide alternative ways to decompose blocks without needing the `distribution\_wght`.

The rake distribution type is initialized as a standard, Cartesian distribution. Using the work-per-block estimates, blocks are "raked" onto neighboring processors as needed to improve load balancing characteristics among processors, first in the x direction and then in y.

Space-filling curves reduce a multi-dimensional space (2D, in our case) to one dimension. The curve is composed of a string of blocks that is snipped into sections, again based on the work per processor, and each piece is placed on a processor for optimal load balancing. This option requires that the block size be chosen such that the number of blocks in the x direction and the number of blocks in the y direction must be factorable as $2^n 3^m 5^p$ where $n, m, p$ are integers. For example, a 16x16 array of blocks, each containing 20x24 grid cells, fills the gx1 grid ($n = 4, m = p = 0$). If either of these conditions is not met, the spacecurve decomposition will fail.

While the Cartesian distribution groups sets of blocks by processor, the 'roundrobin' distribution loops through the blocks and processors together, putting one block on each processor until the blocks are gone. This provides good load balancing but poor communication characteristics due to the number of neighbors and the amount of data needed to communicate. The 'sectrobin' and 'sectcart' algorithms loop similarly, but put groups of blocks on each processor to improve the communication characteristics. In the 'sectcart' case, the domain is divided into four (east-west,north-south) quarters and the loops are done over each, sequentially.

The `wghtfile` decomposition drives the decomposition based on weights provided in a weight file. That file should be a netcdf file with a double real field called `wght` containing the relative weight of each gridcell. *Decomposition options* (b) and (c) show an example. The weights associated with each gridcell will be summed on a per block basis and normalized to about 10 bins to carry out the distribution of highest to lowest block weights to processors. *Scorecard* provides an overview of the pros and cons of the various distribution types.

Figure *Scorecard* shows the scorecard for block distribution choices in CICE, courtesy T. Craig. For more information, see [6] or http://www.cesm.ucar.edu/events/workshops/ws.2012/presentations/sewg/craig.pdf

The `maskhalo` options in the namelist improve performance by removing unnecessary halo communications where

## Grading CICE Decompositions

| decomposition | cice load balance, ice coverage | cice load balance, radiation | number of neighbors | amount of data to communicate | land block elimination in decomp | flexibility wrt pe counts | notes |
|---|---|---|---|---|---|---|---|
| cartesian, square- | F | F | B | A | F | C | |
| cartesian, slenderX1 | A | B | A | C | F | F | better for lon/lat type grids |
| cartesian, slenderX2 | B | C | B | B | F | F | better for lon/lat type grids |
| rake | C | C | B | A | A | C | |
| spacecurve | D | D | B | B | A | A | limited to certain block sizes |
| roundrobin | A | A | D | D | A | A | |
| sectrobin | B | A | C | C | A | A | |
| sectcart | B | A | B | C | F | C | |
| spiralcurve | B | B | D | D | A | A | better for polar domains |
| wghtfile | A | B | C | C | A | A | requires input file preparation |

Fig. 4: Scorecard

there is no ice. There is some overhead in setting up the halo masks, which is done during the timestepping procedure as the ice area changes, but this option usually improves timings even for relatively small processor counts. T. Craig has found that performance improved by more than 20% for combinations of updated decompositions and masked haloes, in CESM's version of CICE.

Throughout the code, (i, j) loops have been combined into a single loop, often over just ocean cells or those containing sea ice. This was done to reduce unnecessary operations and to improve vector performance.

*Timings* illustrates the CICE v5 computational expense of various options, relative to the total time (excluding initialization) of a 7-layer configuration using BL99 thermodynamics, EVP dynamics, and the 'ccsm3' shortwave parameterization on the gx1 grid, run for one year from a no-ice initial condition. The block distribution consisted of $20 \times 192$ blocks spread over 32 processors ('slenderX2') with no threads and -O2 optimization. Timings varied by about $\pm 3\%$ in identically configured runs due to machine load. Extra time required for tracers has two components, that needed to carry the tracer itself (advection, category conversions) and that needed for the calculations associated with the particular tracer. The age tracers (FY and iage) require very little extra calculation, so their timings represent essentially the time needed just to carry an extra tracer. The topo melt pond scheme is slightly faster than the others because it calculates pond area and volume once per grid cell, while the others calculate it for each thickness category.

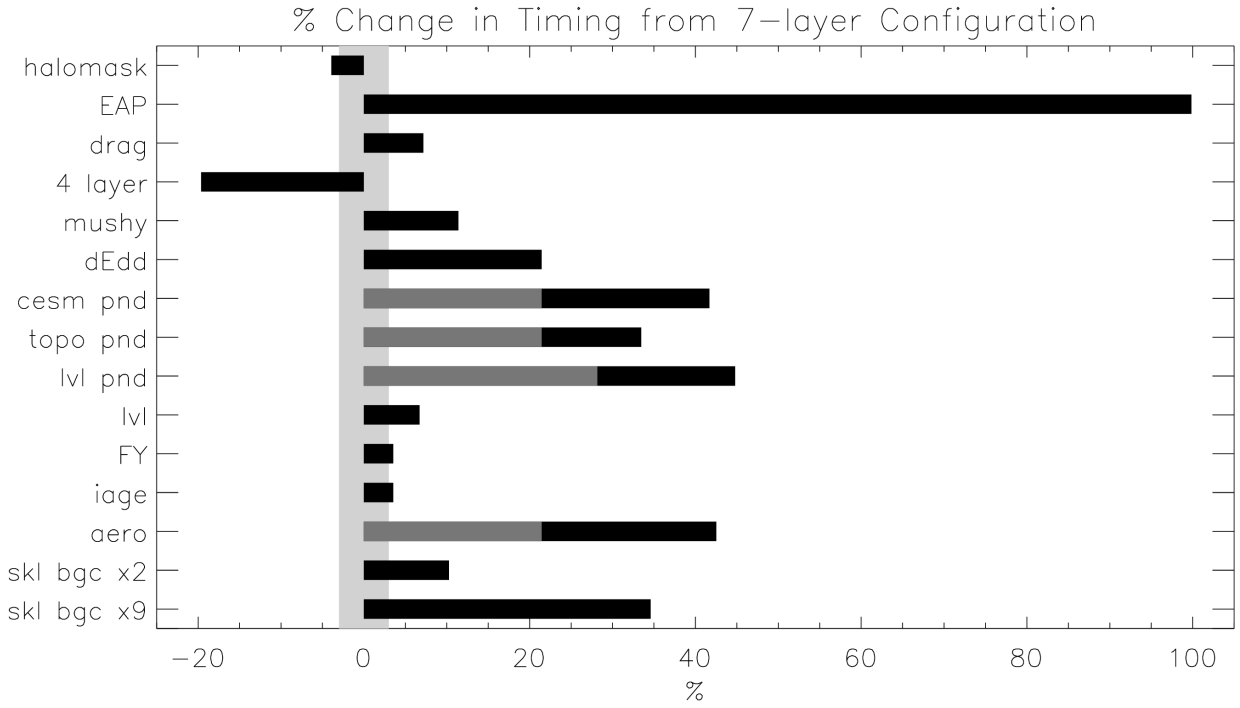


Fig. 5: Timings

Figure *Timings* shows change in 'TimeLoop' timings from the 7-layer configuration using BL99 thermodynamics and EVP dynamics. Timings were made on a nondedicated machine, with variations of about $\pm 3\%$ in identically configured runs (light grey). Darker grey indicates the time needed for extra required options; The Delta-Eddington radiation scheme is required for all melt pond schemes and the aerosol tracers, and the level-ice pond parameterization additionally requires the level-ice tracers.

### 3.1.3 Initialization and coupling

The ice model's parameters and variables are initialized in several steps. Many constants and physical parameters are set in **ice_constants.F90**. Namelist variables (*Table of namelist options*), whose values can be altered at run time, are handled in *input_data* and other initialization routines. These variables are given default values in the code, which

may then be changed when the input file **ice_in** is read. Other physical constants, numerical parameters, and variables are first set in initialization routines for each ice model component or module. Then, if the ice model is being restarted from a previous run, core variables are read and reinitialized in *restartfile*, while tracer variables needed for specific configurations are read in separate restart routines associated with each tracer or specialized parameterization. Finally, albedo and other quantities dependent on the initial ice state are set. Some of these parameters will be described in more detail in *Table of namelist options*.

The restart files supplied with the code release include the core variables on the default configuration, that is, with seven vertical layers and the ice thickness distribution defined by `kcatbound` = 0. Restart information for some tracers is also included in the netCDF restart files.

Three namelist variables control model initialization, `ice\_ic`, `runtype`, and `restart`, as described in *Ice Initial State*. It is possible to do an initial run from a file **filename** in two ways: (1) set runtype = 'initial', restart = true and ice_ic = **filename**, or (2) runtype = 'continue' and pointer_file = **./restart/ice.restart_file** where **./restart/ice.restart_file** contains the line "./restart/[filename]". The first option is convenient when repeatedly starting from a given file when subsequent restart files have been written. With this arrangement, the tracer restart flags can be set to true or false, depending on whether the tracer restart data exist. With the second option, tracer restart flags are set to 'continue' for all active tracers.

An additional namelist option, `restart\_ext` specifies whether halo cells are included in the restart files. This option is useful for tripole and regional grids, but can not be used with PIO.

MPI is initialized in *init_communicate* for both coupled and stand-alone MPI runs. The ice component communicates with a flux coupler or other climate components via external routines that handle the variables listed in the Icepack documentation. For stand-alone runs, routines in **ice_forcing.F90** read and interpolate data from files, and are intended merely to provide guidance for the user to write his or her own routines. Whether the code is to be run in stand-alone or coupled mode is determined at compile time, as described below.

Table *Ice Initial State* shows ice initial state resulting from combinations of `ice\_ic`, `runtype` and `restart`. [a]If false, restart is reset to true. [b]restart is reset to false. [c]ice_ic is reset to 'none.'

Table 1: Ice Initial State

| ice_ic | | | |
|---|---|---|---|
| | initial/false | initial/true | continue/true (or false[a]) |
| none | no ice | no ice[b] | restart using **pointer_file** |
| default | SST/latitude dependent | SST/latitude dependent[b] | restart using **pointer_file** |
| **filename** | no ice[c] | start from **filename** | restart using **pointer_file** |

### 3.1.4 Choosing an appropriate time step

The time step is chosen based on stability of the transport component (both horizontal and in thickness space) and on resolution of the physical forcing. CICE allows the dynamics, advection and ridging portion of the code to be run with a shorter timestep, $\Delta t_{dyn}$ (`dt\_dyn`), than the thermodynamics timestep $\Delta t$ (`dt`). In this case, `dt` and the integer ndtd are specified, and `dt\_dyn` = `dt/ndtd`.

A conservative estimate of the horizontal transport time step bound, or CFL condition, under remapping yields

$$\Delta t_{dyn} < \frac{\min\left(\Delta x, \Delta y\right)}{2\max\left(u, v\right)}.$$

Numerical estimates for this bound for several POP grids, assuming $\max(u, v) = 0.5$ m/s, are as follows:

Table 2: *Time Step Bound*

| grid label | N pole singularity | dimensions | min $\sqrt{\Delta x \cdot \Delta y}$ | max $\Delta t_{dyn}$ |
|---|---|---|---|---|
| gx3 | Greenland | $100 \times 116$ | $39 \times 10^3$ m | 10.8hr |
| gx1 | Greenland | $320 \times 384$ | $18 \times 10^3$ m | 5.0hr |
| p4 | Canada | $900 \times 600$ | $6.5 \times 10^3$ m | 1.8hr |

As discussed in [29], the maximum time step in practice is usually determined by the time scale for large changes in the ice strength (which depends in part on wind strength). Using the strength parameterization of [39], limits the time step to ∼30 minutes for the old ridging scheme (`krdg\_partic` = 0), and to ∼2 hours for the new scheme (`krdg\_partic` = 1), assuming $\Delta x$ = 10 km. Practical limits may be somewhat less, depending on the strength of the atmospheric winds.

Transport in thickness space imposes a similar restraint on the time step, given by the ice growth/melt rate and the smallest range of thickness among the categories, $\Delta t < \min(\Delta H)/2\max(f)$, where $\Delta H$ is the distance between category boundaries and $f$ is the thermodynamic growth rate. For the 5-category ice thickness distribution used as the default in this distribution, this is not a stringent limitation: $\Delta t < 19.4$ hr, assuming $\max(f) = 40$ cm/day.

In the classic EVP or EAP approach (`kdyn` = 1 or 2, `revised\_evp` = false), the dynamics component is subcycled ndte ($N$) times per dynamics time step so that the elastic waves essentially disappear before the next time step. The subcycling time step ($\Delta t_e$) is thus

$$dte = dt\_dyn/ndte.$$

A second parameter, $E_\circ$ (`eyc`), defines the elastic wave damping timescale $T$, described in Section *Dynamics*, as `eyc\ * dt\_dyn`. The forcing terms are not updated during the subcycling. Given the small step (`dte`) at which the EVP dynamics model is subcycled, the elastic parameter $E$ is also limited by stability constraints, as discussed in [16]. Linear stability analysis for the dynamics component shows that the numerical method is stable as long as the subcycling time step $\Delta t_e$ sufficiently resolves the damping timescale $T$. For the stability analysis we had to make several simplifications of the problem; hence the location of the boundary between stable and unstable regions is merely an estimate. In practice, the ratio $\Delta t_e : T : \Delta t = 1 : 40 : 120$ provides both stability and acceptable efficiency for time steps ($\Delta t$) on the order of 1 hour.

Note that only $T$ and $\Delta t_e$ figure into the stability of the dynamics component; $\Delta t$ does not. Although the time step may not be tightly limited by stability considerations, large time steps (*e.g.,* $\Delta t = 1$ day, given daily forcing) do not produce accurate results in the dynamics component. The reasons for this error are discussed in [16]; see [20] for its practical effects. The thermodynamics component is stable for any time step, as long as the surface temperature $T_{sfc}$ is computed internally. The numerical constraint on the thermodynamics time step is associated with the transport scheme rather than the thermodynamic solver.

For the revised EVP approach (`kdyn` = 1, `revised\_evp` = true), the relaxation parameter `arlx1i` effectively sets the damping timescale in the problem, and `brlx` represents the effective subcycling [4] (see Section *Revised approach*).

### 3.1.5 Model output

#### History files

Model output data is averaged over the period(s) given by `histfreq` and `histfreq\_n`, and written to binary or netCDF files prepended by `history\_file` in **ice_in**. These settings for history files are set in the **setup_nml** section of **ice_in** (see *Table of namelist options*). If `history\_file` = 'iceh' then the filenames will have the form **iceh.[timeID].nc** or **iceh.[timeID].da**, depending on the output file format chosen in **cice.settings** (set `ICE\_IOTYPE`). The netCDF history files are CF-compliant; header information for data contained in the netCDF files is displayed with the command `ncdump -h filename.nc`. Parallel netCDF output is available using the PIO library; the attribute `io\_flavor` distinguishes output files written with PIO from those written with standard

netCDF. With binary files, a separate header file is written with equivalent information. Standard fields are output according to settings in the **icefields_nml** section of **ice_in** (see *Table of namelist options*). The user may add (or subtract) variables not already available in the namelist by following the instructions in section *Adding History fields*.

The history module has been divided into several modules based on the desired formatting and on the variables themselves. Parameters, variables and routines needed by multiple modules is in **ice_history_shared.F90**, while the primary routines for initializing and accumulating all of the history variables are in **ice_history.F90**. These routines call format-specific code in the **io_binary**, **io_netcdf** and **io_pio** directories. History variables specific to certain components or parameterizations are collected in their own history modules (**ice_history_bgc.F90**, **ice_history_drag.F90**, **ice_history_mechred.F90**, **ice_history_pond.F90**).

The history modules allow output at different frequencies. Five output frequencies (`1`, `h`, `d`, `m`, `y`) are available simultaneously during a run. The same variable can be output at different frequencies (say daily and monthly) via its namelist flag, $f\_\ \langle var \rangle$, which is now a character string corresponding to `histfreq` or 'x' for none. (Grid variable flags are still logicals, since they are written to all files, no matter what the frequency is.) If there are no namelist flags with a given `histfreq` value, or if an element of `histfreq\_n` is 0, then no file will be written at that frequency. The output period can be discerned from the filenames.

For example, in the namelist:

```
``histfreq`` = '1', 'h', 'd', 'm', 'y'
``histfreq_n`` = 1, 6, 0, 1, 1
``f_hi`` = '1'
``f_hs`` = 'h'
``f_Tsfc`` = 'd'
``f_aice`` = 'm'
``f_meltb`` = 'mh'
``f_iage`` = 'x'
```

Here, `hi` will be written to a file on every timestep, `hs` will be written once every 6 hours, `aice` once a month, `meltb` once a month AND once every 6 hours, and `Tsfc` and `iage` will not be written.

From an efficiency standpoint, it is best to set unused frequencies in `histfreq` to 'x'. Having output at all 5 frequencies takes nearly 5 times as long as for a single frequency. If you only want monthly output, the most efficient setting is `histfreq` = 'm','x','x','x','x'. The code counts the number of desired streams (`nstreams`) based on `histfreq`.

The history variable names must be unique for netCDF, so in cases where a variable is written at more than one frequency, the variable name is appended with the frequency in files after the first one. In the example above, `meltb` is called `meltb` in the monthly file (for backward compatibility with the default configuration) and `meltb\_h` in the 6-hourly file.

Using the same frequency twice in `histfreq` will have unexpected consequences and currently will cause the code to abort. It is not possible at the moment to output averages once a month and also once every 3 months, for example.

If `write\_ic` is set to true in **ice_in**, a snapshot of the same set of history fields at the start of the run will be written to the history directory in **iceh_ic.[timeID].nc(da)**. Several history variables are hard-coded for instantaneous output regardless of the averaging flag, at the frequency given by their namelist flag.

The normalized principal components of internal ice stress are computed in *principal_stress* and written to the history file. This calculation is not necessary for the simulation; principal stresses are merely computed for diagnostic purposes and included here for the user's convenience.

Several history variables are available in two forms, a value representing an average over the sea ice fraction of the grid cell, and another that is multiplied by $a_i$, representing an average over the grid cell area. Our naming convention attaches the suffix "_ai" to the grid-cell-mean variable names.

Beginning with CICE v6, history variables requested by the Sea Ice Model Intercomparison Project (SIMIP) [33] have been added as possible history output variables (e.g. `f_sithick`, `f_sidmassgrowthbottom`, etc.). The lists of monthly and daily requested SIMIP variables provide the names of possible history fields in CICE. However, each

of the additional variables can be output at any temporal frequency specified in the **icefields_nml** section of **ice_in** as detailed above. Additionally, a new history output variable, `f_CMIP`, has been added. When `f_CMIP` is added to the **icefields_nml** section of **ice_in** then all SIMIP variables will be turned on for output at the frequency specified by `f_CMIP`.

### Diagnostic files

Like `histfreq`, the parameter `diagfreq` can be used to regulate how often output is written to a log file. The log file unit to which diagnostic output is written is set in **ice_fileunits.F90**. If `diag\_type` = 'stdout', then it is written to standard out (or to **ice.log.[ID]** if you redirect standard out as in **cice.run**); otherwise it is written to the file given by `diag\_file`. In addition to the standard diagnostic output (maximum area-averaged thickness, velocity, average albedo, total ice area, and total ice and snow volumes), the namelist options `print\_points` and `print\_global` cause additional diagnostic information to be computed and written. `print\_global` outputs global sums that are useful for checking global conservation of mass and energy. `print\_points` writes data for two specific grid points. Currently, one point is near the North Pole and the other is in the Weddell Sea; these may be changed in **ice_in**.

Timers are declared and initialized in **ice_timers.F90**, and the code to be timed is wrapped with calls to *ice_timer_start* and *ice_timer_stop*. Finally, *ice_timer_print* writes the results to the log file. The optional "stats" argument (true/false) prints additional statistics. Calling *ice_timer_print_all* prints all of the timings at once, rather than having to call each individually. Currently, the timers are set up as in *CICE timers*. Section *Adding Timers* contains instructions for adding timers.

The timings provided by these timers are not mutually exclusive. For example, the column timer (5) includes the timings from 6–10, and subroutine *bound* (timer 15) is called from many different places in the code, including the dynamics and advection routines.

The timers use *MPI_WTIME* for parallel runs and the F90 intrinsic *system_clock* for single-processor runs.

Table 3: CICE timers

| **Timer** | | |
|---|---|---|
| **Index** | **Label** | |
| 1 | Total | the entire run |
| 2 | Step | total minus initialization and exit |
| 3 | Dynamics | EVP |
| 4 | Advection | horizontal transport |
| 5 | Column | all vertical (column) processes |
| 6 | Thermo | vertical thermodynamics |
| 7 | Shortwave | SW radiation and albedo |
| 8 | Meltponds | melt ponds |
| 9 | Ridging | mechanical redistribution |
| 10 | Cat Conv | transport in thickness space |
| 11 | Coupling | sending/receiving coupler messages |
| 12 | ReadWrite | reading/writing files |
| 13 | Diags | diagnostics (log file) |
| 14 | History | history output |
| 15 | Bound | boundary conditions and subdomain communications |
| 16 | BGC | biogeochemistry |

### Restart files

CICE provides restart data in binary unformatted or netCDF formats, via the `ICE\_IOTYPE` flag in **cice.settings** and namelist variable `restart\_format`. Restart and history files must use the same format. As with the history

output, there is also an option for writing parallel netCDF restart files using PIO.

The restart files created by CICE contain all of the variables needed for a full, exact restart. The filename begins with the character string 'iced.', and the restart dump frequency is given by the namelist variables `dumpfreq` and `dumpfreq\_n`. The pointer to the filename from which the restart data is to be read for a continuation run is set in `pointer\_file`. The code assumes that auxiliary binary tracer restart files will be identified using the same pointer and file name prefix, but with an additional character string in the file name that is associated with each tracer set. All variables are included in netCDF restart files.

Additional namelist flags provide further control of restart behavior. `dump\_last` = true causes a set of restart files to be written at the end of a run when it is otherwise not scheduled to occur. The flag `use\_restart\_time` enables the user to choose to use the model date provided in the restart files. If `use\_restart\_time` = false then the initial model date stamp is determined from the namelist parameters. lcdf64 = true sets 64-bit netCDF output, allowing larger file sizes.

Routines for gathering, scattering and (unformatted) reading and writing of the "extended" global grid, including the physical domain and ghost (halo) cells around the outer edges, allow exact restarts on regional grids with open boundary conditions, and they will also simplify restarts on the various tripole grids. They are accessed by setting `restart\_ext` = true in namelist. Extended grid restarts are not available when using PIO; in this case extra halo update calls fill ghost cells for tripole grids (do not use PIO for regional grids).

Two netCDF restart files are available for the CICE v5 and v6 code distributions for the gx3 and gx1 grids (see *Forcing data* for information about obtaining these files). They were created using the default v5 model configuration, but initialized with no ice. The gx3 case was run for 1 year using the 1997 forcing data provided with the code. The gx1 case was run for 20 years, so that the date of restart in the file is 1978-01-01. Note that the restart dates provided in the restart files can be overridden using the namelist variables `use\_restart\_time`, `year\_init` and `istep0`. The forcing time can also be overridden using `fyear\_init`.

## 3.2 Running CICE

Quick-start instructions are provided in the *Quick Start* section.

### 3.2.1 Software Requirements

To run stand-alone, CICE requires

- gmake (GNU Make)
- Fortran and C compilers (Intel, PGI, GNU, Cray, and NAG have been tested)
- NetCDF
- MPI (this is actually optional but without it you can only run on 1 processor)

Below are lists of software versions that the Consortium has tested at some point. There is no guarantee that all compiler versions work with all CICE model versions. At any given point, the Consortium is regularly testing on several different compilers, but not necessarily on all possible versions or combinations. A CICE goal is to be relatively portable across different hardware, compilers, and other software. As a result, the coding implementation tends to be on the conservative side at times. If there are problems porting to a particular system, please let the Consortium know.

The Consortium has tested the following compilers at some point,

- Intel 15.0.3.187
- Intel 16.0.1.150
- Intel 17.0.1.132

- Intel 17.0.2.174
- Intel 17.0.5.239
- Intel 18.0.1.163
- Intel 19.0.2
- Intel 19.0.3.199
- PGI 16.10.0
- GNU 6.3.0
- GNU 7.2.0
- GNU 7.3.0
- Cray 8.5.8
- Cray 8.6.4
- NAG 6.2

The Consortium has tested the following mpi versions,

- MPICH 7.3.2
- MPICH 7.5.3
- MPICH 7.6.2
- MPICH 7.6.3
- MPICH 7.7.6
- Intel MPI 18.0.1
- MPT 2.14
- MPT 2.17
- MPT 2.18
- MPT 2.19
- OpenMPI 1.6.5

The NetCDF implementation is relatively general and should work with any version of NetCDF 3 or 4. The Consortium has tested

- NetCDF 4.3.0
- NetCDF 4.3.2
- NetCDF 4.4.0
- NetCDF 4.4.1.1.32
- NetCDF 4.4.1.1
- NetCDF 4.4.2
- NetCDF 4.5.0
- NetCDF 4.6.1.3

Please email the Consortium if this list can be extended.

### 3.2.2 Scripts

The CICE scripts are written to allow quick setup of cases and tests. Once a case is generated, users can manually modify the namelist and other files to custom configure the case. Several settings are available via scripts as well.

#### Overview

Most of the scripts that configure, build and run CICE are contained in the directory **configuration/scripts/**, except for **cice.setup**, which is in the main directory. **cice.setup** is the main script that generates a case.

Users may need to port the scripts to their local machine. Specific instructions for porting are provided in *Porting*.

`cice.setup -h` will provide the latest information about how to use the tool. `cice.setup --help` will provide an extended version of the help. There are three usage modes,

- `--case` or `-c` creates individual stand alone cases.
- `--test` creates individual tests. Tests are just cases that have some extra automation in order to carry out particular tests such as exact restart.
- `--suite` creates a test suite. Test suites are predefined sets of tests and `--suite` provides the ability to quickly setup, build, and run a full suite of tests.

All modes will require use of `--mach` or `-m` to specify the machine and case and test modes can use `--set` or `-s` to define specific options. `--test` and `--suite` will require `--testid` to be set and both of the test modes can use `--bdir`, `--bgen`, `--bcmp`, and `--diff` to generate (save) results and compare results with prior results as well as `--tdir` to specify the location of the test directory. Testing will be described in greater detail in the *Testing CICE* section.

Again, `cice.setup --help` will show the latest usage information including the available `--set` options, the current ported machines, and the test choices.

To create a case, run **cice.setup**:

```
cice.setup -c mycase -m machine
cd mycase
```

Once a case/test is created, several files are placed in the case directory

- **env.[machine]** defines the environment
- **cice.settings** defines many variables associated with building and running the model
- **makdep.c** is a tool that will automatically generate the make dependencies
- **Macros.[machine]** defines the Makefile macros
- **Makefile** is the makefile used to build the model
- **cice.build** is a script that builds and compiles the model
- **ice_in** is the namelist input file
- **setup_run_dirs.csh** is a script that will create the run directories. This will be called automatically from the **cice.run** script if the user does not invoke it.
- **cice.run** is a batch run script
- **cice.submit** is a simple script that submits the cice.run script

Once the case is created, all scripts and namelist are fully resolved. Users can edit any of the files in the case directory manually to change the model configuration, build options, or batch settings. The file dependency is indicated in the above list. For instance, if any of the files before **cice.build** in the list are edited, **cice.build** should be rerun.

The **casescripts/** directory holds scripts used to create the case and can largely be ignored. Once a case is created, the **cice.build** script should be run interactively and then the case should be submitted by executing the **cice.submit** script interactively. The **cice.submit** script simply submits the **cice.run script**. You can also submit the **cice.run** script on the command line.

Some hints:

- To change the block sizes required at build time, edit the **cice.settings** file.
- To change namelist, manually edit the **ice_in** file
- To change batch settings, manually edit the top of the **cice.run** or **cice.test** (if running a test) file
- To turn on the debug compiler flags, set `ICE_BLDDEBUG` in **cice.setttings** to true. It is also possible to use the `debug` option (`-s debug`) when creating the case with **cice.setup** to set this option automatically.
- To change compiler options, manually edit the Macros file
- To clean the build before each compile, set `ICE_CLEANBUILD` in **cice.settings** to true (this is the default value), or use the `buildclean` option (`-s buildclean`) when creating the case with **cice.setup**. To not clean before the build, set `ICE_CLEANBUILD` in **cice.settings** to false, or use the `buildincremental` option (`-s buildincremental`) when creating the case with **cice.setup**. It is recommended that the `ICE_CLEANBUILD` be set to true if there are any questions about whether the build is proceeding properly.

To build and run:

```
./cice.build
./cice.submit
```

The build and run log files will be copied into the logs directory in the case directory. Other model output will be in the run directory. The run directory is set in **cice.settings** via the `ICE_RUNDIR` variable. To modify the case setup, changes should be made in the case directory, NOT the run directory.

### Command Line Options

`cice.setup -h` provides a summary of the command line options. There are three different modes, `--case`, `--test`, and `--suite`. This section provides details about the relevant options for setting up cases with examples. Testing will be described in greater detail in the *Testing CICE* section.

**`--help, -h`** prints `cice.setup` help information to the terminal and exits.

**`--version`** prints the CICE version to the terminal and exits.

**`--setvers VERSION`** internally updates the CICE version in your sandbox. Those changes can then be commited (or not) to the repository. –version will show the updated value. The argument VERSION is typically a string like "5.1.2" but could be any alphanumeric string.

**`--case, -c` CASE** specifies the case name. This can be either a relative path of an absolute path. This cannot be used with –test or –suite. Either `--case`, `--test`, or `--suite` is required.

**`--mach, -m` MACHINE** specifies the machine name. This should be consistent with the name defined in the Macros and env files in **configurations/scripts/machines**. This is required in all modes.

**`--env, -e` ENVIRONMENT1,ENVIRONMENT2,ENVIRONMENT3** specifies the environment or compiler associated with the machine. This should be consistent with the name defined in the Macros and env files in **configurations/scripts/machines**. Each machine can have multiple supported environments including support for different compilers or other system setups. When used with `--suite` or `--test`, the ENVIRONMENT can be a set of comma deliminated values with no spaces and the tests will then be run for all of those environments. With `--case`, only one ENVIRONMENT should be specified. (default is intel)

**--pes, -p MxN[[xBXxBY[xMB]** specifies the number of tasks and threads the case should be run on. This only works with `--case`. The format is tasks x threads or "M"x"N" where M is tasks and N is threads and both are integers. BX, BY, and MB can also be set via this option where BX is the x-direction blocksize, BY is the y-direction blocksize, and MB is the max-blocks setting. If BX, BY, and MB are not set, they will be computed automatically based on the grid size and the task/thread count. More specifically, this option has three modes, –pes MxN, –pes MxNxBXxBY, and –pes MxNxBXxBYxMB. (default is 4x1)

**--acct ACCOUNT** specifies a batch account number. This is optional. See *Machine Account Settings* for more information.

**--grid, -g GRID** specifies the grid. This is a string and for the current CICE driver, gx1, gx3, and tx1 are supported. (default = gx3)

**--set, -s SET1,SET2,SET3** specifies the optional settings for the case. The settings for `--suite` are defined in the suite file. Multiple settings can be specified by providing a comma deliminated set of values without spaces between settings. The available settings are in **configurations/scripts/options** and `cice.setup --help` will also list them. These settings files can change either the namelist values or overall case settings (such as the debug flag).

For CICE, when setting up cases, the `--case` and `--mach` must be specified. It's also recommended that `--env` be set explicitly as well. `--pes` and `--grid` can be very useful. `--acct` is not normally used. A more convenient method is to use the **~/cice_proj** file, see *Machine Account Settings*. The `--set` option can be extremely handy. The `--set` options are documented in *Preset Options*.

### Preset Options

There are several preset options. These are hardwired in **configurations/scripts/options** and are specfied for a case or test by the `--set` command line option. You can see the full list of settings by doing `cice.setup --help`.

The default CICE namelist and CICE settings are specified in the files **configuration/scripts/ice_in** and **configuration/scripts/cice.settings** respectively. When picking settings (options), the set_env.setting and set_nml.setting will be used to change the defaults. This is done as part of the `cice.setup` and the modifications are resolved in the **cice.settings** and **ice_in** file placed in the case directory. If multiple options are chosen and then conflict, then the last option chosen takes precedent. Not all options are compatible with each other.

Some of the options are

`debug` which turns on the compiler debug flags

`buildclean` which turns on the option to clean the build before each compile

`buildincremental` which turns off the option to clean the build before each compile

`short`, `medium`, `long` which change the batch time limit

`gx3`, `gx1`, `tx1` are associate with grid specific settings

`diag1` which turns on diagnostics each timestep

`run10day`, `run1year`, etc which specifies a run length

`dslenderX1`, `droundrobin`, `dspacecurve`, etc specify decomposition options

`bgcISPOL` and `bgcNICE` specify bgc options

`boxadv`, `boxdyn`, and `boxrestore` are simple box configurations

`alt*` which turns on various combinations of dynamics and physics options for testing

and there are others. These may change as needed. Use `cice.setup --help` to see the latest. To add a new option, just add the appropriate file in **configuration/scripts/options**. For more information, see *Test Options*

#### Examples

The simplest case is just to setup a default configuration specifying the case name, machine, and environment:

```
cice.setup --case mycase1 --mach spirit --env intel
```

To add some optional settings, one might do:

```
cice.setup --case mycase2 --mach spirit --env intel --set debug,diag1,run1year
```

Once the cases are created, users are free to modify the cice.settings and ice_in namelist to further modify their setup.

### 3.2.3 Porting

To port, an **env.[machine]_[environment]** and **Macros.[machine]_[environment]** file have to be added to the **configuration/scripts/machines/** directory and the **configuration/scripts/cice.batch.csh** file needs to be modified. In general, the machine is specified in `cice.setup` with `--mach` and the environment (compiler) is specified with `--env`.

- cd to **configuration/scripts/machines/**
- Copy an existing env and a Macros file to new names for your new machine
- Edit your env and Macros files
- cd .. to **configuration/scripts/**
- Edit the **cice.batch.csh** script to add a section for your machine with batch settings and job launch settings
- Download and untar a forcing dataset to the location defined by `ICE_MACHINE_INPUTDATA` in the env file

In fact, this process almost certainly will require some iteration. The easiest way to carry this out is to create an initial set of changes as described above, then create a case and manually modify the **env.[machine]** file and **Macros.[machine]** file until the case can build and run. Then copy the files from the case directory back to **configuration/scripts/machines/** and update the **configuration/scripts/cice.batch.csh** file, retest, and then add and commit the updated machine files to the repository.

#### Cross-compiling

It can happen that the model must be built on a platform and run on another, for example when the run environment is only available in a batch queue. The program **makdep** (see *Overview*), however, is both compiled and run as part of the build process.

In order to support this, the Makefile uses a variable `CFLAGS_HOST` that can hold compiler flags specfic to the build machine for the compilation of makdep. If this feature is needed, add the variable `CFLAGS_HOST` to the **Macros.[machine]_[environment]** file. For example :

```
CFLAGS_HOST = -xHost
```

#### Machine Account Settings

The machine account default is specified by the variable `ICE_MACHINE_ACCT` in the **env.[machine]** file. The easiest way to change a user's default is to create a file in your home directory called **.cice_proj** and add your preferred account name to the first line. There is also an option (`--acct`) in **cice.setup** to define the account number. The order of precedent is **cice.setup** command line option, **.cice_proj** setting, and then value in the **env.[machine]** file.

#### Machine Queue Settings

Supported machines will have a default queue specified by the variable `ICE_MACHINE_QUEUE` in the **env.[machine]** file. This can also be manually changed in the **cice.run** or **cice.test** scripts or even better, use the `--queue` option in **cice.setup**.

### 3.2.4 Forcing data

The input data space is defined on a per machine basis by the `ICE_MACHINE_INPUTDATA` variable in the **env.[machine]** file. That file space is often shared among multiple users, and it can be desirable to consider using a common file space with group read and write permissions such that a set of users can update the inputdata area as new datasets are available.

CICE input datasets are stored on an anonymous ftp server. More information about how to download the input data can be found at https://github.com/CICE-Consortium/CICE/wiki. Test forcing datasets are available for various grids at the ftp site. These data files are designed only for testing the code, not for use in production runs or as observational data. Please do not publish results based on these data sets.

### 3.2.5 Run Directories

The **cice.setup** script creates a case directory. However, the model is actually built and run under the `ICE_OBJDIR` and `ICE_RUNDIR` directories as defined in the **cice.settings** file.

Build and run logs will be copied from the run directory into the case **logs/** directory when complete.

### 3.2.6 Local modifications

Scripts and other case settings can be changed manually in the case directory and used. Source code can be modified in the main sandbox. When changes are made, the code should be rebuilt before being resubmitted. It is always recommended that users modify the scripts and input settings in the case directory, NOT the run directory. In general, files in the run directory are overwritten by versions in the case directory when the model is built, submitted, and run.

## 3.3 Testing CICE

This section documents primarily how to use the CICE scripts to carry out CICE testing. Exactly what to test is a separate question and depends on the kinds of code changes being made. Prior to merging changes to the CICE Consortium master, changes will be reviewed and developers will need to provide a summary of the tests carried out.

There is a base suite of tests provided by default with CICE and this may be a good starting point for testing.

The testing scripts support several features

- Ability to test individual (via `--test`) or multiple tests (via `--suite`) using an input file to define the suite
- Ability to use test suites defined in the package or test suites defined by the user
- Ability to store test results for regresssion testing (`--bgen`)
- Ability to compare results to prior baselines to verify bit-for-bit (`--bcmp`)
- Ability to define where baseline tests are stored (`--bdir`)
- Ability to compare tests against each other (`--diff`)
- Ability to set account number (`--acct`), which is otherwise not set and may result in tests not being submitted

### 3.3.1 Individual Tests

The CICE scripts support both setup of individual tests as well as test suites. Individual tests are run from the command line:

```
./cice.setup --test smoke --mach conrad --env cray --set diag1,debug --testid myid
```

Tests are just like cases but have some additional scripting around them. Individual tests can be created and manually modified just like cases. Many of the command line arguments for individual tests are similar to *Command Line Options* for `--case`. For individual tests, the following command line options can be set

**`--test` TESTNAME** specifies the test type. This is probably either smoke or restart but see *cice.setup –help* for the latest. This is required instead of `--case`.

**`--testid` ID** specifies the testid. This is required for every use of `--test` and `--suite`. This is a user defined string that will allow each test to have a unique case and run directory name. This is also required.

**`--tdir` PATH** specifies the test directory. Testcases will be created in this directory. (default is .)

`--mach` MACHINE (see *Command Line Options*)

`--env` ENVIRONMENT1 (see *Command Line Options*)

`--set` SET1,SET2,SET3 (see *Command Line Options*)

`--acct` ACCOUNT (see *Command Line Options*)

`--grid` GRID (see *Command Line Options*)

`--pes` MxNxBXxBYxMB (see *Command Line Options*)

There are several additional options that come with `--test` that are not available with `--case` for regression and comparision testing,

**`--bdir` DIR** specifies the top level location of the baseline results. This is used in conjuction with `--bgen` and `--bcmp`. The default is set by ICE_MACHINE_BASELINE in the env.[machine]_[environment] file.

**`--bgen` DIR** specifies the name of the directory under [bdir] where test results will be stored. When this flag is set, it automatically creates that directory and stores results from the test under that directory. If DIR is set to `default`, then the scripts will automatically generate a directory name based on the CICE hash and the date and time. This can be useful for tracking the baselines by hash.

**`--bcmp` DIR** specifies the name of the directory under [bdir] that the current tests will be compared to. When this flag is set, it automatically invokes regression testing and compares results from the current test to those prior results. If DIR is set to `default`, then the script will automatically generate the last directory name in the [bdir] directory. This can be useful for automated regression testing.

**`--diff` LONG_TESTNAME** invokes a comparison against another local test. This allows different tests to be compared to each other for bit-for-bit-ness. This is different than `--bcmp`. `--bcmp` is regression testing, comparing identical test results between different model versions. `--diff` allows comparison of two different test cases against each other. For instance, different block sizes, decompositions, and other model features are expected to produced identical results and `--diff` supports that testing. The restrictions for use of `--diff` are that the test has to already be completed and the testid has to match. The LONG_TESTNAME string should be of format [test]_[grid]_[pes]_[sets]. The [machine], [env], and [testid] will be added to that string to complete the testname being compared. (See also *Individual Test Examples* #5)

The format of the case directory name for a test will always be `[machine]_[env]_[test]_[grid]_[pes]_[sets].[testid]` The [sets] will always be sorted alphabetically by the script so `--set debug,diag1` and `--set diag1,debug` produces the same testname and test with _debug_diag1 in that order.

To build and run a test after invoking the ./cice.setup command, the process is the same as for a case. cd to the test directory, run the build script, and run the submit script:

```
cd [test_case]
./cice.build
./cice.submit
```

The test results will be generated in a local file called **test_output**. To check those results:

```
cat test_output
```

Tests are defined under **configuration/scripts/tests/**. Some tests currently supported are:

- **smoke - Runs the model for default length. The length and options can** be set with the `--set` command line option. The test passes if the model completes successfully.
- **restart - Runs the model for 10 days, writing a restart file at the end of day 5 and** again at the end of the run. Runs the model a second time starting from the day 5 restart and writes a restart at then end of day 10 of the model run. The test passes if both runs complete and if the restart files at the end of day 10 from both runs are bit-for-bit identical.
- decomp - Runs a set of different decompositions on a given configuration

Please run `./cice.setup --help` for the latest information.

### Adding a new test

See *Test scripts*

### Individual Test Examples

1) **Basic default single test**

   Define the test, mach, env, and testid.

```
./cice.setup --test smoke --mach wolf --env gnu --testid t00
cd wolf_gnu_smoke_col_1x1.t00
./cice.build
./cice.submit
./cat test_output
```

2) **Simple test with some options**

   Add `--set`

```
./cice.setup --test smoke --mach wolf --env gnu --set diag1,debug --testid t00
cd wolf_gnu_smoke_col_1x1_debug_diag1.t00
./cice.build
./cice.submit
./cat test_output
```

3) **Single test, generate a baseline dataset**

   Add `--bgen`

```
./cice.setup --test smoke --mach wolf -env gnu --bgen cice.v01 --testid t00 --set␣
↪diag1
cd wolf_gnu_smoke_col_1x1_diag1.t00
./cice.build
./cice.submit
./cat test_output
```

4) **Single test, compare results to a prior baseline**

Add `--bcmp`. For this to work, the prior baseline must exist and have the exact same base testname [machine]_[env]_[test]_[grid]_[pes]_[sets]

```
./cice.setup --test smoke --mach wolf -env gnu --bcmp cice.v01 --testid t01 --set␣
↪diag1
cd wolf_gnu_smoke_col_1x1_diag1.t01
./cice.build
./cice.submit
./cat test_output
```

5) **Simple test, generate a baseline dataset and compare to a prior baseline**

Use `--bgen` and `--bcmp`. The prior baseline must exist already.

```
./cice.setup --test smoke --mach wolf -env gnu --bgen cice.v02 --bcmp cice.v01 --
↪testid t02 --set diag1
cd wolf_gnu_smoke_col_1x1_diag1.t02
./cice.build
./cice.submit
./cat test_output
```

6) **Simple test, comparison against another test**

`--diff` provides a way to compare tests with each other. For this to work, the tests have to be run in a specific order and the testids need to match. The test is always compared relative to the current case directory.

To run the first test,

```
./cice.setup --test smoke --mach wolf -env gnu --testid tx01 --set debug
cd wolf_gnu_smoke_col_1x1_debug.tx01
./cice.build
./cice.submit
./cat test_output
```

Then to run the second test and compare to the results from the first test

```
./cice.setup --test smoke --mach wolf -env gnu --testid tx01 --diff smoke_col_1x1_
↪debug
cd wolf_gnu_smoke_col_1x1.tx01
./cice.build
./cice.submit
./cat test_output
```

The scripts will add a [machine]_[environment] to the beginning of the diff argument and the same testid to the end of the diff argument. Then the runs will be compared for bit-for-bit and a result will be produced in test_output.

### Specific Test Cases

In addition to the test implemented in the general testing framework, specific tests have been developed to validate specific portions of the model. These specific tests are detailed in this section.

### box2001

The box2001 test case is configured to perform the rectangular-grid box test detailed in [15]. It is configured to run a 72-hour simulation with thermodynamics disabled in a rectangular domain (80 x 80 grid cells) with a land boundary around the entire domain. It includes the following namelist modifications:

- dxrect: 16.e5 cm
- dyrect: 16.e5 cm
- ktherm: -1 (disables thermodynamics)
- coriolis: zero (zero coriolis force)
- ice_data_type : box2001 (special ice concentration initialization)
- atm_data_type : box2001 (special atmospheric and ocean forcing)

Ocean stresses are computed as in [15] where they are circular and centered in the square domain. The ice distribution is fixed, with a constant 2 meter ice thickness and a concentration field that varies linearly in the x-direction from 0 to 1 and is constant in the y-direction. No islands are included in this configuration. The test is configured to run on a single processor.

To run the test:

```
./cice.setup -m <machine> --test smoke -s box2001 --testid <test_id> --grid gbox80 --
↪acct <queue manager account> -p 1x1
```

### boxslotcyl

The boxslotcyl test case is an advection test configured to perform the slotted cylinder test detailed in [51]. It is configured to run a 12-day simulation with thermodynamics, ridging and dynamics disabled, in a square domain (80 x 80 grid cells) with a land boundary around the entire domain. It includes the following namelist modifications:

- dxrect: 10.e5 cm (10 km)
- dyrect: 10.e5 cm (10 km)
- ktherm: -1 (disables thermodynamics)
- kridge: -1 (disables ridging)
- kdyn: -1 (disables dynamics)
- ice_data_type : boxslotcyl (special ice concentration and velocity initialization)

Dynamics is disabled because we directly impose a constant ice velocity. The ice velocity field is circular and centered in the square domain, and such that the slotted cylinder makes a complete revolution with a period $T = 12$ days :

$$(u, v) = u_0 \left( \frac{2y - L}{L}, \frac{-2x + L}{L} \right) \tag{3.1}$$

where $L$ is the physical domain length and $u_0 = \pi L/T$. The initial ice distribution is a slotted cylinder of radius $r = 3L/10$ centered at $(x, y) = (L/2, 3L/4)$. The slot has a width of $L/6$ and a depth of $5L/6$ and is placed radially.

The time step is one hour, which with the above speed and mesh size yields a Courant number of 0.86.

The test can run on multiple processors.

To run the test:

```
./cice.setup -m <machine> --test smoke -s boxslotcyl --testid <test_id> --grid gbox80␣
↪--acct <queue manager account> -p nxm
```

### 3.3.2 Test suites

Test suites support running multiple tests specified via an input file. When invoking the test suite option (`--suite`) with **cice.setup**, all tests will be created, built, and submitted automatically under a local directory called test-suite.[testid] as part of involing the suite.:

```
./cice.setup --suite base_suite --mach wolf --env gnu --testid myid
```

Like an individual test, the `--testid` option must be specified and can be any string. Once the tests are complete, results can be checked by running the results.csh script in the [suite_name].[testid]:

```
cd testsuite.[testid]
./results.csh
```

To report the test results, as is required for Pull Requests to be accepted into the master the CICE Consortium code see *Test Reporting*.

If using the `--tdir` option, that directory must not exist before the script is run. The tdir directory will be created by the script and it will be populated by all tests as well as scripts that support the test suite:

```
./cice.setup --suite base_suite --mach wolf --env gnu --testid myid --tdir /scratch/
↪$user/testsuite.myid
```

Multiple suites are supported on the command line as comma separated arguments:

```
./cice.setup --suite base_suite,decomp_suite --mach wolf --env gnu --testid myid
```

If a user adds `--set` to the suite, all tests in that suite will add that option:

```
./cice.setup --suite base_suite,decomp_suite --mach wolf --env gnu --testid myid -s␣
↪debug
```

The option settings defined in the suite have precendent over the command line values if there are conflicts.

The predefined test suites are defined under **configuration/scripts/tests** and the files defining the suites have a suffix of .ts in that directory. The format for the test suite file is relatively simple. It is a text file with white space delimited columns that define a handful of values in a specific order. The first column is the test name, the second the grid, the third the pe count, the fourth column is the `--set` options and the fifth column is the `--diff` argument. The fourth and fifth columns are optional. Lines that begin with # or are blank are ignored. For example,

```
#Test   Grid  PEs  Sets                Diff
 smoke   col  1x1  diag1
 smoke   col  1x1  diag1,run1year  smoke_col_1x1_diag1
 smoke   col  1x1  debug,run1year
restart  col  1x1  debug
restart  col  1x1  diag1
restart  col  1x1  pondcesm
restart  col  1x1  pondlvl
restart  col  1x1  pondtopo
```

The argument to `--suite` defines the test suite (.ts) filename and that argument can contain a path. **cice.setup** will look for the filename in the local directory, in **configuration/scripts/tests/**, or in the path defined by the `--suite` option.

Because many of the command line options are specified in the input file, ONLY the following options are valid for suites,

**--suite filename** required, input filename with list of suites

**--mach MACHINE** required

**--env ENVIRONMENT1,ENVIRONMENT2** strongly recommended

**--set SET1,SET2** optional

**--acct ACCOUNT** optional

**--tdir PATH** optional

**--testid ID** required

**--bdir DIR** optional, top level baselines directory and defined by default by ICE_MACHINE_BASELINE in **env.[machine]_[environment]**.

**--bgen DIR** recommended, test output is copied to this directory under [bdir]

**--bcmp DIR** recommended, test output are compared to prior results in this directory under [bdir]

**--report** This is only used by `--suite` and when set, invokes a script that sends the test results to the results page when all tests are complete. Please see *Test Reporting* for more information.

Please see *Command Line Options* and *Individual Tests* for more details about how these options are used.

### Test Suite Examples

1) **Basic test suite**

   Specify suite, mach, env, testid.

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

2) **Basic test suite with user defined test directory**

   Specify suite, mach, env, testid, tdir.

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪tdir /scratch/$user/ts.v01a
cd /scratch/$user/ts.v01a
# wait for runs to complete
./results.csh
```

3) **Basic test suite on multiple environments**

   Specify multiple envs.

```
./cice.setup --suite base_suite --mach conrad --env cray,pgi,intel,gnu --
↪testid v01a
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

   Each env can be run as a separate invokation of *cice.setup* but if that approach is taken, it is recommended that different testids be used.

4) **Basic test suite with generate option defined**

Add `--set`

```
 ./cice.setup --suite base_suite --mach conrad --env gnu --testid v01b --
↪set diag1
 cd testsuite.v01b
 # wait for runs to complete
./results.csh
```

If there are conflicts between the `--set` options in the suite and on the command line, the suite will take precedent.

5) **Multiple test suites from a single command line**

Add comma delimited list of suites

```
./cice.setup --suite base_suite,decomp_suite --mach conrad --env gnu --
↪testid v01c
cd testsuite.v01c
# wait for runs to complete
./results.csh
```

If there are redundant tests in multiple suites, the scripts will understand that and only create one test.

6) **Basic test suite, store baselines in user defined name**

Add `--bgen`

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen cice.v01a
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

This will store the results in the default [bdir] directory under the subdirectory cice.v01a.

7) **Basic test suite, store baselines in user defined top level directory**

Add `--bgen` and `--bdir`

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen cice.v01a --bdir /tmp/user/CICE_BASELINES
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

This will store the results in /tmp/user/CICE_BASELINES/cice.v01a.

8) **Basic test suite, store baselines in auto-generated directory**

Add `--bgen default`

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a --
↪bgen default
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

This will store the results in the default [bdir] directory under a directory name generated by the script that includes the hash and date.

9) **Basic test suite, compare to prior baselines**

Add `--bcmp`

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v02a --
↪bcmp cice.v01a
cd testsuite.v02a
# wait for runs to complete
./results.csh
```

This will compare to results saved in the baseline [bdir] directory under the subdirectory cice.v01a. With the `--bcmp` option, the results will be tested against prior baselines to verify bit-for-bit, which is an important step prior to approval of many (not all, see *Code Compliance Test (non bit-for-bit validation)*) Pull Requests to incorporate code into the CICE Consortium master code. You can use other regression options as well. (`--bdir` and `--bgen`)

10) **Basic test suite, use of default string in regression testing**

default is a special argument to `--bgen` and `--bcmp`. When used, the scripts will automate generation of the directories. In the case of `--bgen`, a unique directory name consisting of the hash and a date will be created. In the case of `--bcmp`, the latest directory in [bdir] will automatically be used. This provides a number of useful features

- the `--bgen` directory will be named after the hash automatically
- the `--bcmp` will always find the most recent set of baselines
- the `--bcmp` reporting will include information about the comparison directory name which will include hash information
- automation can be invoked easily, especially if `--bdir` is used to create separate baseline directories as needed.

Imagine the case where the default settings are used and `--bdir` is used to create a unique location. You could easily carry out regular builds automatically via,

```
set mydate = `date -u "+%Y%m%d"`
git clone https://github.com/myfork/cice cice.$mydate --recursive
cd cice.$mydate
./cice.setup --suite base_suite --mach conrad --env cray,gnu,intel,
↪pgi --testid $mydate --bcmp default --bgen default --bdir /tmp/
↪work/user/CICE_BASELINES_MASTER
```

When this is invoked, a new set of baselines will be generated and compared to the prior results each time without having to change the arguments.

11) **Reusing a test suite**

Add the buildincremental option (`-s buildincremental`). This permits the suite to be rerun without recompiling the whole code.

```
./cice.setup --suite base_suite --mach conrad --env intel --testid␣
↪v01b --set buildincremental
cd testsuite.v01b
# wait for runs to complete
./results.csh
# modify code
./suite.submit # or ./suite.run to run the suite interactively
# wait for runs to complete
./results.csh
```

Only modified files will be recompiled, and the suite will be rerun.

12) **Create and test a custom suite**

**Create your own input text file consisting of 5 columns of data,**

- Test
- Grid
- pes
- sets (optional)
- diff test (optional)

such as

```
> cat mysuite
smoke    col  1x1  diag1,debug
restart  col  1x1
restart  col  1x1  diag1,debug    restart_col_1x1
restart  col  1x1  mynewoption,diag1,debug
```

then use that input file, mysuite

```
./cice.setup --suite mysuite --mach conrad --env cray --testid v01a -
↪-bgen default
cd testsuite.v01a
# wait for runs to complete
./results.csh
```

You can use all the standard regression testing options (`--bgen`, `--bcmp`, `--bdir`). Make sure any "diff" testing that goes on is on tests that are created earlier in the test list, as early as possible. Unfortunately, there is still no absolute guarantee the tests will be completed in the correct sequence.

### 3.3.3 Test Reporting

The CICE testing scripts have the capability to post test results to the official CICE Consortium Test-Results wiki page. You may need write permission on the wiki. If you are interested in using the wiki, please contact the Consortium. Note that in order for code to be accepted to the CICE master through a Pull Request it is necessary for the developer to provide proof that their code passes relevant tests. This can be accomplished by posting the full results to the wiki, or by copying the testing summary to the Pull Request comments.

To post results, once a test suite is complete, run `results.csh` and `report_results.csh` from the suite directory,

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a
cd testsuite.v01a
#wait for runs to complete
./results.csh
./report_results.csh
```

The reporting can also be automated by adding `--report` to `cice.setup`

```
./cice.setup --suite base_suite --mach conrad --env cray --testid v01a --report
```

With `--report`, the suite will create all the tests, build and submit them, wait for all runs to be complete, and run the results and report_results scripts.

### 3.3.4 Code Compliance Test (non bit-for-bit validation)

A core tenet of CICE dycore and CICE innovations is that they must not change the physics and biogeochemistry of existing model configurations, notwithstanding obsolete model components. Therefore, alterations to existing CICE Consortium code must only fix demonstrable numerical or scientific inaccuracies or bugs, or be necessary to introduce new science into the code. New physics and biogeochemistry introduced into the model must not change model answers when switched off, and in that case CICEcore and CICE must reproduce answers bit-for-bit as compared to previous simulations with the same namelist configurations. This bit-for-bit requirement is common in Earth System Modeling projects, but often cannot be achieved in practice because model additions may require changes to existing code. In this circumstance, bit-for-bit reproducibility using one compiler may not be unachievable on a different computing platform with a different compiler. Therefore, tools for scientific testing of CICE code changes have been developed to accompany bit-for-bit testing. These tools exploit the statistical properties of simulated sea ice thickness to confirm or deny the null hypothesis, which is that new additions to the CICE dycore and CICE have not significantly altered simulated ice volume using previous model configurations. Here we describe the CICE testing tools, which are applies to output from five-year gx-1 simulations that use the standard CICE atmospheric forcing. A scientific justification of the testing is provided in [19]. The following sections follow [36].

#### Two-Stage Paired Thickness Test

The first quality check aims to confirm the null hypotheses $H_0 : \mu_d{=}0$ at every model grid point, given the mean thickness difference $\mu_d$ between paired CICE simulations '$a$' and '$b$' that should be identical. $\mu_d$ is approximated as $\bar{h}_d = \frac{1}{n}\sum_{i=1}^{n}(h_{ai}{-}h_{bi})$ for $n$ paired samples of ice thickness $h_{ai}$ and $h_{bi}$ in each grid cell of the gx-1 mesh. Following [50], the associated $t$-statistic expects a zero mean, and is therefore

$$t = \frac{\bar{h}_d}{\sigma_d/\sqrt{n_{eff}}} \tag{3.2}$$

given variance $\sigma_d^2 = \frac{1}{n-1}\sum_{i=1}^{n}(h_{di} - \bar{h}_d)^2$ of $h_{di}{=}(h_{ai}{-}h_{bi})$ and effective sample size

$$n_{eff}{=}n\frac{(1-r_1)}{(1+r_1)} \tag{3.3}$$

for lag-1 autocorrelation:

$$r_1 = \frac{\sum_{i=1}^{n-1}\left[(h_{di} - \bar{h}_{d1:n-1})(h_{di+1} - \bar{h}_{d2:n})\right]}{\sqrt{\sum_{i=1}^{n-1}(h_{di} - \bar{h}_{d1:n-1})^2 \sum_{i=2}^{n}(h_{di} - \bar{h}_{d2:n})^2}}. \tag{3.4}$$

Here, $\bar{h}_{d1:n-1}$ is the mean of all samples except the last, and $\bar{h}_{d2:n}$ is the mean of samples except the first, and both differ from the overall mean $\bar{h}_d$ in equations ((3.2)). That is:

$$\bar{h}_{d1:n-1} = \frac{1}{n-1}\sum_{i=1}^{n-1} h_{di}, \quad \bar{h}_{d2:n} = \frac{1}{n-1}\sum_{i=2}^{n} h_{di}, \quad \bar{h}_d = \frac{1}{n}\sum_{i=1}^{n} h_{di} \tag{3.5}$$

Following [52], the effective sample size is limited to $n_{eff} \in [2, n]$. This definition of $n_{eff}$ assumes ice thickness evolves as an AR(1) process [46], which can be justified by analyzing the spectral density of daily samples of ice thickness from 5-year records in CICE Consortium member models [19]. The AR(1) approximation is inadmissible for paired velocity samples, because ice drift possesses periodicity from inertia and tides [14][26][37]. Conversely, tests of paired ice concentration samples may be less sensitive to ice drift than ice thickness. In short, ice thickness is the best variable for CICE Consortium quality control (QC), and for the test of the mean in particular.

Care is required in analyzing mean sea ice thickness changes using ((3.2)) with $N{=}n_{eff}{-}1$ degrees of freedom. [52] demonstrate that the $t$-test in ((3.2)) becomes conservative when $n_{eff} < 30$, meaning that $H_0$ may be erroneously

confirmed for highly auto-correlated series. Strong autocorrelation frequently occurs in modeled sea ice thickness, and $r_1 > 0.99$ is possible in parts of the gx-1 domain for the five-year QC simulations. In the event that $H_0$ is confirmed but $2 \leq n_{eff} < 30$, the $t$-test progresses to the 'Table Lookup Test' of [52], to check that the first-stage test using ((3.2)) was not conservative. The Table Lookup Test chooses critical $t$ values $|t| < t_{crit}(1-\alpha/2, N)$ at the $\alpha$ significance level based on $r_1$. It uses the conventional $t = \bar{h}_d\sqrt{n}/\sigma_d$ statistic with degrees of freedom $N{=}n{-}1$, but with $t_{crit}$ values generated using the Monte Carlo technique described in [52], and summarized in *Two-sided t_{crit} values* for 5-year QC simulations ($N = 1824$) at the two-sided 80% confidence interval ($\alpha = 0.2$). We choose this interval to limit Type II errors, whereby a QC test erroneously confirms $H_0$.

Table *Two-sided t_{crit} values* shows the summary of two-sided $t_{crit}$ values for the Table Lookup Test of [52] at the 80% confidence interval generated for $N = 1824$ degrees of freedom and lag-1 autocorrelation $r_1$.

Table 4: Two-sided $t_{crit}$ values

| $r_1$ | -0.05 | 0.0 | 0.2 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.95 | 0.97 | 0.99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t_{crit}$ | 1.32 | 1.32 | 1.54 | 2.02 | 2.29 | 2.46 | 3.17 | 3.99 | 5.59 | 8.44 | 10.85 | 20.44 |

### Quadratic Skill Compliance Test

In addition to the two-stage test of mean sea ice thickness, we also check that paired simulations are highly correlated and have similar variance using a skill metric adapted from [43]. A general skill score applicable to Taylor diagrams takes the form

$$S_m = \frac{4(1+R)^m}{(\hat{\sigma}_f + 1/\hat{\sigma}_f)^2(1+R_0)^m} \tag{3.6}$$

where $m = 1$ for variance-weighted skill, and $m = 4$ for correlation-weighted performance, as given in equations (4) and (5) of [43], respectively. We choose $m = 2$ to balance the importance of variance and correlation reproduction in QC tests, where $\hat{\sigma}_f = \sigma_b/\sigma_a$ is the ratio of the standard deviations of simulations '$b$' and '$a$', respectively, and simulation '$a$' is the control. $R_0$ is the maximum possible correlation between two series for correlation coefficient $R$ calculated between respective thickness pairs $h_a$ and $h_b$. Bit-for-bit reproduction of previous CICE simulations means that perfect correlation is possible, and so $R_0 = 1$, giving the quadratic skill of run '$b$' relative to run '$a$':

$$S = \left[\frac{(1+R)(\sigma_a\sigma_b)}{({\sigma_a}^2 + {\sigma_b}^2)}\right]^2 \tag{3.7}$$

This provides a skill score between 0 and 1. We apply this $S$ metric separately to the northern and southern hemispheres of the gx-1 grid by area-weighting the daily thickness samples discussed in the Two-Stage Paired Thickness QC Test. The hemispheric mean thickness over a 5-year simulation for run '$a$' is:

$$\bar{h}_a = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J} W_j\, h_{a_{i,j}} \tag{3.8}$$

at time sample $i$ and grid point index $j$, with an equivalent equation for simulation '$b$'. $n$ is the total number of time samples (nominally $n = 1825$) and $J$ is the total number of grid points on the gx-1 grid. $W_j$ is the weight attributed to each grid point according to its area $A_j$, given as

$$W_j = \frac{A_j}{\sum_{j=1}^{J} A_j} \tag{3.9}$$

for all grid points within each hemisphere with one or more non-zero thicknesses in one or both sets of samples $h_{a_{i,j}}$ or $h_{b_{i,j}}$. The area-weighted variance for simulation '$a$' is:

$${\sigma_a}^2 = \frac{\hat{J}}{(n\,\hat{J}-1)}\sum_{i=1}^{n}\sum_{j=1}^{J} W_j\,(h_{a_{i,j}} - \bar{h}_a)^2 \tag{3.10}$$

where $\hat{J}$ is the number of non-zero $W_j$ weights, and $\sigma_b$ is calculated equivalently for run '$b$'. In this context, $R$ becomes a weighted correlation coefficient, calculated as

$$R = \frac{\mathrm{cov}(h_a, h_b)}{\sigma_a \, \sigma_b} \tag{3.11}$$

given the weighted covariance

$$\mathrm{cov}(h_a, h_b) = \frac{\hat{J}}{(n\,\hat{J} - 1)} \sum_{i=1}^{n} \sum_{j=1}^{J} W_j \, (h_{a_{i,j}} - \bar{h}_a)(h_{b_{i,j}} - \bar{h}_b). \tag{3.12}$$

Using equations ((3.7)) to ((3.12)), the skill score $S$ is calculated separately for the northern and southern hemispheres, and must exceed a critical value nominally set to $S_{crit} = 0.99$ to pass the test. Practical illustrations of this test and the Two-Stage test described in the previous section are provided in [19].

### Code Compliance Testing Procedure

The CICE code compliance test is performed by running a python script (**configurations/scripts/tests/QC/cice.t-test.py**). In order to run the script, the following requirements must be met:

- Python v2.7 or later
- netcdf Python package
- numpy Python package
- matplotlib Python package (optional)
- basemap Python package (optional)

In order to generate the files necessary for the compliance test, test cases should be created with the `qc` option (i.e., `--set qc`) when running cice.setup. This option results in daily, non-averaged history files being written for a 5 year simulation.

To install the necessary Python packages, the `pip` Python utility can be used.

```
pip install --user netCDF4
pip install --user numpy
pip install --user matplotlib
```

To run the compliance test, setup a baseline run with the original baseline model and then a perturbation run based on recent model changes. Use `--sets qc` in both runs in addition to other settings needed. Then use the QC script to compare history output,

```
cp configuration/scripts/tests/QC/cice.t-test.py .
./cice.t-test.py /path/to/baseline/history /path/to/test/history
```

The script will produce output similar to:

```
INFO:__main__:Number of files: 1825
INFO:__main__:Two-Stage Test Passed
INFO:__main__:Quadratic Skill Test Passed for Northern Hemisphere
INFO:__main__:Quadratic Skill Test Passed for Southern Hemisphere
INFO:__main__:
INFO:__main__:Quality Control Test PASSED
```

Additionally, the exit code from the test (`echo $?`) will be 0 if the test passed, and 1 if the test failed.

Implementation notes: 1) Provide a pass/fail on each of the confidence intervals, 2) Facilitate output of a bitmap for each test so that locations of failures can be identified.

The cice.t-test.py requires memory to store multiple two-dimensional fields spanning 1825 unique timesteps, a total of several GB. An appropriate resource is needed to run the script. If the script runs out of memory on an interactive resource, try logging into a batch resource or finding a large memory node.

### End-To-End Testing Procedure

Below is an example of a step-by-step procedure for testing a code change that might result in non bit-for-bit results. First, run a regression test,

```
# Run a full regression test to verify bit-for-bit

# Create a baseline dataset (only necessary if no baseline exists on the system)
# git clone the baseline code

./cice.setup -m onyx -e intel --suite base_suite --testid base0 -bgen cice.my.baseline

# Run the test suite with the new code
# git clone the new code

./cice.setup -m onyx -e intel --suite base_suite --testid test0 --bcmp cice.my.
↪baseline

# Check the results

cd testsuite.test0
./results.csh
```

If the regression comparisons fail, then you may want to run the QC test,

```
# Run the QC test

# Create a QC baseline
# From the baseline sandbox

./cice.setup -m onyx -e intel --test smoke -g gx1 -p 44x1 --testid qc_base -s qc,
↪medium
cd onyx_intel_smoke_gx1_44x1_medium_qc.qc_base
./cice.build
./cice.submit

# Create the t-test testing data
# From the update sandbox

./cice.setup -m onyx -e intel --test smoke -g gx1 -p 44x1 -testid qc_test -s qc,medium
cd onyx_intel_smoke_gx1_44x1_medium_qc.qc_test
./cice.build
./cice.submit

# Wait for runs to finish
# Perform the QC test

cp configuration/scripts/tests/QC/cice.t-test.py
./cice.t-test.py /p/work/turner/CICE_RUNS/onyx_intel_smoke_gx1_44x1_medium_qc.qc_base␣
↪\
```

(continues on next page)

(continued from previous page)

```
                /p/work/turner/CICE_RUNS/onyx_intel_smoke_gx1_44x1_medium_qc.qc_test

# Example output:
INFO:__main__:Number of files: 1825
INFO:__main__:Two-Stage Test Passed
INFO:__main__:Quadratic Skill Test Passed for Northern Hemisphere
INFO:__main__:Quadratic Skill Test Passed for Southern Hemisphere
INFO:__main__:
INFO:__main__:Quality Control Test PASSED
```

### 3.3.5 Test Plotting

The CICE scripts include a script (`timeseries.csh`) that will generate timeseries figures from a diagnostic output file. When running a test suite, the `timeseries.csh` script is automatically copied to the suite directory. If the `timeseries.csh` script is to be used on a test or case that is not a part of a test suite, users will need to run the `timeseries.csh` script from the tests directory (`./configuration/scripts/tests/timeseries.csh ./path/`), or copy it to a local directory. When used with the test suites or given a path, it needs to be run in the directory above the particular case being plotted, but it can also be run on isolated log files in the same directory, without a path.

For example:

Run the test suite.

```
$ ./cice.setup -m conrad -e intel --suite base_suite --testid t00
```

Wait for suite to finish then go to the directory.

```
$ cd testsuite.t00
```

Run the timeseries script on the desired case.

```
$ ./timeseries.csh /p/work1/turner/CICE_RUNS/conrad_intel_smoke_col_1x1_diag1_
↪run1year.t00/
```

The output figures are placed in the directory where the `timeseries.csh` script is run.

To generate plots for all of the cases within a suite with a testid, create and run a script such as

```
#!/bin/csh
foreach dir (`ls -1  | grep testid`)
  echo $dir
  timeseries.csh $dir
end
```

This plotting script can be used to plot the following variables:

- total ice area ($km^2$)
- total ice extent ($km^2$)
- total ice volume ($m^3$)
- total snow volume ($m^3$)
- RMS ice speed ($m/s$)

## 3.4 Case Settings

There are two important files that define the case, **cice.settings** and **ice_in**. **cice.settings** is a list of env variables that define many values used to setup, build and run the case. **ice_in** is the input namelist file for CICE. Variables in both files are described below.

### 3.4.1 Table of CICE Settings

The **cice.settings** file is reasonably well self documented. Several of the variables defined in the file are not used in CICE. They exist to support the CICE model.

Table 5: *CICE settings*

| variable | options/format | description | recommended value |
|---|---|---|---|
| ICE_CASENAME | string | case name | set by cice.setup |
| ICE_SANDBOX | string | sandbox directory | set by cice.setup |
| ICE_MACHINE | string | machine name | set by cice.setup |
| ICE_COMPILER | string | environment name | set by cice.setup |
| ICE_MACHCOMP | string | machine_environment name | set by cice.setup |
| ICE_SCRIPTS | string | scripts directory | set by cice.setup |
| ICE_CASEDIR | string | case directory | set by cice.setup |
| ICE_RUNDIR | string | run directory | set by cice.setup |
| ICE_OBJDIR | string | compile directory | ${ICE_RUNDIR}/compile |
| ICE_RSTDIR | string | unused | ${ICE_RUNDIR}/restart |
| ICE_HSTDIR | string | unused | ${ICE_RUNDIR}/history |
| ICE_LOGDIR | string | log directory | ${ICE_CASEDIR}/logs |
| ICE_DRVOPT | string | unused | cice |
| ICE_IOTYPE | string | I/O format | set by cice.setup |
| | netcdf | serial netCDF | |
| | pio | parallel netCDF | |
| | none | netCDF library is not available | |
| ICE_CLEANBUILD | true, false | automatically clean before building | true |
| ICE_QUIETMODE | true, false | reduce build output to the screen | false |
| ICE_GRID | string (see below) | grid | set by cice.setup |
| | gx3 | 3-deg displace-pole (Greenland) global grid | |
| | gx1 | 1-deg displace-pole (Greenland) global grid | |
| | tx1 | 1-deg tripole global grid | |
| | gbox80 | 80x80 box | |
| | gbox128 | 128x128 box | |
| ICE_NTASKS | integer | number of tasks, must be set to 1 | set by cice.setup |
| ICE_NTHRDS | integer | number of threads per task, must be set to 1 | set by cice.setup |
| ICE_TEST | string | test setting if using a test | set by cice.setup |
| ICE_TESTNAME | string | test name if using a test | set by cice.setup |
| ICE_BASELINE | string | baseline directory name, associated with cice.setup -bdir | set by cice.setup |
| ICE_BASEGEN | string | baseline directory name for regression generation, associated with cice.setup -bgen | set by cice.setup |

Continued on next page

Table 5 – continued from previous page

| variable | options/format | description | recommended value |
|---|---|---|---|
| ICE_BASECOM | string | baseline directory name for regression comparison, associated with cice.setup -bcmp | set by cice.setup |
| ICE_BFBCOMP | string | location of case for comparison, associated with cice.setup -td | set by cice.setup |
| ICE_SPVAL | string | special value for cice.settings strings | set by cice.setup |
| ICE_RUNLENGTH | integer (see below) | batch run length default | set by cice.setup |
| | 0 | 15 minutes (default) | |
| | 1 | 59 minutes | |
| | 2 | 2 hours | |
| | other $2 < N < 8$ | N hours | |
| | 8 or larger | 8 hours | |
| ICE_ACCOUNT | string | batch account number | set by cice.setup, .cice_proj or by default |
| ICE_QUEUE | string | batch queue name | set by cice.setup or by default |
| ICE_THREADED | true, false | force threading in compile, will always compile threaded if ICE_NTHRDS $> 1$ | false |
| ICE_BLDDEBUG | true, false | turn on compile debug flags | false |

### 3.4.2 Table of namelist options

Table 6: *Namelist options*

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | | | | |
| | *setup_nml* | | | |
| | | | **Time, Diagnostics** | |
| * | `days_per_year` | `360` or `365` | number of days in a model year | 365 |
| * | `use_leap_years` | true/false | if true, include leap days | |
| * | `year_init` | yyyy | the initial year, if not using restart | |
| * | `istep0` | integer | initial time step number | 0 |
| * | `dt` | seconds | thermodynamics time step length | 3600. |
| * | `npt` | integer | total number of time steps to take | |
| * | `ndtd` | integer | number of dynamics/advection/ridging/steps per thermo timestep | 1 |
| | | | **Initialization/Restarting** | |
| | `runtype` | `initial` | start from `ice_ic` | |
| | | `continue` | restart using `pointer_file` | |
| * | `ice_ic` | `default` | latitude and sst dependent | default |
| | | `none` | no ice | |
| | | path/file | restart file name | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | restart | true/false | initialize using restart file | .true. |
| | use_restart_time | true/false | set initial date using restart file | .true. |
| | restart_format | nc | read/write restart files (use with PIO) | |
| | | bin | read/write binary restart files | |
| | lcdf64 | true/false | if true, use 64-bit format | |
| | numin | integer | minimum internal IO unit number | 11 |
| | numax | integer | maximum internal IO unit number | 99 |
| * | restart_dir | path/ | path to restart directory | |
| | restart_ext | true/false | read/write halo cells in restart files | |
| | restart_file | filename prefix | output file for restart dump | 'iced' |
| | pointer_file | pointer filename | contains restart filename | |
| * | dumpfreq | y | write restart every dumpfreq_n years | y |
| | | m | write restart every dumpfreq_n months | |
| | | d | write restart every dumpfreq_n days | |
| | | h | write restart every dumpfreq_n hours | |
| | | 1 | write restart every dumpfreq_n time step | |
| | dumpfreq_n | integer | frequency restart data is written | 1 |
| * | dump_last | true/false | if true, write restart on last time step of simulation | |
| | | | **Model Output** | |
| | bfbflag | off/lsum4/lsum8/lsumg16/ddpdd/reprosum | 16/8-byte sum/reprosum | off |
| * | diagfreq | integer | frequency of diagnostic output in dt | 24 |
| | | *e.g.*, 10 | once every 10 time steps | |
| * | diag_type | stdout | write diagnostic output to stdout | |
| | | file | write diagnostic output to file | |
| | diag_file | filename | diagnostic output file (script may reset) | |
| | print_global | true/false | print diagnostic data, global sums | .false. |
| | print_points | true/false | print diagnostic data for two grid points | .false. |
| | latpnt | real | latitude of (2) diagnostic points | |
| | lonpnt | real | longitude of (2) diagnostic points | |
| | dbug | true/false | if true, write extra diagnostics | .false. |
| | histfreq | string array | defines output frequencies | |
| | | y | write history every histfreq_n years | |
| | | m | write history every histfreq_n months | |
| | | d | write history every histfreq_n days | |
| | | h | write history every histfreq_n hours | |
| | | 1 | write history every histfreq_n time step | |
| | | x | unused frequency stream (not written) | |
| | histfreq_n | integer array | frequency history output is written | |
| | | 0 | do not write to history | |
| | hist_avg | true | write time-averaged data | .true. |
| | | false | write snapshots of data | |
| | history_dir | path/ | path to history output directory | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | history_file | filename prefix | output file for history | 'iceh' |
| | write_ic | true/false | write initial condition | |
| | incond_dir | path/ | path to initial condition directory | |
| | incond_file | filename prefix | output file for initial condition | 'iceh' |
| | runid | string | label for run (currently CESM only) | |
| | | | | |
| | *grid_nml* | | | |
| | | | **Grid** | |
| | grid_format | nc | read grid and kmt files | 'bin' |
| | | bin | read direct access, binary file | |
| | grid_type | rectangular | defined in *rectgrid* | |
| | | displaced_pole | read from file in *popgrid* | |
| | | tripole | read from file in *popgrid* | |
| | | regional | read from file in *popgrid* | |
| | grid_file | filename | name of grid file to be read | 'grid' |
| | bathymetry_file | filename | name of bathymetry file to be read | 'grid' |
| | use_bathymetry | true/false | use read in bathymetry file for basalstress option | |
| | kmt_file | filename | name of land mask file to be read | 'kmt' |
| | gridcpl_file | filename | input file for coupling grid info | |
| * | kcatbound | 0 | original category boundary formula | 0 |
| | | 1 | new formula with round numbers | |
| | | 2 | WMO standard categories | |
| | | -1 | one category | |
| | dxrect | real | x-direction grid spacing (cm) for rectangular grid | |
| | dyrect | real | y-direction grid spacing (cm) for rectangular grid | |
| | ncat | integer | number of ice thickness categories | 5 |
| | nilyr | integer | number of vertical layers in ice | 7 |
| | nslyr | integer | number of vertical layers in snow | 1 |
| | nblyr | integer | number of zbgc layers | 7 |
| | | | | |
| | *domain_nml* | | | |
| | | | **Domain** | |
| | nprocs | integer | number of processors to use | |
| | nx_global | integer | global grid size in x direction | |
| | ny_global | integer | global grid size in y direction | |
| | block_size_x | integer | block size in x direction | |
| | block_size_y | integer | block size in y direction | |
| | max_blocks | integer | maximum number of blocks per MPI task for memory allocation | |
| | processor_shape | slenderX1 | 1 processor in the y direction (tall, thin) | |
| | | slenderX2 | 2 processors in the y direction (thin) | |
| | | square-ice | more processors in x than y, $\sim$ square | |
| | | square-pop | more processors in y than x, $\sim$ square | |
| | distribution_type | cartesian | distribute blocks in 2D Cartesian array | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | | roundrobin | 1 block per proc until blocks are used | |
| | | sectcart | blocks distributed to domain quadrants | |
| | | sectrobin | several blocks per proc until used | |
| | | rake | redistribute blocks among neighbors | |
| | | spacecurve | distribute blocks via space-filling curves | |
| | | spiralcenter | distribute blocks via roundrobin from center of grid outward in a spiral | |
| | | wghtfile | distribute blocks based on weights specified in distribution_wght_file | |
| | distribution_wght | block | full block size sets work_per_block | |
| | | latitude | latitude/ocean sets work_per_block | |
| | distribution_wght_file | filename | distribution weight file when distribution_type is wghtfile | |
| | ew_boundary_type | cyclic | periodic boundary conditions in x-direction | |
| | | open | Dirichlet boundary conditions in x | |
| | ns_boundary_type | cyclic | periodic boundary conditions in y-direction | |
| | | open | Dirichlet boundary conditions in y | |
| | | tripole | U-fold tripole boundary conditions in y | |
| | | tripoleT | T-fold tripole boundary conditions in y | |
| | maskhalo_dyn | true/false | mask unused halo cells for dynamics | |
| | maskhalo_remap | true/false | mask unused halo cells for transport | |
| | maskhalo_bound | true/false | mask unused halo cells for boundary updates | |
| | | | | |
| | *tracer_nml* | | | |
| | | | **Tracers** | |
| | n_aero | integer | number of aerosol tracers | 1 |
| | n_zaero | 0,1,2,3,4,5,6 | number of z aerosol tracers in use | 0 |
| | n_algae | 0,1,2,3 | number of algal tracers | 0 |
| | n_doc | 0,1,2,3 | number of dissolved organic carbon | 0 |
| | n_dic | 0,1 | number of dissolved inorganic carbon | 0 |
| | n_don | 0,1 | number of dissolved organize nitrogen | 0 |
| | n_fed | 0,1,2 | number of dissolved iron tracers | 0 |
| | n_fep | 0,1,2 | number of particulate iron tracers | 0 |
| * | tr_iage | true/false | ice age | |
| | restart_age | true/false | restart tracer values from file | |
| * | tr_FY | true/false | first-year ice area | |
| | restart_FY | true/false | restart tracer values from file | |
| * | tr_lvl | true/false | level ice area and volume | |
| | restart_lvl | true/false | restart tracer values from file | |
| * | tr_pond_cesm | true/false | CESM melt ponds | |
| | restart_pond_cesm | true/false | restart tracer values from file | |
| * | tr_pond_topo | true/false | topo melt ponds | |
| | restart_pond_topo | true/false | restart tracer values from file | |
| * | tr_pond_lvl | true/false | level-ice melt ponds | |
| | restart_pond_lvl | true/false | restart tracer values from file | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| * | tr_aero | true/false | aerosols | |
| | restart_aero | true/false | restart tracer values from file | |
| | | | | |
| | *thermo_nml* | | | |
| | | | **Thermodynamics** | |
| * | kitd | 0 | delta function ITD approximation | 1 |
| | | 1 | linear remapping ITD approximation | |
| * | ktherm | 0 | zero-layer thermodynamic model | |
| | | 1 | Bitz and Lipscomb thermodynamic model | |
| | | 2 | mushy-layer thermodynamic model | |
| | | -1 | thermodynamics disabled | |
| * | conduct | Maykut71 | conductivity [30] | |
| | | bubbly | conductivity [35] | |
| * | a_rapid_mode | real | brine channel diameter | 0.5x10 $^{-3}$ m |
| * | Rac_rapid_mode | real | critical Rayleigh number | 10 |
| * | aspect_rapid_mode | real | brine convection aspect ratio | 1 |
| * | dSdt_slow_mode | real | drainage strength parameter | -1.5x10 $^{-7}$ m/s/K |
| * | phi_c_slow_mode | $0 < \phi_c < 1$ | critical liquid fraction | 0.05 |
| * | phi_i_mushy | $0 < \phi_i < 1$ | solid fraction at lower boundary | 0.85 |
| | | | | |
| | *dynamics_nml* | | | |
| | | | **Dynamics** | |
| | kdyn | -1 | dynamics OFF | 1 |
| | | 0 | dynamics OFF | |
| | | 1 | EVP dynamics | |
| | | 2 | EAP dynamics | |
| | | 1 | dynamics ON | |
| | revised_evp | true/false | use revised EVP formulation | |
| | ndte | integer | number of EVP subcycles | 240 |
| | advection | remap | linear remapping advection | 'remap' |
| | | upwind | donor cell advection | |
| * | kstrength | 0 | ice strength formulation [11] | 1 |
| | | 1 | ice strength formulation [39] | |
| * | krdg_partic | 0 | old ridging participation function | 1 |
| | | 1 | new ridging participation function | |
| * | krdg_redist | 0 | old ridging redistribution function | 1 |
| | | 1 | new ridging redistribution function | |
| * | mu_rdg | real | e-folding scale of ridged ice | |
| * | Cf | real | ratio of ridging work to PE change in ridging | 17. |
| | coriolis | latitude | Coriolis variable by latitude | 'latitude' |
| | | constant | Constant coriolis value = 1.46e-4 | |
| | | zero | Zero coriolis | |
| | kridge | 1 | Ridging Enabled | 1 |
| | | -1 | Ridging Disabled | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | ktransport | 1 | Transport Enabled | 1 |
| | | -1 | Transport Disabled | |
| | basalstress | true/false | use basal stress parameterization for landfast ice | |
| | k1 | real | 1st free parameter for landfast parameterization | 8. |
| | e_ratio | real | EVP ellipse aspect ratio | 2.0 |
| | Ktens | real | Tensile strength factor (see [2]) | 0.0 |
| | | | | |
| | *shortwave_nml* | | | |
| | | | **Shortwave** | |
| * | shortwave | ccsm3 | NCAR CCSM3 distribution method | |
| | | dEdd | Delta-Eddington method | |
| * | albedo_type | ccsm3 | NCAR CCSM3 albedos | 'default' |
| | | constant | four constant albedos | |
| * | albicev | $0 < \alpha < 1$ | visible ice albedo for thicker ice | |
| * | albicei | $0 < \alpha < 1$ | near infrared ice albedo for thicker ice | |
| * | albsnowv | $0 < \alpha < 1$ | visible, cold snow albedo | |
| * | albsnowi | $0 < \alpha < 1$ | near infrared, cold snow albedo | |
| * | ahmax | real | albedo is constant above this thickness | 0.3 m |
| * | R_ice | real | tuning parameter for sea ice albedo from Delta-Eddington shortwave | |
| * | R_pnd | real | ... for ponded sea ice albedo ... | |
| * | R_snw | real | ... for snow (broadband albedo) ... | |
| * | dT_mlt | real | $\Delta$ temperature per $\Delta$ snow grain radius | |
| * | rsnw_mlt | real | maximum melting snow grain radius | |
| * | kalg | real | absorption coefficient for algae | |
| | | | | |
| | *ponds_nml* | | | |
| | | | **Melt Ponds** | |
| * | hp1 | real | critical ice lid thickness for topo ponds | 0.01 m |
| * | hs0 | real | snow depth of transition to bare sea ice | 0.03 m |
| * | hs1 | real | snow depth of transition to pond ice | 0.03 m |
| * | dpscale | real | time scale for flushing in permeable ice | $1 \times 10^{-3}$ |
| * | frzpnd | hlid | Stefan refreezing with pond ice thickness | 'hlid' |
| | | cesm | CESM refreezing empirical formula | |
| * | rfracmin | $0 \leq r_{min} \leq 1$ | minimum melt water added to ponds | 0.15 |
| * | rfracmax | $0 \leq r_{max} \leq 1$ | maximum melt water added to ponds | 1.0 |
| * | pndaspect | real | aspect ratio of pond changes (depth:area) | 0.8 |
| | | | | |
| | *forcing_nml* | | | |
| | | | **Forcing** | |
| * | formdrag | true/false | calculate form drag | |
| * | atmbndy | default | stability-based boundary layer | 'default' |
| | | constant | bulk transfer coefficients | |
| * | fyear_init | yyyy | first year of atmospheric forcing data | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| * | ycycle | integer | number of years in forcing data cycle | |
| * | calc_strair | true | calculate wind stress and speed | |
| | | false | read wind stress and speed from files | |
| * | highfreq | true/false | high-frequency atmo coupling | |
| * | natmiter | integer | number of atmo boundary layer iterations | |
| * | calc_Tsfc | true/false | calculate surface temperature | .true. |
| * | default_season | winter | Sets initial values of forcing and is overwritten if forcing is read in. | |
| * | precip_units | mks | liquid precipitation data units | |
| | | mm_per_month | | |
| | | mm_per_sec | (same as MKS units) | |
| | | m_per_sec | | |
| * | tfrz_option | minus1p8 | constant ocean freezing temperature ($-1.8°C$) | |
| | | linear_salt | linear function of salinity (ktherm=1) | |
| | | mushy_layer | matches mushy-layer thermo (ktherm=2) | |
| * | ustar_min | real | minimum value of ocean friction velocity | 0.0005 m/s |
| * | emissivity | real | emissivity of snow and ice | 0.95 |
| * | fbot_xfer_type | constant | constant ocean heat transfer coefficient | |
| | | Cdn_ocn | variable ocean heat transfer coefficient | |
| * | update_ocn_f | true | include frazil water/salt fluxes in ocn fluxes | |
| | | false | do not include (when coupling with POP) | |
| * | l_mpond_fresh | true | retain (topo) pond water until ponds drain | |
| | | false | release (topo) pond water immediately to ocean | |
| * | oceanmixed_ice | true/false | active ocean mixed layer calculation | .true. (if uncoupled) |
| * | restore_ocn | true/false | restore sst to data | |
| * | trestore | integer | sst restoring time scale (days) | |
| | restore_ice | true/false | restore ice state along lateral boundaries | |
| * | atm_data_type | default | constant values defined in the code | |
| | | LYq | AOMIP/Large-Yeager forcing data | |
| | | monthly | monthly forcing data | |
| | | ncar | NCAR bulk forcing data | |
| | | box2001 | forcing data for [15] box problem | |
| | | oned | column forcing data | |
| | | hycom | HYCOM atm forcing data in netcdf format | |
| * | ocn_data_type | default | constant values defined in the code | |
| | | clim | climatological data | |
| | | ncar | POP ocean forcing data | |
| | | hycom | HYCOM ocean forcing data in netcdf format | Constant initial forcing |
| | bgc_data_type | default | constant values defined in the code | |
| | | clim | climatological data | |
| | | ncar | POP ocean forcing data | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | | hycom | HYCOM ocean forcing data in netcdf format | Constant initial forcing |
| | fe_data_type | default | default forcing value for iron | |
| | | clim | iron forcing from ocean climatology | |
| | ice_data_type | string | ice initialization for special tests | default |
| | | default | no special initialization | |
| | | box2001 | initialize ice concentration for *box2001* test ([15]) | |
| | | boxslotcyl | initialize ice concentration and velocity for *boxslotcyl* test ([51]) | |
| | atm_data_format | nc | read atmo forcing files | |
| | | bin | read direct access, binary files | |
| | ocn_data_format | nc | read ocean forcing files | |
| | | bin | read direct access, binary files | |
| * | oceanmixed_file | filename | data file containing ocean forcing data | |
| | atm_data_dir | path/ | path to atmospheric forcing data directory | |
| | ocn_data_dir | path/ | path to oceanic forcing data directory | |
| | bgc_data_dir | path/ | path to oceanic forcing data directory | |
| | | | | |
| | *zbgc_nml* | | | |
| | | | **Biogeochemistry** | More information about the BGC tuning can be found in the Icepack Documentation. |
| * | tr_brine | true/false | brine height tracer | |
| * | tr_zaero | true/false | vertical aerosol tracers | |
| * | modal_aero | true/false | modal aersols | |
| | restore_bgc | true/false | restore bgc to data | |
| | solve_zsal | true/false | update salinity tracer profile | |
| * | skl_bgc | true/false | biogeochemistry | |
| | bgc_flux_type | Jin2006 | ice–ocean flux velocity of [21] | |
| | | constant | constant ice–ocean flux velocity | |
| | restart_bgc | true/false | restart tracer values from file | |
| | tr_bgc_C_sk | true/false | algal carbon tracer | |
| | tr_bgc_chl_sk | true/false | algal chlorophyll tracer | |
| | tr_bgc_Am_sk | true/false | ammonium tracer | |
| | tr_bgc_Sil_sk | true/false | silicate tracer | |
| | tr_bgc_DMSPp_sk | true/false | particulate DMSP tracer | |
| | tr_bgc_DMSPd_sk | true/false | dissolved DMSP tracer | |
| | tr_bgc_DMS_sk | true/false | DMS tracer | |
| | phi_snow | real | snow porosity for brine height tracer | |
| | | | | |
| | *icefields_nml* | | | |
| | | | *History Fields* | |

Continued on next page

Table 6 – continued from previous page

| also in Icepack | variable | options/format | description | recommended value |
|---|---|---|---|---|
| | f_<var> | string | frequency units for writing <var> to history | |
| | | y | write history every histfreq_n years | |
| | | m | write history every histfreq_n months | |
| | | d | write history every histfreq_n days | |
| | | h | write history every histfreq_n hours | |
| | | 1 | write history every time step | |
| | | x | do not write <var> to history | |
| | | md | *e.g.,* write both monthly and daily files | |
| | f_<var>_ai | | grid cell average of <var> ($\times a_i$) | |

## 3.5 Troubleshooting

Check the FAQ: https://github.com/CICE-Consortium/CICE/wiki

### 3.5.1 Initial setup

If there are problems, you can manually edit the env, Macros, and **cice.run** files in the case directory until things are working properly. Then you can copy the env and Macros files back to **configuration/scripts/machines**.

Changes made directly in the run directory, e.g. to the namelist file, will be overwritten if scripts in the case directory are run again later.

If changes are needed in the **cice.run.setup.csh** script, it must be manually modified.

Ensure that the block size block_size_x, block_size_y, and max_blocks is compatible with the processor_shape and other domain options in **ice_in**

If using the rake or space-filling curve algorithms for block distribution (*distribution_type* in **ice_in**) the code will abort if max_blocks is not large enough. The correct value is provided in the diagnostic output. Also, the spacecurve setting can only be used with certain block sizes that results in number of blocks in the x and y directions being only multiples of 2, 3, or 5.

If starting from a restart file, ensure that kcatbound is the same as that used to create the file (*kcatbound* = 0 for the files included in this code distribution). Other configuration parameters, such as *NICELYR*, must also be consistent between runs.

For stand-alone runs, check that *-Dcoupled* is *not* set in the **Macros.*** file.

For coupled runs, check that *-Dcoupled* and other coupled-model-specific (e.g., CESM, popcice or hadgem) preprocessing options are set in the **Macros.*** file.

Set ICE_CLEANBUILD to true to clean before rebuilding.

### 3.5.2 Restarts

Manual restart tests require the path to the restart file be included in ice_in in the namelist file.

Ensure that kcatbound is the same as that used to create the restart file. Other configuration parameters, such as nilyr, must also be consistent between runs.

CICE v5 and later use a model configuration that makes restarting from older simulations difficult. In particular, the number of ice categories, the category boundaries, and the number of vertical layers within each category must be the same in the restart file and in the run restarting from that file. Moreover, significant differences in the physics, such as the salinity profile, may cause the code to fail upon restart. Therefore, new model configurations may need to be started using *runtype* = 'initial'. Binary restart files that were provided with CICE v4.1 were made using the BL99 thermodynamics with 4 layers and 5 thickness categories (*kcatbound* = 0) and therefore can not be used for the default CICE v5 and later configuration (7 layers). In addition, CICE's default restart file format is now  instead of binary.

Restarting a run using *runtype* = 'continue' requires restart data for all tracers used in the new run. If tracer restart data is not available, use *runtype* = 'initial', setting *ice_ic* to the name of the core restart file and setting to true the namelist restart flags for each tracer that is available. The unavailable tracers will be initialized to their default settings.

On tripole grids, use *restart_ext* = true when using either binary or regular (non-PIO) netcdf.

Provided that the same number of ice layers (default: 4) will be used for the new runs, it is possible to convert v4.1 restart files to the new file structure and then to  format. If the same physical parameterizations are used, the code should be able to execute from these files. However if different physics is used (for instance, mushy thermo instead of BL99), the code may still fail. To convert a v4.1 restart file, consult section 5.2 in the CICE v5 documentation.

If restart files are taking a long time to be written serially (i.e., not using PIO), see the next section.

### 3.5.3 Slow execution

On some architectures, underflows ($10^{-300}$ for example) are not flushed to zero automatically. Usually a compiler flag is available to do this, but if not, try uncommenting the block of code at the end of subroutine *stress* in **ice_dyn_evp.F90** or **ice_dyn_eap.F90**. You will take a hit for the extra computations, but it will not be as bad as running with the underflows.

### 3.5.4 Debugging hints

Several utilities are available that can be helpful when debugging the code. Not all of these will work everywhere in the code, due to possible conflicts in module dependencies.

***debug_ice* (CICE.F90)** A wrapper for *print_state* that is easily called from numerous points during the timestepping loop (see **CICE_RunMod.F90_debug**, which can be substituted for **CICE_RunMod.F90**).

***print_state* (ice_diagnostics.F90)** Print the ice state and forcing fields for a given grid cell.

***dbug* = true (ice_in)** Print numerous diagnostic quantities.

***print_global* (ice_in)** If true, compute and print numerous global sums for energy and mass balance analysis. This option can significantly degrade code efficiency.

***print_points* (ice_in)** If true, print numerous diagnostic quantities for two grid cells, one near the north pole and one in the Weddell Sea. This utility also provides the local grid indices and block and processor numbers (*ip*, *jp*, *iblkp*, *mtask*) for these points, which can be used in conjunction with *check_step*, to call *print_state*. These flags are set in **ice_diagnostics.F90**. This option can be fairly slow, due to gathering data from processors.

***global_minval, global_maxval, global_sum* (ice_global_reductions.F90)** Compute and print the minimum and maximum values for an individual real array, or its global sum.

### 3.5.5 Known bugs

- Fluxes sent to the CESM coupler may have incorrect values in grid cells that change from an ice-free state to having ice during the given time step, or vice versa, due to scaling by the ice area. The authors of the CESM

flux coupler insist on the area scaling so that the ice and land models are treated consistently in the coupler (but note that the land area does not suddenly become zero in a grid cell, as does the ice area).

- With the old CCSM radiative scheme (*shortwave* = 'default' or 'ccsm3'), a sizable fraction (more than 10%) of the total shortwave radiation is absorbed at the surface but should be penetrating into the ice interior instead. This is due to use of the aggregated, effective albedo rather than the bare ice albedo when *snowpatch* $< 1$.
- The date-of-onset diagnostic variables, *melt_onset* and *frz_onset*, are not included in the core restart file, and therefore may be incorrect for the current year if the run is restarted after Jan 1. Also, these variables were implemented with the Arctic in mind and may be incorrect for the Antarctic.
- The single-processor *system_clock* time may give erratic results on some architectures.
- History files that contain time averaged data (*hist_avg* = true in **ice_in**) will be incorrect if restarting from midway through an averaging period.
- In stand-alone runs, restarts from the end of *ycycle* will not be exact.
- Using the same frequency twice in *histfreq* will have unexpected consequences and causes the code to abort.
- Latitude and longitude fields in the history output may be wrong when using padding.
- History and restart files will not be written on the first timestep in some cases.

### 3.5.6 Interpretation of albedos

More information about interpretation of albedos can be found in the Icepack documentation.

### 3.5.7 Proliferating subprocess parameterizations

With the addition of several alternative parameterizations for sea ice processes, a number of subprocesses now appear in multiple parts of the code with differing descriptions. For instance, sea ice porosity and permeability, along with associated flushing and flooding, are calculated separately for mushy thermodynamics, topo and level-ice melt ponds, and for the brine height tracer, each employing its own equations. Likewise, the BL99 and mushy thermodynamics compute freeboard and snow–ice formation differently, and the topo and level-ice melt pond schemes both allow fresh ice to grow atop melt ponds, using slightly different formulations for Stefan freezing. These various process parameterizations will be compared and their subprocess descriptions possibly unified in the future.

CHAPTER 4

# Developer Guide

## 4.1 About Development

The CICE model consists of four different parts, the CICE dynamics and supporting infrastructure, the CICE driver code, the Icepack column physics code, and the scripts. Development of each of these pieces is described separately.

**Guiding principles for the creation of CICE include the following:**

- CICE can be run in stand-alone or coupled modes. A top layer driver, coupling layer, or model cap can be used to drive the CICE model.
- The Icepack column physics modules are independent, consist of methods that operate on individual grid-cells, and contain no underlying infrastructure. CICE must call into Icepack using interfaces and approaches specified by Icepack.

### 4.1.1 Git workflow and Pull Requests

There is extensive Information for Developers documentation available. See https://github.com/CICE-Consortium/About-Us/wiki/Resource-Index#information-for-developers for information on:

- Contributing to model development
- Software development practices guide
- git Workflow Guide - including extensive information about the Pull Request process and requirements
- Documentation Workflow Guide

## 4.2 Dynamics and Infrastructure

The CICE **cicecore/** directory consists of the non icepack source code. Within that directory there are the following subdirectories

**cicecore/cicedynB/analysis** contains higher level history and diagnostic routines.

**cicecore/cicedynB/dynamics** contains all the dynamical evp, eap, and transport routines.

**cicecore/cicedynB/general** contains routines associated with forcing, flux calculation, initialization, and model timestepping.

**cicecore/cicedynB/infrastructure** contains most of the low-level infrastructure associated with communication (halo updates, gather, scatter, global sums, etc) and I/O reading and writing binary and netcdf files.

**cicecore/drivers/** contains subdirectories that support stand-alone drivers and other high level coupling layers.

**cicecore/shared/** contains some basic methods related to grid decomposition, time managers, constants, kinds, and restart capabilities.

### 4.2.1 Dynamics

#### Dynamical Solvers

The dynamics solvers are found in **cicecore/cicedynB/dynamics/**. A couple of different solvers are available including EVP, revised EVP, and EAP. The dynamics solver is specified in namelist with the `kdyn` variable. `kdyn=1` is evp, `kdyn=2` is eap, and revised evp requires the `revised_evp` namelist flag be set to true.

Multiple evp solvers are supported thru the namelist flag `kevp_kernel`. The standard implementation and current default is `kevp_kernel=0`. In this case, the stress is solved on the regular decomposition via subcycling and calls to subroutine stress and subroutine stepu with MPI global sums required in each subcycling call. With `kevp_kernel=2`, the data required to compute the stress is gathered to the root MPI process and the stress calculation is performed on the root task without any MPI global sums. OpenMP parallelism is supported in `kevp_kernel=2`. The solutions with `kevp_kernel` set to 0 or 2 will not be bit-for-bit identical but should be the same to roundoff and produce the same climate. `kevp_kernel=2` may perform better for some configurations, some machines, and some pe counts. `kevp_kernel=2` is not supported with the tripole grid and is still being validated. Until `kevp_kernel=2` is fully validated, it will abort if set. To override the abort, use value 102 for testing.

#### Transport

The transport (advection) methods are found in **cicecore/cicedynB/dynamics/**. Two methods are supported, upwind and remap. These are set in namelist via the advection variable.

### 4.2.2 Infrastructure

#### Kinds

**cicecore/shared/ice_kinds_mod.F90** defines the kinds datatypes used in CICE. These kinds are used throughout CICE code to define variable types. The CICE kinds are adopted from the kinds defined in Icepack for consistency in interfaces.

#### Constants

**cicecore/shared/ice_constants.F90** defines several model constants. Some are hardwired parameters while others have internal defaults and can be set thru namelist.

### Dynamic Array Allocation

CICE v5 and earlier was implemented using mainly static arrays and required several CPPs to be set to define grid size, blocks sizes, tracer numbers, and so forth. With CICE v6 and later, arrays are dynamically allocated and those parameters are namelist settings. The following CPPs are no longer used in CICE v6 and later versions,

> -DNXGLOB=100 -DNYGLOB=116 -DBLCKX=25 -DBLCKY=29 -DMXBLCKS=4 -DNICELYR=7 -DNSNWLYR=1 -DNICECAT=5 -DTRAGE=1 -DTRFY=1 -DTRLVL=1 -DTRPND=1 -DTRBRI=0 -DNTRAERO=1 -DTRZS=0 -DNBGCLYR=7 -DTRALG=0 -DTRBGCZ=0 -DTRDOC=0 -DTRDOC=0 -DTRDIC=0 -DTRDON=0 -DTRFED=0 -DTRFEP=0 -DTRZAERO=0 -DTRBGCS=0 -DNUMIN=11 -DNUMAX=99

as they have been migrated to *Table of namelist options*

> nx_global, ny_global, block_size_x, block_size_y, max_blocks, nilyr, nslyr, ncat, nblyr, n_aero, n_zaero, n_algae, n_doc, n_dic, n_don, n_fed, n_fep, numin, numax

### Time Manager

Time manager data is module data in **cicecore/shared/ice_calendar.F90**. Much of the time manager data is public and operated on during the model timestepping. The model timestepping actually takes place in the **CICE_RunMod.F90** file which is part of the driver code and tends to look like this:

```
call ice_step
istep  = istep  + 1    ! update time step counters
istep1 = istep1 + 1
time = time + dt       ! determine the time and date
```

### Communication

Two low-level communications packages, mpi and serial, are provided as part of CICE. This software provides a middle layer between the model and the underlying libraries. Only the CICE mpi or serial directories are compiled with CICE, not both.

**cicedynB/infrastructure/comm/mpi/** is based on MPI and provides various methods to do halo updates, global sums, gather/scatter, broadcasts and similar using some fairly generic interfaces to isolate the MPI calls in the code.

**cicedynB/infrastructure/comm/serial/** support the same interfaces, but operates in shared memory mode with no MPI. The serial library will be used, by default in the CICE scripts, if the number of MPI tasks is set to 1. The serial library allows the model to be run on a single core or with OpenMP parallelism only without requiring an MPI library.

### I/O

There are three low-level IO packages in CICE, io_netcdf, io_binary, and io_pio. This software provides a middle layer between the model and the underlying IO writing. Only one of the three IO directories can be built with CICE. The CICE scripts will build with the io_netcdf by default, but other options can be selecting by setting `ICE_IOTYPE` in **cice.settings** in the case. This has to be set before CICE is built.

**cicedynB/infrastructure/io/io_netcdf/** is the default for the standalone CICE model, and it supports writing history and restart files in netcdf format using standard netcdf calls. It does this by writing from and reading to the root task and gathering and scattering fields from the root task to support model parallelism.

**cicedynB/infrastructure/io/io_binary/** supports files in binary format using a gather/scatter approach and reading to and writing from the root task.

**cicedynB/infrastructure/io/io_pio/** support reading and writing through the pio interface. pio is a parallel io library (https://github.com/NCAR/ParallelIO) that supports reading and writing of binary and netcdf file through various interfaces including netcdf and pnetcdf. pio is generally more parallel in memory even when using serial netcdf than the standard gather/scatter methods, and it provides parallel read/write capabilities by optionally linking and using pnetcdf.

## 4.3 Driver and Coupling

The driver and coupling layer is found in **cicecore/drivers/**. The standalone driver is found under **cicecore/drivers/cice/** and other high level coupling layers are found in other directories. In general, CICE will build with only one of these drivers, depending how the model is run and coupled. Within the **cicecore/drivers/cice/** directory, the following files are found,

**CICE.F90** is the top level program file and that calls CICE_Initialize, CICE_Run, and CICE_Finalize methods. **CICE_InitMod.F90** contains the CICE_Initialize method and other next level source code. **CICE_RunMod.F90** contains the CICE_Run method and other next level source code. **CICE_FinalMod.F90** contains the CICE_Finalize method and other next level source code.

Other **cicecore/drivers/** directories are similarly implemented with a top level coupling layer, that is largely specified by an external coupled system and then some version of the **CICE_InitMod.F90**, **CICE_RunMod.F90**, and **CICE_FinalMod.F90** files.

### 4.3.1 Calling Sequence

The initialize calling sequence looks something like:

```
call init_communicate     ! initial setup for message passing
call init_fileunits       ! unit numbers
call icepack_configure()  ! initialize icepack
call input_data           ! namelist variables
call init_zbgc            ! vertical biogeochemistry namelist
call init_domain_blocks   ! set up block decomposition
call init_grid1           ! domain distribution
call alloc_*              ! allocate arrays
call init_ice_timers      ! initialize all timers
call init_grid2           ! grid variables
call init_calendar        ! initialize some calendar stuff
call init_hist (dt)       ! initialize output history file
if (kdyn == 2) then
   call init_eap (dt_dyn) ! define eap dynamics parameters, variables
else                      ! for both kdyn = 0 or 1
   call init_evp (dt_dyn) ! define evp dynamics parameters, variables
endif
call init_coupler_flux    ! initialize fluxes exchanged with coupler
call init_thermo_vertical ! initialize vertical thermodynamics
call icepack_init_itd(ncat, hin_max)  ! ice thickness distribution
call calendar(time)       ! determine the initial date
call init_forcing_ocn(dt) ! initialize sss and sst from data
call init_state           ! initialize the ice state
call init_transport       ! initialize horizontal transport
call ice_HaloRestore_init ! restored boundary conditions
call init_restart         ! initialize restart variables
call init_diags           ! initialize diagnostic output points
call init_history_therm   ! initialize thermo history variables
```

(continues on next page)

(continued from previous page)

```
call init_history_dyn     ! initialize dynamic history variables
call init_shortwave    ! initialize radiative transfer
call init_forcing_atmo    ! initialize atmospheric forcing (standalone)
```

See a **CICE_InitMod.F90** file for the latest.

The run sequence within a time loop looks something like:

```
call init_mass_diags   ! diagnostics per timestep
call init_history_therm
call init_history_bgc

do iblk = 1, nblocks
   if (calc_Tsfc) call prep_radiation (dt, iblk)
   call step_therm1     (dt, iblk) ! vertical thermodynamics
   call biogeochemistry (dt, iblk) ! biogeochemistry
   call step_therm2     (dt, iblk) ! ice thickness distribution thermo
enddo ! iblk

call update_state (dt, daidtt, dvidtt, dagedtt, offset)

do k = 1, ndtd
   call step_dyn_horiz (dt_dyn)
   do iblk = 1, nblocks
      call step_dyn_ridge (dt_dyn, ndtd, iblk)
   enddo
   call update_state (dt_dyn, daidtd, dvidtd, dagedtd, offset)
enddo

do iblk = 1, nblocks
   call step_radiation (dt, iblk)
   call coupling_prep (iblk)
enddo ! iblk
```

See a **CICE_RunMod.F90** file for the latest.

## 4.4 Icepack

The CICE model calls the Icepack columnphysics source code. The Icepack model is documented separately, see https://github.com/CICE-Consortium/Icepack.

More specifically, the CICE model uses methods defined in **icepack_intfc.F90**. It uses the init, query, and write methods to set, get, and document Icepack values. And it follows the icepack_warnings methodology where icepack_warnings_aborted is checked and icepack_warnings_print is called after every call to an Icepack method. It does not directly "use" Icepack data and access Icepack data only thru interfaces.

## 4.5 Scripts

The scripts are the third part of the cice package. They support setting up cases, building, and running the cice stand-alone model.

### 4.5.1 File List

The directory structure under configure/scripts is as follows.

**configuration/scripts/**

**Makefile** primary makefile
**cice.batch.csh** creates batch scripts for particular machines
**cice.build** compiles the code
**cice.decomp.csh** computes a decomposition given a grid and task/thread count
**cice.launch.csh** creates script logic that runs the executable
**cice.run.setup.csh** sets up the run scripts
**cice.settings** defines environment, model configuration and run settings
**cice.test.setup.csh** creates configurations for testing the model
**ice_in** namelist input data
**machines/** machine specific files to set env and Macros
**makdep.c** determines module dependencies
**options/** other namelist configurations available from the cice.setup command line
**parse_namelist.sh** replaces namelist with command-line configuration
**parse_namelist_from_settings.sh** replaces namelist with values from cice.settings
**parse_settings.sh** replaces settings with command-line configuration
**setup_run_dirs.csh** creates the case run directories
**set_version_number.csh** updates the model version number from the **cice.setup** command line
**tests/** scripts for configuring and running basic tests

### 4.5.2 Strategy

The cice scripts are implemented such that everything is resolved after **cice.setup** is called. This is done by both copying specific files into the case directory and running scripts as part of the **cice.setup** command line to setup various files.

**cice.setup** drives the case setup. It is written in csh. All supporting scripts are relatively simple csh or sh scripts. See *Scripts* for additional details.

The file **cice.settings** specifies a set of env defaults for the case. The file **ice_in** defines the namelist input for the cice driver.

### 4.5.3 Preset Case Options

The `cice.setup --set` option allows the user to choose some predetermined cice settings and namelist. Those options are defined in **configurations/scripts/options/** and the files are prefixed by either set_env or set_nml. When **cice.setup** is executed, the appropriate files are read from **configurations/scripts/options/** and the **cice.settings** and/or **ice_in** files are updated in the case directory based on the values in those files.

The filename suffix determines the name of the -s option. So, for instance,

```
cice.setup -s diag1,debug,bgcISPOL
```

will search for option files with suffixes of diag1, debug, and bgcISPOL and then apply those settings.

**parse_namelist.sh**, **parse_settings.sh**, and **parse_namelist_from_settings.sh** are the three scripts that modify **ice_in** and **cice.settings**.

To add new options, just add new files to the **configurations/scripts/options/** directory with appropriate names and syntax. The set_nml file syntax is the same as namelist syntax and the set_env files are consistent with csh setenv syntax. See other files for examples of the syntax.

### 4.5.4 Build Scripts

CICE uses GNU Make to build the model. There is a common **Makefile** for all machines. Each machine provides a Macros file to define some Makefile variables and and an env file to specify the modules/software stack for each compiler. The machine is built by the cice.build script which invokes Make. There is a special trap for circular dependencies in the cice.build script to highlight this error when it occurs.

### 4.5.5 Machines

Machine specific information is contained in **configuration/scripts/machines**. That directory contains a Macros file and an env file for each supported machine. One other files will need to be changed to support a port, that is **configuration/scripts/cice.batch.csh**. To port to a new machine, see *Porting*.

### 4.5.6 Test Options

Values that are associated with the *–sets* cice.setup are defined in **configuration/scripts/options**. Those files are text files and cice.setup uses the values in those files to modify the *cice.settings* and *ice_in* files in the case as the case is created. Files name *set_env.$option* are associated with values in the *cice.settings* file. Files named *set_nml.$option* are associated with values in *ice.in*. These files contain simple keyword pair values one line at a time. A line starting with # is a comment. Files names that start with *test_* are used specifically for tests.

That directory also contains files named *set_files.$option*. This provides an extra layer on top of the individual setting files that allows settings to be defined based on groups of other settings. The *set_files.$option* files contain a list of *–sets* options to be applied.

The $option part of the filename is the argument to *–sets* argument in *cice.setup*. Multiple options can be specified by creating a comma delimited list. In the case where settings contradict each other, the last defined is used.

### 4.5.7 Test scripts

Under **configuration/scripts/tests** are several files including the scripts to setup the various tests, such as smoke and restart tests (**test_smoke.script**, **test_restart.script**) and the files that describe with options files are needed for each test (ie. **test_smoke.files**, **test_restart.files**). A baseline test script (**baseline.script**) is also there to setup the general regression and comparison testing. That directory also contains the preset test suites (ie. **base_suite.ts**) and a file that supports post-processing on the model output (**timeseries.csh**). There is also a script **report_results.csh** that pushes results from test suites back to the CICE-Consortium test results wiki page.

To add a new test (for example newtest), several files may be needed,

- **configuration/scripts/tests/test_newtest.script** defines how to run the test. This chunk of script will be incorporated into the case test script
- **configuration/scripts/tests/test_newtest.files** list the set of options files found in **configuration/scripts/options/** needed to run this test. Those files will be copied into the test directory when the test is invoked so they are available for the **test_newtest.script** to use.
- some new files may be needed in **configuration/scripts/options/**. These could be relatively generic **set_nml** or **set_env** files, or they could be test specific files typically carrying a prefix of **test_nml**.

Generating a new test, particularly the **test_newtest.script** usually takes some iteration before it's working properly.

### 4.5.8 Code Compliance Script

The code compliance test validates non bit-for-bit model changes. The directory **configuration/scripts/tests/QC** contains scripts related to the compliance testing, and this process is described in *Code Compliance Test (non bit-for-bit validation)*. This section will describe a set of scripts that test and validate the code compliance process. This should be done when the compliance test or compliance test scripts (i.e., `cice.t-test.py`) are modified. Again, this section **documents a validation process for the compliance scripts**; it does not describe to how run the compliance test itself.

Two scripts have been created to automatically validate the code compliance script. These scripts are:

- `gen_qc_cases.csh`, which creates the 4 test cases required for validation, builds the executable, and submits to the queue.
- `compare_qc_cases.csh`, which runs the code compliance script on three combinations of the 4 test cases and outputs whether or not the correct response was received.

The `gen_qc_cases.csh` script allows users to pass some arguments similar to the `cice.setup` script. These options include:

- `--mach, -m`: Machine (REQUIRED)
- `--env, -e`: Compiler
- `--pes, -p`: tasks x threads
- `--acct` : Account number for batch submission
- `--grid, -g`: Grid
- `--queue` : Queue for the batch submission
- `--testid` : test ID, user-defined id for testing

The script creates 4 test cases, with testIDs `qc_base`, `qc_bfb`, `qc_nonbfb`, and `qc_fail`. `qc_base` is the base test case with the default QC namelist. `qc_bfb` is identical to `qc_base`. `qc_nonbfb` is a test that is not bit-for-bit when compared to `qc_base`, but not climate changing. `qc_fail` is a test that is not bit-for-bit and also climate changing.

In order to run the `compare_qc_cases.csh` script, the following requirements must be met:

- Python v2.7 or later
- netcdf Python package
- numpy Python package

To install the necessary Python packages, the `pip` Python utility can be used.

```
pip install --user netCDF4
pip install --user numpy
```

**Note:** Some machines might report `pip: Command not found.` If you encounter this error, check to see if there is any Python module (`module avail python`) that you might need to load prior to using `pip`.

To perform the validation, execute the following commands.

```
# From the CICE base directory
cp configuration/scripts/tests/QC/gen_qc_cases.csh .
cp configuration/scripts/tests/QC/compare_qc_cases.csh

# Create the required test cases
./gen_qc_cases.csh -m <machine> --acct <acct>
```

(continues on next page)

(continued from previous page)

```
# Wait for all 4 jobs to complete

# Perform the comparisons
./compare_qc_cases.csh
```

The `compare_qc_cases.csh` script will run the QC script on the following combinations:

- `qc_base` vs. `qc_bfb`
- `qc_base` vs. `qc_nonbfb`
- `qc_base` vs. `qc_fail`

An example of the output from `compare_qc_cases.csh` is shown below.:

```
===== Running QC tests and writing output to validate_qc.log =====
Running QC test on base and bfb directories.
Expected result: PASSED
Result: PASSED
-----------------------------------------------
Running QC test on base and non-bfb directories.
Expected result: PASSED
Result: PASSED
-----------------------------------------------
Running QC test on base and climate-changing directories.
Expected result: FAILED
Result: FAILED


QC Test has validated
```

## 4.6 Other things

### 4.6.1 Reproducible Sums

Reproducible sums in the CICE diagnostics are set with the namelist *bfbflag*. CICE prognostics results do NOT depend on the global sum implementation. The results are bit-for-bit identical with any *bfbflag*. The *bfbflag* only impacts the results and performance of the global diagnostics written to the CICE log file. For best performance, the off (or lsum8 which is equivalent) setting is recommended. This will probably not produce bit-for-bit results with different decompositions. For bit-for-bit results, the reprosum setting is recommended. This should be only slightly slower than the lsum8 implementation.

Global sums of real types are not reproducible due to different order of operations of the sums of the individual data which introduced roundoff errors. This is caused when the model data is laid out in different block decompositions or on different pe counts so the data is stored in memory in different orders. Integer data should be bit-for-bit identical regardless of the order of operation of the sums.

The *bfbflag* namelist is a character string with several valid settings. The tradeoff in these settings is the likelihood for bit-for-bit results versus their cost. The *bfbflag* settings are implemented as follows,

off is the default and equivalent to lsum8.

lsum4 is a local sum computed with single precision (4 byte) data and a scalar mpi allreduce. This is extremely unlikely to be bit-for-bit for different decompositions. This should generally not be used as the accuracy is very poor for a model implemented with double precision (8 byte) variables.

lsum8 is a local sum computed with double precision data and a scalar mpi allreduce. This is extremely unlikely to be bit-for-bit for different decompositions but is fast. For CICE implemented in double precision, the differences in global sums for different decompositions should be at the roundoff level.

lsum16 is a local sum computed with quadruple precision (16 byte) data and a scalar mpi allreduce. This is very likely to be bit-for-bit for different decompositions. However, it should be noted that this implementation is not available or does not work properly with some compiler and some MPI implementation. Support for quad precision and consistency between underlying fortran and c datatypes can result in inability to compile or incorrect results. The source code associated with this implementation can be turned off with the cpp, NO_R16. Otherwise, it is recommended that this option NOT be used or that results be carefully validated on any platform before it is used.

reprosum is a fixed point method based on ordered double integer sums that requires two scalar reductions per global sum. This is extremely likely to be bfb, but will be slightly more expensive than the lsum algorithms. See [31]

ddpdd is a parallel double-double algorithm using single scalar reduction. This is very likely to be bfb, but is not as fast or accurate as the reprosum implementation. See [10]

### 4.6.2 Adding Timers

Timing any section of code, or multiple sections, consists of defining the timer and then wrapping the code with start and stop commands for that timer. Printing of the timer output is done simultaneously for all timers. To add a timer, first declare it (*timer_[tmr]*) at the top of **ice_timers.F90** (we recommend doing this in both the **mpi/** and **serial/** directories), then add a call to *get_ice_timer* in the subroutine *init_ice_timers*. In the module containing the code to be timed, *call ice_timer_start(timer_[tmr])* at the beginning of the section to be timed, and a similar call to *ice_timer_stop* at the end. A use *ice_timers* statement may need to be added to the subroutine being modified. Be careful not to have one command outside of a loop and the other command inside. Timers can be run for individual blocks, if desired, by including the block ID in the timer calls.

### 4.6.3 Adding History fields

To add a variable to be printed in the history output, search for 'example' in **ice_history_shared.F90**:

1. add a frequency flag for the new field
2. add the flag to the namelist (here and also in **ice_in**)
3. add an index number

and in **ice_history.F90**:

1. broadcast the flag
2. add a call to *define_hist_field*
3. add a call to *accum_hist_field*

The example is for a standard, two-dimensional (horizontal) field; for other array sizes, choose another history variable with a similar shape as an example. Some history variables, especially tracers, are grouped in other files according to their purpose (bgc, melt ponds, etc.).

To add an output frequency for an existing variable, see section *History files*.

### 4.6.4 Adding Tracers

We require that any changes made to the code be implemented in such a way that they can be "turned off" through namelist flags. In most cases, code run with such changes should be bit-for-bit identical with the unmodified code.

Occasionally, non-bit-for-bit changes are necessary, e.g. associated with an unavoidable change in the order of operations. In these cases, changes should be made in stages to isolate the non-bit-for-bit changes, so that those that should be bit-for-bit can be tested separately.

Tracers added to CICE will also require extensive modifications to the Icepack driver, including initialization, namelist flags and restart capabilities. Modifications to the Icepack driver should reflect the modifications needed in CICE but are not expected to match completely. We recommend that the logical namelist variable `tr_[tracer]` be used for all calls involving the new tracer outside of **ice_[tracer].F90**, in case other users do not want to use that tracer.

A number of optional tracers are available in the code, including ice age, first-year ice area, melt pond area and volume, brine height, aerosols, and level ice area and volume (from which ridged ice quantities are derived). Salinity, enthalpies, age, aerosols, level-ice volume, brine height and most melt pond quantities are volume-weighted tracers, while first-year area, pond area, and level-ice area are area-weighted tracers. Biogeochemistry tracers in the skeletal layer are area-weighted, and vertical biogeochemistry tracers are volume-weighted. In the absence of sources and sinks, the total mass of a volume-weighted tracer such as aerosol (kg) is conserved under transport in horizontal and thickness space (the mass in a given grid cell will change), whereas the aerosol concentration (kg/m) is unchanged following the motion, and in particular, the concentration is unchanged when there is surface or basal melting. The proper units for a volume-weighted mass tracer in the tracer array are kg/m.

In several places in the code, tracer computations must be performed on the conserved "tracer volume" rather than the tracer itself; for example, the conserved quantity is $h_{pnd}a_{pnd}a_{lvl}a_i$, not $h_{pnd}$. Conserved quantities are thus computed according to the tracer dependencies (weights), which are tracked using the arrays `trcr_depend` (indicates dependency on area, ice volume or snow volume), `trcr_base` (a dependency mask), `n_trcr_strata` (the number of underlying tracer layers), and `nt_strata` (indices of underlying layers). Additional information about tracers can be found in the Icepack documentation.

To add a tracer, follow these steps using one of the existing tracers as a pattern.

1) **icepack_tracers.F90** and **icepack_[tracer].F90**: declare tracers, add flags and indices, and create physics routines as described in the Icepack documentation
2) **ice_arrays_column.F90**: declare arrays
3) **ice_init_column.F90**: initialize arrays
4) **ice_init.F90**: (some of this may be done in **icepack_[tracer].F90** instead)
   - declare `tr_[tracer]` and `nt_[tracer]` as needed
   - add logical namelist variables `tr_[tracer]`, `restart_[tracer]`
   - initialize and broadcast namelist variables
   - check for potential conflicts, aborting if any occur
   - print namelist variables to diagnostic output file
   - initialize tracer flags etc in icepack (call *icepack_init_tracer_flags* etc)
   - increment number of tracers in use based on namelist input (`ntrcr`)
   - define tracer dependencies
5) **CICE_InitMod.F90**: initialize tracer (includes reading restart file)
6) **CICE_RunMod.F90**, **ice_step_mod.F90** (and elsewhere as needed):
   - call routine to write tracer restart data
   - call Icepack or other routines to update tracer value (often called from **ice_step_mod.F90**)
7) **ice_restart.F90**: define restart variables (for binary, netCDF and PIO)
8) **ice_restart_column.F90**: create routines to read, write tracer restart data

9) **ice_fileunits.F90**: add new dump and restart file units

10) **ice_history_[tracer].F90**: add history variables (Section *Adding History fields*)

11) **ice_in**: add namelist variables to *tracer_nml* and *icefields_nml*. Best practice is to set the namelist values so that the new capability is turned off, and create an option file with your preferred configuration in **configuration/scripts/options**.

12) If strict conservation is necessary, add diagnostics as noted for topo ponds in the Icepack documentation.

13) Update documentation, including **cice_index.rst** and **ug_case_settings.rst**

# CHAPTER 5

# Index of primary variables and parameters

This index defines many of the symbols used frequently in the CICE model code. Values appearing in this list are fixed or recommended; most namelist parameters are indicated ( $E_\circ$) with their default values. For other namelist options, see Section *Table of namelist options*. All quantities in the code are expressed in MKS units (temperatures may take either Celsius or Kelvin units).

Table 1: *Alphabetical Index*

| | | | |
|---|---|---|---|
| **A** | | | |
| a11,a12 | structure tensor components | | |
| a2D | history field accumulations, 2d | | |
| a3Dz | history field accumulations, 3D vertical | | |
| a3Db | history field accumulations, 3D bio grid | | |
| a3Dc | history field accumulations, 3D categories | | |
| a4Di | history field accumulations, 4D categories, vertical ice | | |
| a4Db | history field accumulations, 4D categories, vertical bio grid | | |
| a4Ds | history field accumulations, 4D categories, vertical snow | | |
| a_min | minimum area concentration for computing velocity | 0.001 | |
| a_rapid_mode | • brine channel diameter | | |
| advection | • type of advection algorithm used ('remap' or 'upwind') | remap | |
| ahmax | • thickness above which ice albedo is constant | 0.3m | |
| aice_extmin | minimum value for ice extent diagnostic | 0.15 | |
| aice_init | concentration of ice at beginning of timestep | | |
| aice0 | fractional open water area | | |
| aice(n) | total concentration of ice in grid cell (in category n) | | |
| albedo_type | • type of albedo parameterization ('ccsm3' or 'constant') | | |
| albcnt | counter for averaging albedo | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| albice | bare ice albedo | | |
| albicei | • near infrared ice albedo for thicker ice | | |
| albicev | • visible ice albedo for thicker ice | | |
| albocn | ocean albedo | 0.06 | |
| albpnd | melt pond albedo | | |
| albsno | snow albedo | | |
| albsnowi | • near infrared, cold snow albedo | | |
| albsnowv | • visible, cold snow albedo | | |
| algalN | algal nitrogen concentration | mmol/$m^3$ | |
| alv(n)dr(f) | albedo: visible (near IR), direct (diffuse) | | |
| alv(n)dr(f)_ai | grid-box-mean value of alv(n)dr(f) | | |
| amm | ammonia/um concentration | mmol/$m^3$ | |
| ANGLE | for conversions between the POP grid and latitude-longitude grids | radians | |
| ANGLET | ANGLE converted to T-cells | radians | |
| aparticn | participation function | | |
| apeff_ai | grid-cell-mean effective pond fraction | | |
| apondn | area concentration of melt ponds | | |
| arlx1i | relaxation constant for dynamics (stress) | | |
| araftn | area fraction of rafted ice | | |
| aredistrn | redistribution function: fraction of new ridge area | | |
| ardgn | fractional area of ridged ice | | |
| aspect_rapid_mode | • brine convection aspect ratio | 1 | |
| astar | e-folding scale for participation function | 0.05 | |
| atm_data_dir | • directory for atmospheric forcing data | | |
| atm_data_format | • format of atmospheric forcing files | | |
| atm_data_type | • type of atmospheric forcing | | |
| atmbndy | • atmo boundary layer parameterization (‘default’ or ‘constant’) | | |
| avail_hist_fields | type for history field data | | |
| awtidf | weighting factor for near-ir, diffuse albedo | 0.36218 | |
| awtidr | weighting factor for near-ir, direct albedo | 0.00182 | |
| awtvdf | weighting factor for visible, diffuse albedo | 0.63282 | |
| awtvdr | weighting factor for visible, direct albedo | 0.00318 | |
| **B** | | | |
| bfb_flag | • for bit-for-bit reproducible diagnostics | | |
| bgc_data_dir | • data directory for bgc | | |
| bgc_data_type | • source of silicate, nitrate data | | |
| bgc_flux_type | • ice–ocean flux velocity | | |
| bgc_tracer_type | tracer_type for bgc tracers | | |
| bgrid | nondimensional vertical grid points for bio grid | | |
| bignum | a large number | $10^{30}$ | |
| block | data type for blocks | | |
| block_id | global block number | | |
| block_size_x(y) | number of cells along x(y) direction of block | | |
| blockGlobalID | global block IDs | | |
| blockLocalID | local block IDs | | |
| blockLocation | processor location of block | | |
| blocks_ice | local block IDs | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| bphi | porosity of ice layers on bio grid | | |
| brlx | relaxation constant for dynamics (momentum) | | |
| bTiz | temperature of ice layers on bio grid | | |
| **C** | | | |
| c<n> | real($n$) | | |
| calc_strair | • if true, calculate wind stress | T | |
| calc_Tsfc | • if true, calculate surface temperature | T | |
| Cdn_atm | atmospheric drag coefficient | | |
| Cdn_ocn | ocean drag coefficient | | |
| Cf | • ratio of ridging work to PE change in ridging | 17. | |
| cgrid | vertical grid points for ice grid (compare bgrid) | | |
| char_len | length of character variable strings | 80 | |
| char_len_long | length of longer character variable strings | 256 | |
| check_step | time step on which to begin writing debugging data | | |
| check_umax | if true, check for ice speed $>$ umax_stab | | |
| cldf | cloud fraction | | |
| cm_to_m | cm to meters conversion | 0.01 | |
| coldice | value for constant albedo parameterization | 0.70 | |
| coldsnow | value for constant albedo parameterization | 0.81 | |
| conduct | • conductivity parameterization | | |
| congel | basal ice growth | m | |
| cosw | cosine of the turning angle in water | 1. | |
| coszen | cosine of the zenith angle | | |
| Cp | proportionality constant for potential energy | kg/m$^2$/s$^2$ | |
| cp_air | specific heat of air | 1005.0 J/kg/K | |
| cp_ice | specific heat of fresh ice | 2106. J/kg/K | |
| cp_ocn | specific heat of sea water | 4218. J/kg/K | |
| cp_wv | specific heat of water vapor | 1.81x10$^3$ J/kg/K | |
| cp063 | diffuse fresnel reflectivity (above) | 0.063 | |
| cp455 | diffuse fresnel reflectivity (below) | 0.455 | |
| Cs | fraction of shear energy contributing to ridging | 0.25 | |
| Cstar | constant in Hibler ice strength formula | 20. | |
| cxm | combination of HTN values | | |
| cxp | combination of HTN values | | |
| cym | combination of HTE values | | |
| cyp | combination of HTE values | | |
| **D** | | | |
| daice_da | data assimilation concentration increment rate | | |
| daidtd | ice area tendency due to dynamics/transport | 1/s | |
| daidtt | ice area tendency due to thermodynamics | 1/s | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| dalb_mlt | [see **icepack_shortwave.F90**] | -0.075 | |
| dalb_mlti | [see **icepack_shortwave.F90**] | -0.100 | |
| dalb_mltv | [see **icepack_shortwave.F90**] | -0.150 | |
| darcy_V | Darcy velocity used for brine height tracer | | |
| dardg1(n)dt | rate of fractional area loss by ridging ice (category n) | 1/s | |
| dardg2(n)dt | rate of fractional area gain by new ridges (category n) | 1/s | |
| daymo | number of days in one month | | |
| daycal | day number at end of month | | |
| days_per_year | • number of days in one year | 365 | |
| dbl_kind | definition of double precision | selected_real_kind(13) | |
| dbug | • write extra diagnostics | .false. | |
| Delta | function of strain rates (see Section *Dynamics*) | 1/s | |
| default_season | Season from which initial values of forcing are set. | winter | |
| denom1 | combination of constants for stress equation | | |
| depressT | ratio of freezing temperature to salinity of brine | 0.054 deg/ppt | |
| dhbr_bt | change in brine height at the bottom of the column | | |
| dhbr_top | change in brine height at the top of the column | | |
| dhsn | depth difference for snow on sea ice and pond ice | | |
| diag_file | • diagnostic output file (alternative to standard out) | | |
| diag_type | • where diagnostic output is written | stdout | |
| diagfreq | • how often diagnostic output is written (10 = once per 10 dt) | | |
| distrb | distribution data type | | |
| distrb_info | block distribution information | | |
| distribution_type | • method used to distribute blocks on processors | | |
| distribution_weight | • weighting method used to compute work per block | | |
| divu | strain rate I component, velocity divergence | 1/s | |
| divu_adv | divergence associated with advection | 1/s | |
| dms | dimethyl sulfide concentration | mmol/m$^3$ | |
| dmsp | dimethyl sulfoniopropionate concentration | mmol/m$^3$ | |
| dpscale | • time scale for flushing in permeable ice | $1 \times 10^{-3}$ | |
| dragio | drag coefficient for water on ice | 0.00536 | |
| dSdt_slow_mode | • drainage strength parameter | | |
| dsnow | change in snow thickness | m | |
| dt | • thermodynamics time step | 3600. s | |
| dt_dyn | dynamics/ridging/transport time step | | |
| dT_mlt | • $\Delta$ temperature per $\Delta$ snow grain radius | 1. deg | |
| dte | subcycling time step for EVP dynamics ($\Delta t_e$) | s | |
| dte2T | dte / 2(damping time scale) | | |
| dtei | 1/dte, where dte is the EVP subcycling time step | 1/s | |
| dump_file | • output file for restart dump | | |
| dumpfreq | • dump frequency for restarts, y, m, d, h or 1 | | |
| dumpfreq_n | • restart output frequency | | |
| dump_last | • if true, write restart on last time step of simulation | | |
| dxhy | combination of HTE values | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| dxt | width of T cell ($\Delta x$) through the middle | m | |
| dxu | width of U cell ($\Delta x$) through the middle | m | |
| dyhx | combination of HTN values | | |
| dyn_dt | dynamics and transport time step ($\Delta t_{dyn}$) | s | |
| dyt | height of T cell ($\Delta y$) through the middle | m | |
| dyu | height of U cell ($\Delta y$) through the middle | m | |
| dvidtd | ice volume tendency due to dynamics/transport | m/s | |
| dvidtt | ice volume tendency due to thermodynamics | m/s | |
| dvirdg(n)dt | ice volume ridging rate (category n) | m/s | |
| **E** | | | |
| e11, e12, e22 | strain rate tensor components | | |
| ecci | yield curve minor/major axis ratio, squared | 1/4 | |
| eice(n) | energy of melting of ice per unit area (in category n) | J/m$^2$ | |
| emissivity | emissivity of snow and ice | 0.95 | |
| eps13 | a small number | $10^{-13}$ | |
| eps16 | a small number | $10^{-16}$ | |
| esno(n) | energy of melting of snow per unit area (in category n) | J/m$^2$ | |
| evap | evaporative water flux | kg/m$^2$/s | |
| ew_boundary_type | • type of east-west boundary condition | | |
| eyc | coefficient for calculating the parameter E, 0< eyc <1 | 0.36 | |
| **F** | | | |
| faero_atm | aerosol deposition rate | kg/m$^2$/s | |
| faero_ocn | aerosol flux to the ocean | kg/m$^2$/s | |
| fbot_xfer_type | • type of heat transfer coefficient under ice | | |
| fcondtop(n)(_f) | conductive heat flux | W/m$^2$ | |
| fcor_blk | Coriolis parameter | 1/s | |
| ferrmax | max allowed energy flux error (thermodynamics) | 1x $10^{-3}$ W/m$^2$ | |
| ffracn | fraction of fsurfn used to melt pond ice | | |
| fhocn | net heat flux to ocean | W/m$^2$ | |
| fhocn_ai | grid-box-mean net heat flux to ocean (fhocn) | W/m$^2$ | |
| field_loc_center | field centered on grid cell | 1 | |
| field_loc_Eface | field centered on east face | 4 | |
| field_loc_NEcorner | field on northeast corner | 2 | |
| field_loc_Nface | field centered on north face | 3 | |
| field_loc_noupdate | ignore location of field | -1 | |
| field_loc_unknown | unknown location of field | 0 | |
| field_loc_Wface | field centered on west face | 5 | |
| field_type_angle | angle field type | 3 | |
| field_type_noupdate | ignore field type | -1 | |
| field_type_scalar | scalar field type | 1 | |
| field_type_unknown | unknown field type | 0 | |
| field_type_vector | vector field type | 2 | |
| first_ice | flag for initial ice formation | | |
| flat | latent heat flux | W/m$^2$ | |
| floediam | effective floe diameter for lateral melt | 300. m | |
| floeshape | floe shape constant for lateral melt | 0.66 | |
| flux_bio | all biogeochemistry fluxes passed to ocean | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| flux_bio_ai | all biogeochemistry fluxes passed to ocean, grid cell mean | | |
| flw | incoming longwave radiation | W/m$^2$ | |
| flwout | outgoing longwave radiation | W/m$^2$ | |
| fm | Coriolis parameter * mass in U cell | kg/s | |
| formdrag | • calculate form drag | | |
| fpond | fresh water flux to ponds | kg/m$^2$/s | |
| fr_resp | bgc respiration fraction | 0.05 | |
| frain | rainfall rate | kg/m$^2$/s | |
| frazil | frazil ice growth | m | |
| fresh | fresh water flux to ocean | kg/m$^2$/s | |
| fresh_ai | grid-box-mean fresh water flux (fresh) | kg/m$^2$/s | |
| frz_onset | day of year that freezing begins | | |
| frzmlt | freezing/melting potential | W/m$^2$ | |
| frzmlt_init | freezing/melting potential at beginning of time step | W/m$^2$ | |
| frzmlt_max | maximum magnitude of freezing/melting potential | 1000. W/m$^2$ | |
| frzpnd | • Stefan refreezing of melt ponds | 'hlid' | |
| fsalt | net salt flux to ocean | kg/m$^2$/s | |
| fsalt_ai | grid-box-mean salt flux to ocean (fsalt) | kg/m$^2$/s | |
| fsens | sensible heat flux | W/m$^2$ | |
| fsnow | snowfall rate | kg/m$^2$/s | |
| fsnowrdg | snow fraction that survives in ridging | 0.5 | |
| fsurf(n)(_f) | net surface heat flux excluding fcondtop | W/m$^2$ | |
| fsw | incoming shortwave radiation | W/m$^2$ | |
| fswabs | total absorbed shortwave radiation | W/m$^2$ | |
| fswfac | scaling factor to adjust ice quantities for updated data | | |
| fswint | shortwave absorbed in ice interior | W/m$^2$ | |
| fswpenl | shortwave penetrating through ice layers | W/m$^2$ | |
| fswthru | shortwave penetrating to ocean | W/m$^2$ | |
| fswthru_ai | grid-box-mean shortwave penetrating to ocean (fswthru) | W/m$^2$ | |
| fyear | current data year | | |
| fyear_final | last data year | | |
| fyear_init | • initial data year | | |
| **G** | | | |
| gravit | gravitational acceleration | 9.80616 m/s$^2$ | |
| grid_file | • input file for grid info | | |
| grid_format | • format of grid files | | |
| grid_type | • 'rectangular', 'displaced_pole', 'column' or 'regional' | | |
| gridcpl_file | • input file for coupling grid info | | |
| grow_net | specific biogeochemistry growth rate per grid cell | s $^{-1}$ | |
| Gstar | piecewise-linear ridging participation function parameter | 0.15 | |
| **H** | | | |
| halo_info | information for updating ghost cells | | |
| heat_capacity | • if true, use salinity-dependent thermodynamics | T | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| hfrazilmin | minimum thickness of new frazil ice | 0.05 m | |
| hi_min | minimum ice thickness for thinnest ice category | 0.01 m | |
| hi_ssl | ice surface scattering layer thickness | 0.05 m | |
| hicen | ice thickness in category n | m | |
| highfreq | • high-frequency atmo coupling | F | |
| hin_old | ice thickness prior to growth/melt | m | |
| hin_max | category thickness limits | m | |
| hist_avg | • if true, write averaged data instead of snapshots | T | |
| histfreq | • units of history output frequency: y, m, w, d or 1 | | |
| histfreq_n | • integer output frequency in histfreq units | | |
| history_dir | • path to history output files | | |
| history_file | • history output file prefix | | |
| hm | land/boundary mask, thickness (T-cell) | | |
| hmix | ocean mixed layer depth | 20. m | |
| hour | hour of the year | | |
| hp0 | pond depth at which shortwave transition to bare ice occurs | 0.2 m | |
| hp1 | • critical ice lid thickness for topo ponds (dEdd) | 0.01 m | |
| hpmin | minimum melt pond depth (shortwave) | 0.005 m | |
| hpondn | melt pond depth | m | |
| hs_min | minimum thickness for which $T_s$ is computed | $1.\times10^{-4}$ m | |
| hs0 | • snow depth at which transition to ice occurs (dEdd) | 0.03 m | |
| hs1 | • snow depth of transition to pond ice | 0.03 m | |
| hs_ssl | snow surface scattering layer thickness | 0.04 m | |
| Hstar | determines mean thickness of ridged ice | 25. m | |
| HTE | length of eastern edge ($\Delta y$) of T-cell | m | |
| HTN | length of northern edge ($\Delta x$) of T-cell | m | |
| HTS | length of southern edge ($\Delta x$) of T-cell | m | |
| HTW | length of western edge of ($\Delta y$) T-cell | m | |
| **I** | | | |
| i(j)_glob | global domain location for each grid cell | | |
| i0vis | fraction of penetrating visible solar radiation | 0.70 | |
| iblkp | block on which to write debugging data | | |
| i(j)block | Cartesian i,j position of block | | |
| ice_hist_field | type for history variables | | |
| ice_ic | • choice of initial conditions (see *Ice Initial State*) | | |
| ice_stdout | unit number for standard output | | |
| ice_stderr | unit number for standard error output | | |
| ice_ref_salinity | reference salinity for ice–ocean exchanges | 4. ppt | |
| icells | number of grid cells with specified property (for vectorization) | | |
| iceruf | ice surface roughness | $5.\times10^{-4}$ m | |
| icetmask | ice extent mask (T-cell) | | |
| iceumask | ice extent mask (U-cell) | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| idate | the date at the end of the current time step (yyyymmdd) | | |
| idate0 | initial date | | |
| ierr | general-use error flag | | |
| igrid | interface points for vertical bio grid | | |
| i(j)hi | last i(j) index of physical domain (local) | | |
| i(j)lo | first i(j) index of physical domain (local) | | |
| incond_dir | • directory to write snapshot of initial condition | | |
| incond_file | • prefix for initial condition file name | | |
| int_kind | definition of an integer | selected_real_kind(6) | |
| integral_order | polynomial order of quadrature integrals in remapping | 3 | |
| ip, jp | local processor coordinates on which to write debugging data | | |
| istep | local step counter for time loop | | |
| istep0 | • number of steps taken in previous run | 0 | |
| istep1 | total number of steps at current time step | | |
| Iswabs | shortwave radiation absorbed in ice layers | W/m$^2$ | |
| **J** | | | |
| **K** | | | |
| kalg | • absorption coefficient for algae | | |
| kappav | visible extinction coefficient in ice, wavelength<700nm | 1.4 m$^{-1}$ | |
| kcatbound | • category boundary formula | | |
| kdyn | • type of dynamics (1 = EVP, 0 = off) | 1 | |
| kg_to_g | kg to g conversion factor | 1000. | |
| kice | thermal conductivity of fresh ice ([3]) | 2.03 W/m/deg | |
| kitd | • type of itd conversions (0 = delta function, 1 = linear remap) | 1 | |
| kmt_file | • input file for land mask info | | |
| krdg_partic | • ridging participation function | 1 | |
| krdg_redist | • ridging redistribution function | 1 | |
| krgdn | mean ridge thickness per thickness of ridging ice | | |
| kseaice | thermal conductivity of ice for zero-layer thermodynamics | 2.0 W/m/deg | |
| ksno | thermal conductivity of snow | 0.30 W/m/deg | |
| kstrength | • ice stength formulation (1= [39], 0 = [11]) | 1 | |
| ktherm | • thermodynamic formulation (0 = zero-layer, 1 = [3], 2 = mushy) | | |
| **L** | | | |
| l_brine | flag for brine pocket effects | | |
| l_conservation_check | if true, check conservation when ridging | | |
| l_fixed_area | flag for prescribing remapping fluxes | | |
| l_mpond_fresh | • if true, retain (topo) pond water until ponds drain | | |
| latpnt | • desired latitude of diagnostic points | degrees N | |
| latt(u)_bounds | latitude of T(U) grid cell corners | degrees N | |
| lcdf64 | • if true, use 64-bit format | | |
| Lfresh | latent heat of melting of fresh ice = Lsub - Lvap | J/kg | |
| lhcoef | transfer coefficient for latent heat | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| lmask_n(s) | northern (southern) hemisphere mask | | |
| local_id | local address of block in current distribution | | |
| log_kind | definition of a logical variable | kind(.true.) | |
| lonpnt | • desired longitude of diagnostic points | degrees E | |
| lont(u)_bounds | longitude of T(U) grid cell corners | degrees E | |
| Lsub | latent heat of sublimation for fresh water | $2.835\times 10^6$ J/kg | |
| ltripole_grid | flag to signal use of tripole grid | | |
| Lvap | latent heat of vaporization for fresh water | $2.501\times 10^6$ J/kg | |
| **M** | | | |
| m_min | minimum mass for computing velocity | 0.01 kg/m$^2$ | |
| m_to_cm | meters to cm conversion | 100. | |
| m1 | constant for lateral melt rate | $1.6\times10^{-6}$ m/s deg$^{-m2}$ | |
| m2 | constant for lateral melt rate | 1.36 | |
| m2_to_km2 | m$^2$ to km$^2$ conversion | $1\times10^{-6}$ | |
| maskhalo_bound | • turns on *bound_state* halo masking | | |
| maskhalo_dyn | • turns on dynamics halo masking | | |
| maskhalo_remap | • turns on transport halo masking | | |
| master_task | task ID for the controlling processor | | |
| max_blocks | maximum number of blocks per processor | | |
| max_ntrcr | maximum number of tracers available | 5 | |
| maxraft | maximum thickness of ice that rafts | 1. m | |
| mday | day of the month | | |
| meltb | basal ice melt | m | |
| meltl | lateral ice melt | m | |
| melts | snow melt | m | |
| meltt | top ice melt | m | |
| min_salin | threshold for brine pockets | 0.1 ppt | |
| mlt_onset | day of year that surface melt begins | | |
| month | the month number | | |
| monthp | previous month number | | |
| mps_to_cmpdy | m per s to cm per day conversion | $8.64\times10^6$ | |
| mtask | local processor number that writes debugging data | | |
| mu_rdg | • e-folding scale of ridged ice | | |
| my_task | task ID for the current processor | | |
| **N** | | | |
| n_aero | number of aerosol species | | |
| natmiter | • number of atmo boundary layer iterations | 5 | |
| nblocks | number of blocks on current processor | | |
| nblocks_tot | total number of blocks in decomposition | | |
| nblocks_x(y) | total number of blocks in x(y) direction | | |
| nbtrcr | number of biology tracers | | |
| ncat | number of ice categories | 5 | |
| ncat_hist | number of categories written to history | | |
| ndte | • number of subcycles | 120 | |
| ndtd | • number of dynamics/advection steps under thermo | 1 | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| new_day | flag for beginning new day | | |
| new_hour | flag for beginning new hour | | |
| new_month | flag for beginning new month | | |
| new_year | flag for beginning new year | | |
| nghost | number of rows of ghost cells surrounding each subdomain | 1 | |
| ngroups | number of groups of flux triangles in remapping | 5 | |
| nhlat | northern latitude of artificial mask edge | 30°S | |
| nilyr | number of ice layers in each category | 4 | |
| nit | nitrate concentration | mmol/$m^3$ | |
| nlt_bgc_[chem] | ocean sources and sinks for biogeochemistry | | |
| nml_filename | namelist file name | | |
| nprocs | • total number of processors | | |
| npt | • total number of time steps (dt) | | |
| ns_boundary_type | • type of north-south boundary condition | | |
| nslyr | number of snow layers in each category | | |
| nspint | number of solar spectral intervals | | |
| nstreams | number of history output streams (frequencies) | | |
| nt_<trcr> | tracer index | | |
| ntrace | number of fields being transported | | |
| ntrcr | number of tracers | | |
| nu_diag | unit number for diagnostics output file | | |
| nu_dump | unit number for dump file for restarting | | |
| nu_dump_eap | unit number for EAP dynamics dump file for restarting | | |
| nu_dump_[tracer] | unit number for tracer dump file for restarting | | |
| nu_forcing | unit number for forcing data file | | |
| nu_grid | unit number for grid file | | |
| nu_hdr | unit number for binary history header file | | |
| nu_history | unit number for history file | | |
| nu_kmt | unit number for land mask file | | |
| nu_nml | unit number for namelist input file | | |
| nu_restart | unit number for restart input file | | |
| nu_restart_eap | unit number for EAP dynamics restart input file | | |
| nu_restart_[tracer] | unit number for tracer restart input file | | |
| nu_rst_pointer | unit number for pointer to latest restart file | | |
| num_avail_hist_fields_[shape] | number of history fields of each array shape | | |
| nvar | number of horizontal grid fields written to history | | |
| nvarz | number of category, vertical grid fields written to history | | |
| nx(y)_block | total number of gridpoints on block in x(y) direction | | |
| nx(y)_global | number of physical gridpoints in x(y) direction, global domain | | |
| nyr | year number | | |
| **O** | | | |
| ocean_bio | concentrations of bgc constituents in the ocean | | |
| oceanmixed_file | • data file containing ocean forcing data | | |
| oceanmixed_ice | • if true, use internal ocean mixed layer | | |
| ocn_data_dir | • directory for ocean forcing data | | |
| ocn_data_format | • format of ocean forcing files | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| ocn_data_type | • source of surface temperature, salinity data | | |
| omega | angular velocity of Earth | $7.292\times10^{-5}$ rad/s | |
| opening | rate of ice opening due to divergence and shear | 1/s | |
| **P** | | | |
| p001 | 1/1000 | | |
| p01 | 1/100 | | |
| p025 | 1/40 | | |
| p027 | 1/36 | | |
| p05 | 1/20 | | |
| p055 | 1/18 | | |
| p1 | 1/10 | | |
| p111 | 1/9 | | |
| p15 | 15/100 | | |
| p166 | 1/6 | | |
| p2 | 1/5 | | |
| p222 | 2/9 | | |
| p25 | 1/4 | | |
| p333 | 1/3 | | |
| p4 | 2/5 | | |
| p5 | 1/2 | | |
| p52083 | 25/48 | | |
| p5625m | -9/16 | | |
| p6 | 3/5 | | |
| p666 | 2/3 | | |
| p75 | 3/4 | | |
| phi_c_slow_mode | • critical liquid fraction | | |
| phi_i_mushy | • solid fraction at lower boundary | | |
| phi_sk | skeletal layer porosity | | |
| phi_snow | • snow porosity for brine height tracer | | |
| pi | $\pi$ | | |
| pi2 | $2\pi$ | | |
| pih | $\pi/2$ | | |
| piq | $\pi/4$ | | |
| pi(j,b,m)loc | x (y, block, task) location of diagnostic points | | |
| plat | grid latitude of diagnostic points | | |
| plon | grid longitude of diagnostic points | | |
| pndaspect | • aspect ratio of pond changes (depth:area) | 0.8 | |
| pointer_file | • input file for restarting | | |
| potT | atmospheric potential temperature | K | |
| PP_net | total primary productivity per grid cell | mg C/m$^2$/s | |
| precip_units | • liquid precipitation data units | | |
| print_global | • if true, print global data | F | |
| print_points | • if true, print point data | F | |
| processor_shape | • descriptor for processor aspect ratio | | |
| prs_sig | replacement pressure | N/m | |
| Pstar | ice strength parameter | $2.75\times10^{4}$N/m | |
| puny | a small positive number | $1\times10^{-11}$ | |
| **Q** | | | |
| Qa | specific humidity at 10 m | kg/kg | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| qdp | deep ocean heat flux | W/m$^2$ | |
| qqqice | for saturated specific humidity over ice | $1.16378\times10^7$kg/m$^3$ | |
| qqqocn | for saturated specific humidity over ocean | $6.275724\times10^6$kg/m$^3$ | |
| Qref | 2m atmospheric reference specific humidity | kg/kg | |
| **R** | | | |
| R_C2N | algal carbon to nitrate factor | 7. mole/mole | |
| R_gC2molC | mg/mmol carbon | 12.01 mg/mole | |
| R_chl2N | algal chlorophyll to nitrate factor | 3. mg/mmol | |
| R_ice | • parameter for Delta-Eddington ice albedo | | |
| R_pnd | • parameter for Delta-Eddington pond albedo | | |
| R_S2N | algal silicate to nitrate factor | 0.03 mole/mole | |
| R_snw | • parameter for Delta-Eddington snow albedo | | |
| r16_kind | definition of quad precision | selected_real_kind(26) | |
| Rac_rapid_mode | • critical Rayleigh number | 10 | |
| rad_to_deg | degree-radian conversion | $180/\pi$ | |
| radius | earth radius | $6.37\times10^6$ m | |
| rdg_conv | convergence for ridging | 1/s | |
| rdg_shear | shear for ridging | 1/s | |
| real_kind | definition of single precision real | selected_real_kind(6) | |
| refindx | refractive index of sea ice | 1.310 | |
| revp | real(revised_evp) | | |
| restart | • if true, initialize using restart file instead of defaults | T | |
| restart_age | • if true, read age restart file | | |
| restart_bgc | • if true, read bgc restart file | | |
| restart_dir | • path to restart/dump files | | |
| restart_file | • restart file prefix | | |
| restart_format | • restart file format | | |
| restart_[tracer] | • if true, read tracer restart file | | |
| restart_ext | • if true, read/write halo cells in restart file | | |
| restore_bgc | • if true, restore nitrate/silicate to data | | |
| restore_ice | • if true, restore ice state along lateral boundaries | | |
| restore_ocn | • restore sst to data | | |
| revised_evp | • if true, use revised EVP parameters and approach | | |
| rfracmin | • minimum melt water fraction added to ponds | 0.15 | |
| rfracmax | • maximum melt water fraction added to ponds | 1.0 | |
| rhoa | air density | kg/m$^3$ | |
| rhofresh | density of fresh water | 1000.0 kg/m$^3$ | |
| rhoi | density of ice | 917. kg/m$^3$ | |
| rhos | density of snow | 330. kg/m$^3$ | |
| rhosi | average sea ice density (for hbrine tracer) | 940. kg/m$^3$ | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| rhow | density of seawater | 1026. kg/m$^3$ | |
| rnilyr | real(nlyr) | | |
| rside | fraction of ice that melts laterally | | |
| rsnw_fresh | freshly fallen snow grain radius | 100. $\times 10^{-6}$ m | |
| rsnw_melt | • melting snow grain radius | 1000. $\times 10^{-6}$ m | |
| rsnw_nonmelt | nonmelting snow grain radius | 500. $\times 10^{-6}$ m | |
| rsnw_sig | standard deviation of snow grain radius | 250. $\times 10^{-6}$ m | |
| runid | • identifier for run | | |
| runtype | • type of initialization used | | |
| **S** | | | |
| s11, s12, s22 | stress tensor components | | |
| saltmax | max salinity, at ice base ([3]) | 3.2 ppt | |
| scale_factor | scaling factor for shortwave radiation components | | |
| sec | seconds elasped into idate | | |
| secday | number of seconds in a day | 86400. | |
| shcoef | transfer coefficient for sensible heat | | |
| shear | strain rate II component | 1/s | |
| shlat | southern latitude of artificial mask edge | 30°N | |
| shortwave | • flag for shortwave parameterization ('ccsm3' or 'dEdd') | | |
| sig1(2) | principal stress components (diagnostic) | | |
| sil | silicate concentration | mmol/m$^3$ | |
| sinw | sine of the turning angle in water | 0. | |
| Sinz | ice salinity profile | ppt | |
| sk_l | skeletal layer thickness | 0.03 m | |
| snoice | snow–ice formation | m | |
| snowpatch | length scale for parameterizing nonuniform snow coverage | 0.02 m | |
| skl_bgc | • biogeochemistry on/off | | |
| spval | special value (single precision) | $10^{30}$ | |
| spval_dbl | special value (double precision) | $10^{30}$ | |
| ss_tltx(y) | sea surface in the x(y) direction | m/m | |
| sss | sea surface salinity | ppt | |
| sst | sea surface temperature | C | |
| Sswabs | shortwave radiation absorbed in snow layers | W/m$^2$ | |
| stefan-boltzmann | Stefan-Boltzmann constant | $5.67\times10^{-8}$ W/m$^2$K$^4$ | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| stop_now | if 1, end program execution | | |
| strairx(y) | stress on ice by air in the x(y)-direction (centered in U cell) | N/m$^2$ | |
| strairx(y)T | stress on ice by air, x(y)-direction (centered in T cell) | N/m$^2$ | |
| strax(y) | wind stress components from data | N/m$^2$ | |
| strength | ice strength (pressure) | N/m | |
| stress12 | internal ice stress, $\sigma_{12}$ | N/m | |
| stressm | internal ice stress, $\sigma_{11} - \sigma_{22}$ | N/m | |
| stressp | internal ice stress, $\sigma_{11} + \sigma_{22}$ | N/m | |
| strintx(y) | divergence of internal ice stress, x(y) | N/m$^2$ | |
| strocnx(y) | ice–ocean stress in the x(y)-direction (U-cell) | N/m$^2$ | |
| strocnx(y)T | ice–ocean stress, x(y)-dir. (T-cell) | N/m$^2$ | |
| strtltx(y) | surface stress due to sea surface slope | N/m$^2$ | |
| swv(n)dr(f) | incoming shortwave radiation, visible (near IR), direct (diffuse) | W/m$^2$ | |
| **T** | | | |
| Tair | air temperature at 10 m | K | |
| tarea | area of T-cell | m$^2$ | |
| tarean | area of northern hemisphere T-cells | m$^2$ | |
| tarear | 1/tarea | 1/m$^2$ | |
| tareas | area of southern hemisphere T-cells | m$^2$ | |
| tcstr | string identifying T grid for history variables | | |
| tday | absolute day number | | |
| Tf | freezing temperature | C | |
| Tffresh | freezing temp of fresh ice | 273.15 K | |
| tfrz_option | • form of ocean freezing temperature | | |
| thinS | minimum ice thickness for brine tracer | | |
| time | total elapsed time | s | |
| time_beg | beginning time for history averages | | |
| time_bounds | beginning and ending time for history averages | | |
| time_end | ending time for history averages | | |
| time_forc | time of last forcing update | s | |
| Timelt | melting temperature of ice top surface | 0. C | |
| tinyarea | puny * tarea | m$^2$ | |
| Tinz | Internal ice temperature | C | |
| TLAT | latitude of cell center | radians | |
| TLON | longitude of cell center | radians | |
| tmask | land/boundary mask, thickness (T-cell) | | |
| tmass | total mass of ice and snow | kg/m$^2$ | |
| Tmin | minimum allowed internal temperature | -100. C | |
| Tmltz | melting temperature profile of ice | | |
| Tocnfrz | temperature of constant freezing point parameterization | -1.8 C | |
| tr_aero | • if true, use aerosol tracers | | |
| tr_bgc_[tracer] | • if true, use biogeochemistry tracers | | |
| tr_brine | • if true, use brine height tracer | | |
| tr_FY | • if true, use first-year area tracer | | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| tr_iage | • if true, use ice age tracer | | |
| tr_lvl | • if true, use level ice area and volume tracers | | |
| tr_pond_cesm | • if true, use CESM melt pond scheme | | |
| tr_pond_lvl | • if true, use level-ice melt pond scheme | | |
| tr_pond_topo | • if true, use topo melt pond scheme | | |
| trcr | ice tracers | | |
| trcr_depend | tracer dependency on basic state variables | | |
| Tref | 2m atmospheric reference temperature | K | |
| trestore | • restoring time scale | days | |
| tripole | if true, block lies along tripole boundary | | |
| tripoleT | if true, tripole boundary is T-fold; if false, U-fold | | |
| Tsf_errmax | max allowed $T_{sf}$ error (thermodynamics) | $5.\times10^{-4}$deg | |
| Tsfc(n) | temperature of ice/snow top surface (in category n) | C | |
| Tsnz | Internal snow temperature | C | |
| Tsmelt | melting temperature of snow top surface | 0. C | |
| TTTice | for saturated specific humidity over ice | 5897.8 K | |
| TTTocn | for saturated specific humidity over ocean | 5107.4 K | |
| **U** | | | |
| uarea | area of U-cell | m $^{2}$ | |
| uarear | 1/uarea | m $^{-2}$ | |
| uatm | wind velocity in the x direction | m/s | |
| ULAT | latitude of U-cell centers | radians | |
| ULON | longitude of U-cell centers | radians | |
| umask | land/boundary mask, velocity (U-cell) | | |
| umax_stab | ice speed threshold (diagnostics) | 1. m/s | |
| umin | min wind speed for turbulent fluxes | 1. m/s | |
| uocn | ocean current in the x-direction | m/s | |
| update_ocn_f | • if true, include frazil ice fluxes in ocean flux fields | | |
| use_leap_years | • if true, include leap days | | |
| use_restart_time | • if true, use date from restart file | | |
| ustar_min | • minimum friction velocity under ice | | |
| ucstr | string identifying U grid for history variables | | |
| uvel | x-component of ice velocity | m/s | |
| uvel_init | x-component of ice velocity at beginning of time step | m/s | |
| uvm | land/boundary mask, velocity (U-cell) | | |
| **V** | | | |
| vatm | wind velocity in the y direction | m/s | |
| vice(n) | volume per unit area of ice (in category n) | m | |
| vicen_init | ice volume at beginning of timestep | m | |
| viscosity_dyn | dynamic viscosity of brine | $1.79 \times 10^{-3}$ kg/m/s | |
| vocn | ocean current in the y-direction | m/s | |
| vonkar | von Karman constant | 0.4 | |
| vraftn | volume of rafted ice | m | |

Continued on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| vrdgn | volume of ridged ice | m | |
| vredistrn | redistribution function: fraction of new ridge volume | | |
| vsno(n) | volume per unit area of snow (in category n) | m | |
| vvel | y-component of ice velocity | m/s | |
| vvel_init | y-component of ice velocity at beginning of time step | m/s | |
| **W** | | | |
| warmice | value for constant albedo parameterization | 0.68 | |
| warmsno | value for constant albedo parameterization | 0.77 | |
| wind | wind speed | m/s | |
| write_history | if true, write history now | | |
| write_ic | • if true, write initial conditions | | |
| write_restart | if 1, write restart now | | |
| **X** | | | |
| **Y** | | | |
| ycycle | • number of years in forcing data cycle | | |
| yday | day of the year | | |
| yield_curve | type of yield curve | ellipse | |
| yieldstress11(12, 22) | yield stress tensor components | | |
| year_init | • the initial year | | |
| **Z** | | | |
| zlvl | atmospheric level height | m | |
| zref | reference height for stability | 10. m | |
| zTrf | reference height for $T_{ref}$, $Q_{ref}$, $U_{ref}$ | 2. m | |
| zvir | gas constant (water vapor)/gas constant (air) - 1 | 0.606 | |

CHAPTER 6

# References

### References

- search

# Bibliography


[1] T.L. Amundrud, H. Malling, and R.G. Ingram. Geometrical constraints on the evolution of ridged sea ice. *J. Geophys. Res. Oceans*, 2004. URL: http://dx.doi.org/10.1029/2003JC002251.

[2] C. Konig Beatty and D.M. Holland. Modeling landfast ice by adding tensile strength. *J. Phys. Oceanogr.*, 40:185–198, 2010. URL: http://dx.doi.org/10.1175/2009JPO4105.1.

[3] C.M. Bitz and W.H. Lipscomb. An energy-conserving thermodynamic sea ice model for climate study. *J. Geophys. Res. Oceans*, 104(C7):15669–15677, 1999. URL: http://dx.doi.org/10.1029/1999JC900100.

[4] S. Bouillon, T. Fichefet, V. Legat, and G. Madec. The elastic-viscous-plastic method revisited. *Ocean Modelling*, 71:1–12, 2013. URL: http://dx.doi.org/10.1016/j.ocemod.2013.05.013.

[5] W.M. Connolley, J.M. Gregory, E.C. Hunke, and A.J. McLaren. On the consistent scaling of terms in the sea ice dynamics equation. *J. Phys. Oceanogr.*, 34:1776–1780, 2004. URL: http://dx.doi.org/10.1175/1520-0485(2004)034\T1\textless{}1776:OTCSOT\T1\textgreater{}2.0.CO;2.

[6] A. Craig, S. Mickelson, E.C. Hunke, and D. Bailey. Improved parallel performance of the CICE model in CESM1. *Int. J High Perform. Comput. Appl*, 29(2):154–165, 2014. URL: http://dx.doi.org/10.1177/1094342014548771.

[7] J.K. Dukowicz and J.R. Baumgardner. Incremental remapping as a transport/advection algorithm. *J. Comput. Phys.*, 160:318–335, 2000. URL: http://dx.doi.org/10.1006/jcph.2000.6465.

[8] G.M. Flato and W.D. Hibler. Ridging and strength in modeling the thickness distribution of Arctic sea ice. *J. Geophys. Res. Oceans*, 100:18611–18626, 1995. URL: http://dx.doi.org/10.1029/95JC02091.

[9] C.A. Geiger, W.D. Hibler, and S.F. Ackley. Large-scale sea ice drift and deformation: Comparison between models and observations in the western Weddell Sea during 1992. *J. Geophys. Res. Oceans*, 103:21893–21913, 1998. URL: http://dx.doi.org/10.1029/98JC01258.

[10] Y. He and C.H.Q. Ding. Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications. *The Journal of Supercomputing*, 18:259–277, 2001. URL: http://dx.doi.org/10.1023/A:1008153532043.

[11] W.D. Hibler. A dynamic thermodynamic sea ice model. *J. Phys. Oceanogr.*, 9:817–846, 1979. URL: http://dx.doi.org/10.1175/1520-0485(1979)009\T1\textless{}0815:ADTSIM\T1\textgreater{}2.0.CO;2.

[12] W.D. Hibler. Modeling a variable thickness sea ice cover. *Mon. Wea. Rev.*, 108:1943–1973, 1980. URL: http://dx.doi.org/10.1175/1520-0493(1980)108\T1\textless{}1943:MAVTSI\T1\textgreater{}2.0.CO;2.

[13] W.D. Hibler and K. Bryan. A diagnostic ice-ocean model. *J. Phys. Oceanogr.*, 17:987–1015, 1987. URL: http://dx.doi.org/10.1175/1520-0485(1987)017\T1\textless{}0987:ADIM\T1\textgreater{}2.0.CO;2.

[14] W.D. Hibler, A. Roberts, P. Heil, A.Y. Proshutinsky, H.L. Simmons, and J. Lovick. Modeling M2 tidal variability in Arctic sea-ice drift and deformation. *Ann. Glaciol.*, 2006. URL: http://dx.doi.org/10.3189/172756406781811178.

[15] E.C. Hunke. Viscous-plastic sea ice dynamics with the EVP model: Linearization issues. *J. Comp. Phys.*, 170:18–38, 2001. URL: http://dx.doi.org/10.1006/jcph.2001.6710.

[16] E.C. Hunke and J.K. Dukowicz. An elastic-viscous-plastic model for sea ice dynamics. *J. Phys. Oceanogr.*, 27:1849–1867, 1997. URL: http://dx.doi.org/10.1175/1520-0485(1997)027\T1\textless{}1849:AEVPMF\T1\textgreater{}2.0.CO;2.

[17] E.C. Hunke and J.K. Dukowicz. The Elastic-Viscous-Plastic sea ice dynamics model in general orthogonal curvilinear coordinates on a sphere—Effect of metric terms. *Mon. Wea. Rev.*, 130:1848–1865, 2002. URL: http://dx.doi.org/10.1175/1520-0493(2002)130\T1\textless{}1848:TEVPSI\T1\textgreater{}2.0.CO;2.

[18] E.C. Hunke and J.K. Dukowicz. *The sea ice momentum equation in the free drift regime*. Technical Report LA-UR-03-2219, Los Alamos National Laboratory, 2003. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/LAUR-03-2219.pdf.

[19] E.C. Hunke, A. Roberts, R. Allard, J.F. Lemieux, M. Turner, A.P. Craig, A.K. DuVivier, D. Bailey, M.M. Holland, M. Winton, F. Dupont, and R. Grumbine. The CICE Consortium Sea Ice Modeling Suite. *In Prep.*, 2018. URL: http://dx.doi.org/IN-PROGRESS.

[20] E.C. Hunke and Y. Zhang. A comparison of sea ice dynamics models at high resolution. *Mon. Wea. Rev.*, 127:396–408, 1999. URL: http://dx.doi.org/10.1175/1520-0493(1999)127\T1\textless{}0396:ACOSID\T1\textgreater{}2.0.CO;2.

[21] M. Jin, C. Deal, J. Wang, K.H. Shin, N. Tanaka, T.E. Whiteledge, S.H. Lee, and R.R. Gradinger. Controls of the landfast ice-ocean ecosystem offshore Barrow, Alaska. *Ann. Glaciol.*, 44:63–72, 2006. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/JDWSTWLG06.pdf.

[22] B.G. Kauffman and W.G. Large. *The CCSM coupler, version 5.0.1*. 2002. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/KL_NCAR2002.pdf.

[23] M. Kimmritz, S. Danilov, and M. Losch. On the convergence of the modified elastic-viscous-plastic method for solving the sea ice momentum equation. *J. Comp. Phys.*, 296:90–100, 2015. URL: http://dx.doi.org/10.1016/j.jcp.2015.04.051.

[24] J.F. Lemieux, F. Dupont, P. Blain, F. Roy, G.C. Smith, and G.M. Flato. Improving the simulation of landfast ice by combining tensile strength and a parameterization for grounded ridges. *J. Geophys. Res. Oceans*, 121:7354–7368, 2016. URL: http://dx.doi.org/10.1002/2016JC012006.

[25] J.F. Lemieux, D.A. Knoll, B. Tremblay, D.M. Holland, and M. Losch. A comparison of the Jacobian-free Newton Krylov method and the EVP model for solving the sea ice momentum equation with a viscous-plastic formulation: a serial algorithm study. *J. Comp. Phys.*, 231:5926–5944, 2012. URL: http://dx.doi.org/10.1016/j.jcp.2012.05.024.

[26] M. Leppäranta, A. Oikkonen, K. Shirasawa, and Y. Fukamachi. A treatise on frequency spectrum of drift ice velocity. *Cold Reg. Sci. Technol.*, 76-77:83–91, 2012. doi:http://dx.doi.org/10.1016/j.coldregions.2011.12.005.

[27] W.H. Lipscomb. Remapping the thickness distribution in sea ice models. *J. Geophys. Res. Oceans*, 106:13989–14000, 2001. URL: http://dx.doi.org/10.1029/2000JC000518.

[28] W.H. Lipscomb and E.C. Hunke. Modeling sea ice transport using incremental remapping. *Mon. Wea. Rev.*, 132:1341–1354, 2004. URL: http://dx.doi.org/10.1175/1520-0493(2004)132\T1\textless{}1341:MSITUI\T1\textgreater{}2.0.CO;2.

[29] W.H. Lipscomb, E.C. Hunke, W. Maslowski, and J. Jakacki. Ridging, strength, and stability in high-resolution sea ice models. *J. Geophys. Res. Oceans*, 2007. URL: http://dx.doi.org/10.1029/2005JC003355.

[30] G.A. Maykut and N. Untersteiner. Some results from a time dependent thermodynamic model of sea ice. *J. Geophys. Res.*, 76:1550–1575, 1971. URL: http://dx.doi.org/10.1029/JC076i006p01550.

[31] A.A. Mirin and P.H. Worley. Improving the Performance Scalability of the Community Atmosphere Model. *Int. J High Perform. Comput. Appl*, 26(1):17–30, 2012. URL: http://dx.doi.org/10.1177/1094342011412630.

[32] R.J. Murray. Explicit generation of orthogonal grids for ocean models. *J. Comput. Phys.*, 126:251–273, 1996. URL: http://dx.doi.org/10.1006/jcph.1996.0136.

[33] D. Notz, A. Jahn, E. Hunke, F. Massonnet, J. Stroeve, B. Tremblay, and M. Vancoppenolle. The CMIP6 Sea-Ice Model Intercomparison Project (SIMIP): understanding sea ice through climate-model simulations. *Geosci. Model Dev.*, 9:3427–3446, 2016. URL: http://dx.doi.org/10.5194/gmd-9-3427-2016.

[34] C.L. Parkinson and W.M. Washington. A large-scale numerical model of sea ice. *J. Geophys. Res. Oceans*, 84(C1):331–337, 1979. URL: http://dx.doi.org/10.1029/JC084iC01p00311.

[35] D.J. Pringle, H. Eicken, H.J. Trodahl, and L.G.E. Backstrom. Thermal conductivity of landfast Antarctic and Arctic sea ice. *J. Geophys. Res. Oceans*, 2007. URL: http://dx.doi.org/10.1029/2006JC003641.

[36] A. Roberts, E.C. Hunke, R. Allard, D.A. Bailey, A.P. Craig, J. Lemieux, and M.D. Turner. Quality control for community-based sea-ice model development. *Philos. Trans. Royal Soc. A*, 2018. URL: http://dx.doi.org/10.1098/rsta.2017.0344.

[37] A.F. Roberts, A.P. Craig, W. Maslowski, R. Osinski, A.K. DuVivier, M. Hughes, B. Nijssen, J.J. Cassano, and M. Brunke. Simulating transient ice-ocean Ekman transport in the Regional Arctic System Model and Community Earth System Model. *Ann. Glaciol.*, 56(69):211–228, 2015. URL: http://dx.doi.org/10.3189/2015AoG69A760.

[38] A. Rosati and K. Miyakoda. A general circulation model for upper ocean simulation. *J. Phys. Oceanogr.*, 18:1601–1626, 1988. URL: http://dx.doi.org/10.1175/1520-0485(1988)018\T1\textless{}1601:AGCMFU\T1\textgreater{}2.0.CO;2.

[39] D.A. Rothrock. The energetics of plastic deformation of pack ice by ridging. *J. Geophys. Res.*, 80:4514–4519, 1975. URL: http://dx.doi.org/10.1029/JC080i033p04514.

[40] E.M. Schulson. Brittle failure of ice. *Eng. Fract. Mech.*, 68:1839–1887, 2001. URL: http://dx.doi.org/10.1016/S0013-7944(01)00037-6.

[41] R.D. Smith, S. Kortas, and B. Meltz. *Curvilinear coordinates for global ocean models*. Technical Report LA-UR-95-1146, Los Alamos National Laboratory, 1995. URL: https://github.com/CICE-Consortium/CICE/blob/master/doc/PDF/LAUR-95-1146.pdf.

[42] A.H. Stroud. *Approximate Calculation of Multiple Integrals*. Prentice-Hall, 1971. Englewood Cliffs, New Jersey.

[43] K.E. Taylor. Summarizing multiple aspects of model performance in a single diagram. *J. Geophys. Res. Atmos.*, 106(D7):7183–7192, 2001. URL: http://dx.doi.org/10.1029/2000JD900719.

[44] A.S. Thorndike, D.A. Rothrock, G.A. Maykut, and R. Colony. The thickness distribution of sea ice. *J. Geophys. Res.*, 80:4501–4513, 1975. URL: http://dx.doi.org/10.1029/JC080i033p04501.

[45] M. Tsamados, D.L. Feltham, and A.V. Wilchinsky. Impact of a new anisotropic rheology on simulations of Arctic sea ice. *J. Geophys. Res. Oceans*, 118:91–107, 2013. URL: http://dx.doi.org/10.1029/2012JC007990.

[46] H. von Storch and F.W. Zwiers. *Statistical Analysis in Climate Research*. Cambridge University Press, 1999. Cambridge, UK.

[47] J. Weiss and E.M. Schulson. Coulombic faulting from the grain scale to the geophysical scale: lessons from ice. *J. of Phys. D: Appl. Phys.*, 42:214017, 2009. URL: http://dx.doi.org/10.1088/0022-3727/42/21/214017.

[48] A.V. Wilchinsky and D.L. Feltham. Dependence of sea ice yield-curve shape on ice thickness. *J. Phys. Oceanogr.*, 34:2852–2856, 2004. URL: http://dx.doi.org/10.1175/JPO2667.1.

[49] A.V. Wilchinsky and D.L. Feltham. Modelling the rheology of sea ice as a collection of diamond-shaped floes. *J. Non-Newtonian Fluid Mech.*, 138:22–32, 2006. URL: http://dx.doi.org/10.1016/j.jnnfm.2006.05.001.

[50] D.S. Wilks. *Statistical methods in the atmospheric sciences*. Academic Press, 2006. 2nd ed.

[51] S. T. Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. *J. Comp. Phys.*, 31(3):335–362, 1979. URL: http://dx.doi.org/10.1016/0021-9991(79)90051-2.

[52] F.W. Zwiers and H. von Storch. Taking serial correlation into account in tests of the mean. *J. Climate*, 8(2):336–351, 1995. URL: http://dx.doi.org/10.1175/1520-0442(1995)008\T1\textless{}0336:TSCIAI\T1\textgreater{}2.0.CO;2.